[redacted]

A CANONIC TRANSLATOR

by

Joseph Wright Alsop

Submitted in Partial Fulfillment
of the Requirements for the
Degree of Bachelor of Science
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1967

Signature of Author ______________________________
Department of Electrical Engineering
19 May 1967

Certified by ______________________________
Thesis Supervisor

Accepted by ______________________________
Head of Department

Acknowledgement

A heartfelt note of thanks is due my thesis supervisor. Not only has Professor Donovan's work provided the motivation for the work herein described; his suggestions, ideas and enthusiasm have made possible a successful conclusion to this thesis.

I am indebted to Professor Joseph Weizenbaum for the use of the SLIP system and most particularly for the time he took to explain the details of its use.

Summary

An algorithm to recognize and translate sets of character strings specified by canonic systems is presented. The ability of canonic systems to define the context sensitive features of strings and to specify their translation allows the algorithm to recognize and translate real computer languages. It is also applicable in other language systems.

Canonic systems are discussed, and several examples of their use are given. The algorithm is described, and examples of canonic translation are presented using a program which implements it.

Figures and Diagrams

A Canonic Translator

The development of a generalized compiler whose function is directed by a formal language specification has aroused significant interest and effort. This thesis presents an algorithm for the recognition and translation of character strings belonging to a set of strings whose syntax and translation have been defined by a canonic system. Since these systems are capable of defining context sensitive features of language, the algorithm can recognize and translate real computer languages. It is applicable to an even wider class of language systems, including boolean algebras and theorem proving, which can be characterized by this method.

Canonic systems form the basis and motivation for this work. The first task of the paper is to discuss briefly and informally the improved specification of syntax and translation made possible by the development of canonic systems. The discussion includes a description of the form of the systems and several examples, among them a complete formal description for the syntax of the string processing language SNOBOL. The contribution of this thesis lies in the presentation of an explicit algorithm which employs a canonic system characterizing the syntax and translation of a set of source strings to recognize a particular source string and perform the translation. The latter part of the thesis describes the algorithm and the program which implements it.

I. Formal Syntax Specifications

Backus-Naur Form is the most widely known formal specification of syntax. It provides a convenient starting point for a discussion of canonic systems. The general form of a rule or production of a BNF specification is as follows:

<name 1> ::= terminal 10 <name 11> ... <name 1n> terminal 1n |
terminal 20 <name 22> ... <name 2m> terminal 2m | ....

The sign ::= should be read "may be replaced by" and the vertical bar represents "or". The names enclosed within brackets are arbitrary designations for defined sets of strings. The definition may be recursive; that is, the set on the left may be defined in terms of itself if the name of the set also appears on the right. "terminal n m" designates an arbitrary string of terminal characters, possibly the null string. As a concrete example, consider the following BNF system.

<assignment> ::= <letter> = <expression>
<expression> ::= <letter> | <letter> + <expression>
<letter> ::= X | Y | Z

An example of a string which is a member of the set <assignment> is:

Y = X + Z

The strings comprising a set defined by a BNF system normally appear to be generated in a "top-down" manner. The highest level definition (<assignment>) is generally placed first, and one normally reads a BNF rule from left to right. In order to gain some insight into the form and nature of canonic systems without launching into a formal definition, consider turning a BNF production around and modifying the punctuation somewhat.

1. $v\ \underline{letter} \pitchfork x\ \underline{expression} \vdash v = x\ \underline{assignment}$

The lower case letters (v and x) are *variables* representing strings chosen from their respective sets (*letter* and *expression*). The names of the sets are underlined and called *predicates*. The definition may be read very elaborately as follows: "If v represents a string chosen from the set *letter*, and if x represents a string chosen from the set *expression*, then the string formed by concatenating the string represented by v with an equals sign and the string represented by x is a member of the set *assignment*." The sign ꝗ acts as the conjunctive "and", and the sign ⊢ acts as an assertion sign. A string of variables and terminal characters (e.g. v=x) is a *term*, and a term followed by a predicate in the manner above is a *remark*. Those remarks to the left of the assertion sign are referred to as *premises*; those to the right as *conclusions*. This example illustrates the most basic form of a *canon* in a canonic system. A more formal description may be found in Donovan (2) and Donovan and Ledgard (3). This discussion will remain highly informal.

What improvements in the definition of a syntax do canonic systems permit? The principal weakness of BNF systems is their inability to describe the context sensitive features of a set of strings; for example, the requirement in most computer languages that all reference labels of a program be singly defined as statement labels. This restriction could only be imposed in BNF notation by some process akin to defining each possible legal program, in toto, in a separate BNF rule. Certainly all sets of strings which can be defined in BNF may be defined by a canonic system by transforming the rules in the manner illustrated above. In addition, one may "cross-reference", or use a variable more than once on the left.

2. x *name* ꝗ x *label* ⊢ x *labelname*

Labelname is the intersection of *label* and *name*; that is, only those strings which are members of both the set *label* and the set *name* are members

[redacted]

and makes it possible to generate all ordered pairs with the property described.

A concrete example of the production of a particular member of a defined set will perhaps serve to clarify the nature and recursive properties of canonic systems. Assume we wish to show that $\langle A_{<} X, Y, \rangle$ is a member of the set notin. Using canon 4, we may assert

A *letter*.

We may then substitute this result into the premise of canon 7, and assert that

$\langle A_{<} \Lambda \rangle$ *notin*.

We then derive from canon 5 that

$\langle A_{<} X \rangle$ *differ*

$\langle A_{<} Y \rangle$ *differ*.

Finally, we apply canon 8 twice as follows:

$\langle A_{<} \Lambda \rangle$ *notin* ꝗ $\langle A_{<} X \rangle$ *differ* $\vdash$ $\langle A_{<} X, \rangle$ *notin*

$\langle A_{<} X, \rangle$ *notin* ꝗ $\langle A_{<} Y \rangle$ *differ* $\vdash$ $\langle A_{<} X, Y, \rangle$ *notin*

Note that we use the conclusion from the first application of the canon to establish the premise in the second application.

Now that the reader has grasped some of the power and elegance of canonic systems, a short history of their development is in order. This work is based completely upon the presentation of canonic systems by Donovan and Ledgard (3) and Donovan (2), who is responsible for their appearance in present form. His work evolved from an applied variant of Smullyan's elementary formal systems (6) and Post's canonical systems (4). The present canonic systems are so named in recognition of Post's work.

To further illustrate canonic systems, I present a complete syntactic definition of a restricted computer language MINI MAD. The present example and the foregoing example of *notin* both draw heavily from the examples presented in Donovan (2).

MINI MAD will permit only a few principal types of statements: an assignment statement, a transfer statement, and a statement formed by combining a simple conditional with one of the two other statements. All programs must terminate with an unlabeled END OF PROGRAM statement. The only boolean operator allowed is arithmetic equality (.E.), the only arithmetic operator allowed is addition (+), and only arbitrary length integers will be permitted as constants. The permissible statement labels are the single letters A, B and C; the variable names allowed are the letters X, Y and Z. In addition, restrictions on statement length will be omitted and no blanks will be allowed save those which are part of the statement definition (e.g. TRANSFER TO). The character * will be adopted as an end-of-card character, analogous to a carriage return. It should be understood that all restrictions and omissions are introduced for the sake of simplicity. A complete formal syntactic definition of the string-processing language SNOBOL may be found in appendix 2.

The following example is a member of the set *MINI MAD program*, with a carriage return substituted for the character *.

```
A    X=15
B    X=X+1
     WHENEVER X .E. 123, TRANSFER TO A
     TRANSFER TO B
```

Three canons will suffice to define the set of arbitrary length integers.

9. $\vdash$ 0▵1▵2▵ ... ▵8▵9 *digit*

10. d *digit* $\vdash$ d *integer*

11. d *digit* ꝗ i *integer* $\vdash$ di *integer*

The use of the predicate *notin*, defined previously, will later implement the restriction that no statement labels be multiply defined.

12. $\vdash \langle A_{<}B\rangle_{\vartriangle} \langle A_{<}C\rangle_{\vartriangle} \langle B_{<}C\rangle \ \underline{\text{differ}}$

13. $\langle x_{<}y\rangle \ \underline{\text{differ}} \vdash \langle y_{<}x\rangle \ \underline{\text{differ}}$

14. $d \ \underline{\text{digit}} \vdash \langle d_{<}\Lambda\rangle \ \underline{\text{notin}}$

15. $\langle x_{<}y\rangle \ \underline{\text{notin}} \ ¢ \ \langle x_{<}d\rangle \ \underline{\text{differ}} \vdash \langle x_{<}d, y\rangle \ \underline{\text{notin}}$

One should keep in mind that only lower case letters are used as variables representing strings. The signs $\vdash$, $¢$, $<$, $\vartriangle$ are punctuation signs in the canonic system itself. All other characters are drawn from the alphabet of the language being defined.

The definition of the predicate *in* will serve to implement the restriction that all reference labels be defined. The set *in* will consist of pairs of letter lists such that all letters in the first list appear somewhere in the second list. If the list of reference labels and the list of statement labels in a program satisfy this relationship, we know that there is at least one statement label corresponding to every reference label.

16. $\vdash A_{\vartriangle}B_{\vartriangle}C \ \underline{\text{label}}$

17. $\vdash \langle \Lambda_{<}\Lambda \rangle \ \underline{\text{in}}$

18. $\langle x_{<}y\rangle \ \underline{\text{in}} \ ¢ \ \ell \ \underline{\text{label}} \vdash \langle x_{<}\ell_1 y\rangle \ \underline{\text{in}}$

19. $\langle x_{<}y\rangle \ \underline{\text{in}} \ ¢ \ \ell \ \underline{\text{label}} \vdash \langle \ell_1 x_{<}\ell_1 y\rangle \ \underline{\text{in}}$

20. $\langle x_{<}y\rangle \ \underline{\text{in}} \ ¢ \ \langle z_{<}y\rangle \ \underline{\text{in}} \vdash \langle xz_{<}y\rangle \ \underline{\text{in}}$

Canon 17 provides a simple starting point for the recursive production of the more elaborate members of *in*, and corresponds to a program with neither statement nor reference labels. The next two canons describe the ways in which one may add to the lists of statement and reference labels. We may of course add a label at will to the list of statement labels, and may add a label to the reference label list as long as we also add it to the list of statement labels. The last canon provides for multiple referencing of a statement label. Using canons 16 through 19 alone, it is not possible to produce the following member of *in*

< B, B,< A, B, C,>

We may define the set *expression* as follows.

21. ⊢ X▴Y▴Z *variable*

22. v *variable* ⊢ v *expression*

23. i *integer* ⊢ i *expression*

24. v *variable* ɸ x *expression* ⊢ v + x *expression*

25. i *integer* ɸ x *expression* ⊢ i + x *expression*

The predicate next defined, *conditional*, will permit us to transform any unconditional statement into a conditional statement when a string from the set is placed before the unconditional statement.

26. ⊢ Λ *conditional*

27. x *expression* ɸ y *expression* ⊢ WHENEVER X .E. Y, *conditional*

Canon 26 allows us to produce a string which leaves the statement unchanged. Canon 27 defines a set of strings which will change any unconditional MINI MAD statement (e.g. X = 3) into a conditional statement (e.g. WHENEVER X + Y .E. Z, X = 3).

The "building block" sets defined so far will permit us to define the set of MINI MAD programs in fairly short order. A convenient vehicle for the task is a predicate of order three. The first element of the ordered triplets which make up the set *program with label lists* will be a list, punctuated by commas, of all statement labels used. The third element will be a similar list of reference labels. The second element will be the string of statements in which these labels are used. Again, we begin with a convenient starting point for later recursion.

28. ⊢< Λ < Λ < Λ > *program with label lists*

29. <s< p <r> *program with label lists* ɸ v *variable* ɸ x *expression* ɸ c *conditiona*
    ⊢ <s<   CV =X *p< r   *program with label lists*

30. <s< p <r> *program with label lists* ɸ ℓ *label* ɸ v *variable* ɸ x *expression* ɸ
    c *conditional* ɸ<ℓ< s> *notin*
    ⊢<ℓ_1s<ℓ CV=V *p<r> *program with label lists*

Canons 29 and 30 describe the way in which we may add an assignment statement, either conditional or unconditional. Using the first canon of the two, we may add an unlabeled assignment statement; using the second, we may add a labeled statement. Note that the use of <u>notin</u> in canon 30 imposes the restriction that the label used must not be in the list of previous statement labels.

30. $\langle s_{<} p_{<} r \rangle$ <u>program with label lists</u> ɸ ℓ <u>label</u> ɸ c <u>conditional</u>
   $\vdash$ $\langle s_{<}$ C TRANSFER TO ℓ* $p_{<}$ ℓ,r$\rangle$ <u>program with label lists</u>

31. $\langle s_{<} p_{<} r \rangle$ <u>program with label lists</u> ɸ $\ell_{\Delta} m$ <u>label</u> ɸ c <u>conditional</u> ɸ
   $\langle m_{<} s$ <u>notin</u>
   $\vdash$ $m_1 s_{<} m$ C TRANSFER TO ℓ* $p_{<}$ $\ell_1 r$ <u>program with label lists</u>

These two canons allow use to construct strings which include labeled and unlabeled, conditional and unconditional transfer statements in a manner analogous to that of the preceding pair of canons. We now need but one more canon to produce strings which are legal MINI MAD programs.

32. $\langle s_{<} p_{<} r \rangle$ <u>program with label lists</u> ɸ $\langle r_{<} s \rangle$ <u>in</u>
   $\vdash$ p END OF PROGRAM* <u>MINI MAD program</u>

This canon insures that all reference labels in the members of the set <u>MINI MAD program</u> are defined, and that all programs are properly terminated. This completes one of many possible canonic system definition or programs in MINI MAD. The canons are collected in sequence in appendix 1.

If the reader has clearly understood the manner in which canons may define, by production, the syntax of real computer languages, one further illustration may provide some insight into the manner in which these systems may also define translation. Assume one wishes to translate MINI MAD into another language, for instance an assembly language such as FAP. In order to accomplish this, one might expand <u>program with label lists</u> to include a fourth term which would contain the translation of the string of statements.

The canon for an unconditional, unlabeled TRANSFER TO statement might appear as follows.

33. $< s_{<} p_{<} r_{<} t >$ program with label lists and translation $\dashv$ ℓ label
$\vdash$ $< s_{<}$ TRANSFER TO ℓ^{*} $p_{<} \ell_{1} r_{<} t$ TRA $\ell^{*} >$
program with label lists and translation

This possibility of canonic specification of translation will be pursued further in the description of the algorithm which forms the contribution of this thesis, to which I now turn.

II. The Recognition and Translation Algorithm

Canonic systems will prove very useful in explicitly and concisely defining sets of strings such as computer languages. Such definitions would eliminate many ambiguities existing in language manuals. These systems could prove of greater value, however, if a canonic system could be used as a basis for recognizing strings from the defined set. In addition, if the members of the defined set are ordered pairs, triplets, etc., the usefulness of canonic systems would be still further extended if the algorithm could be used to produce the missing terms corresponding to a given term. The remaining part of this thesis discusses such an algorithm, the program which implements it, and the nature of the constraints imposed on the canons in order that the program be able to interpret them.

This algorithm is an extension of the algorithm presented by Cheathem and Sattley (1), which is capable of recognizing strings produced by a Backus-Naur system. The modifications to their algorithm, which appears here in quite different form, reflect the greater power of canonic systems in defining strings. These modifications include mechanisms for handling predicates of degree greater than one, for properly interpreting the multiple use of a variable among the premises, and for generating the translation specified. In the case of a canonic system where all predicates are of degree one, and no "cross-referencing" is used, the algorithm operates in a manner almost identical to that of Cheatham and Sattley.

The program which embodies the algorithm divides into two parts. A preliminary phase checks the syntax of the canonic system used. It insures, for example, that all variables used in the conclusion of a canon are to be found in the premises, and that all predicates used as premises are defined somewhere as conclusions. Further restrictions, which will be clarified later, are imposed on the form of the canons and are checked at this point. The program then assembles the canons into a list structure which reflects their form and content, and control is passed to the evaluative phase of the program. The SLIP list-processing system, developed by Weizenbaum (7) vastly simplified the implementation of the algorithm.

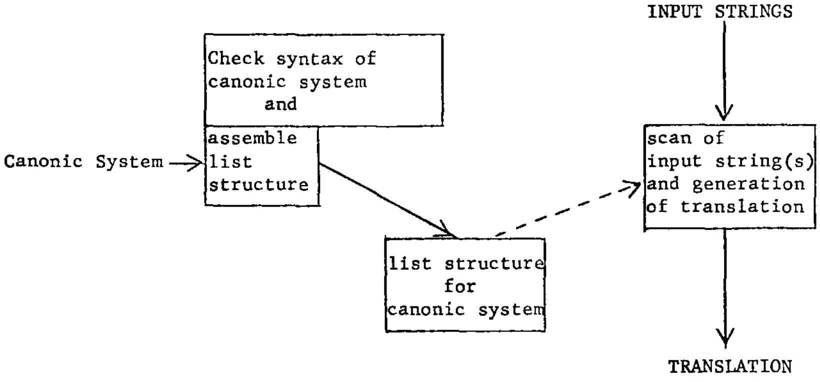


Fig. 1. Structure of Program

The second part of the program represents the principal programming effort. This phase scans the input string, determines whether it satisfies the canonic definition, and generates any associated translations. The algorithm is principally "top-down"; it attempts first to match the input string against the final predicate in the canonic system (e.g. <u>MINI MAD program</u>), and it arrives only through recursion at a lower-level predicate, (e.g. <u>integer</u> or <u>digit</u>). Consider the following simplified statement of the algorithm for the case of a canonic system involving only predicates of degree one. The simplified algorithm will be later expanded to include more general cases. Imagine an arbitrary character string, with a mental pointer to the left of the first character, and a canonic system defining a set of strings. We wish to determine whether the character string is a member of the set.

1. The program considers in sequence those canons directly defining the string in question, and performs the following steps (2 through 6) for each such canon.

2. The conclusion of the canon is matched, item by item, against the input string. If the item in the conclusion is a terminal character, step 3 is performed; if a variable, step 4 is performed. If the end of the canon is reached, the algorithm proceeds to step 5.

3. The item in the conclusion is a terminal character. It is compared with the character in the input string to the right of the mental pointer. If they are identical, the program returns to step 2 to consider the next item in the conclusion, with the pointer shifted one position to the right. If not, the scan fails and the program returns to step 1 to consider any remaining canons for the string.

4. The item in the conclusion is a variable, and the program must operate recursively to determine the definition of the variable in terms of the input string. In other words, it must determine the number of characters from the input string, commencing with the character to the right of the pointer, which should be alloted to the definition of this variable. To accomplish this, the program assembles a new input string which is a copy of all input characters to the right of the pointer, and picks a predicate among the premises of the canon which contains the variable. After saving its present state, the program returns to step 1 to determine the definition of the variable by examining the canons defining the premise predicate chosen. If there is no response upon return, the scan fails and the program returns to step 1 to consider alternative definitions of the string.

If there is a response, the program conpares it with the original input string to determine the definition of the variable and moves the mental pointer to its new position following the definition of the variable. The algorithm returns to step 2.

5. The scan of the conclusion is complete, and the definitions, in terms of the input characters, of the variables appearing in the conclusion have been recorded. The algorithm now inspects the premises. Those premises used in step 4 to determine the definitions of the variables in the conclusion may already be asserted, since they were used to generate the definitions. However, a variable may appear twice in the premises, and we must insure that the string which forms the definition of the variable is a member of both sets. The algorithm forms an input string from the definition of the variable and operates recursively to determine if the other premise containing the variable is also true; i.e., if the string which is the definition of the variable is also a member of the second set named as a premise predicate. Upon return, if there is no response, the algorithm returns to step 1 to pursue alternatives as before. If there is a response, the program insures that the string has been fully scanned. If there are still more unchecked premises, it treats them in the same manner. After all such premises have been successfully verified, the simplified algorithm proceeds to the last step.

6. The results of the scan at this level, which constitute the response for the next higher level, are assembled. There are no results if the scan failed. Otherwise, they consist of the input string with the mental pointer resting at the point where the scan of the conclusion was completed. The algorithm now returns to step 1, if there are more canons directly defining the set of which the input string is possibly a member. Since

each canon could conceivably add to the results, the program must actually be equipped to handle multiple results and hence multiple responses at the next higher level, and check out each possibility. The example which follows will serve to clarify the problem. If there are no further canons, the program proceeds to step 7.

7. The program "pops" its state; that is, it returns to pick up where it left off at the next higher level. If the highest level has been reached, then the results are examined for a completely scanned input string. If such a response is found, the input string is a member of the originally defined set. If not, there exists a syntax error in the string. It is not clear that the set of all syntactically incorrect sets will be recognized by the algorithm. This recognition may be unsolvable in general. The algorithm is flowcharted below.

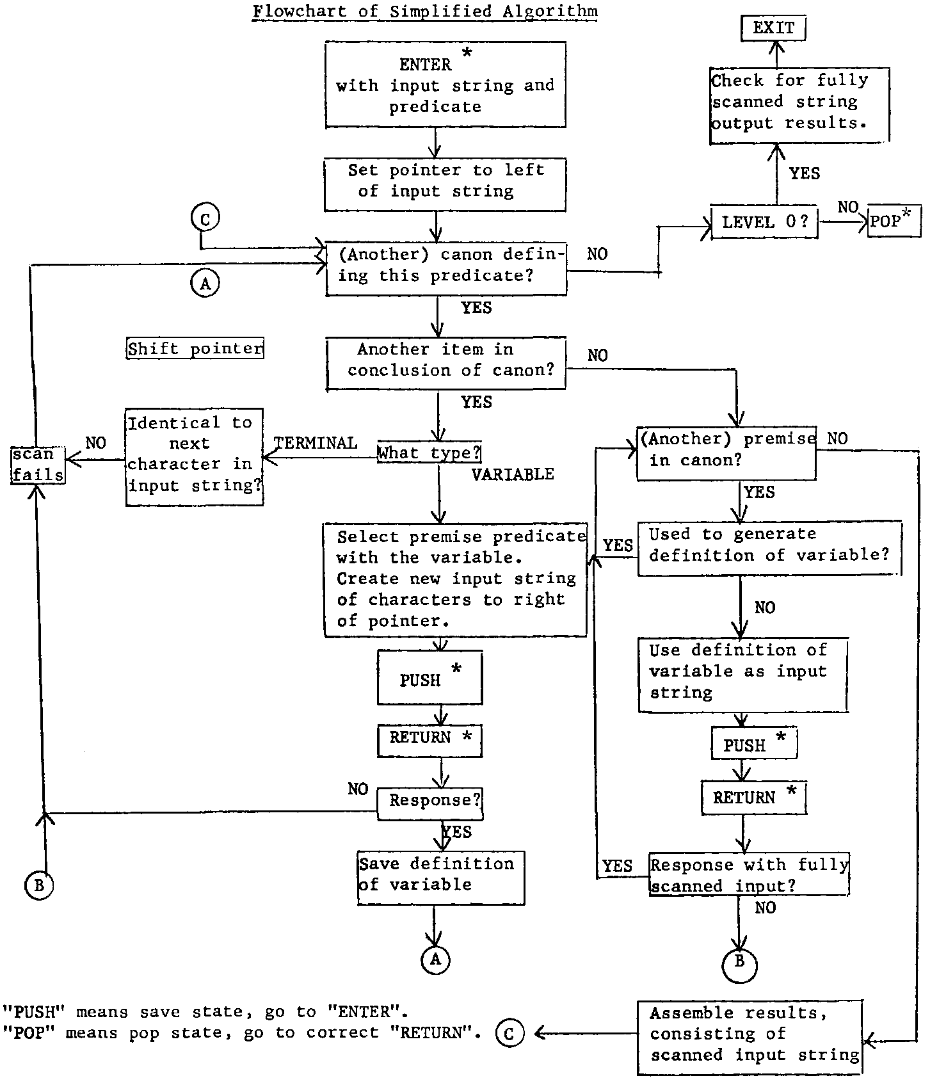
A simple example will serve to illustrate the process and the problems involved in multiple answers. Consider the following canonic system.

34. $\vdash$ 1 *digit*
35. $\vdash$ 2 *digit*
36. $\vdash$ 3 *digit*
37. d *digit* $\vdash$ d *integer*
38. d *digit* ϕ i *integer* $\vdash$ di *integer*

This system defines integers as arbitrary length strings of 1, 2 and 3. We wish to determine by use of the algorithm whether the string 31 is an integer. The process is described in the shorthand fashion below.

Step	Recursion Level	Input String	Canon Considered	Result(s)	Next Action
1	0	↓31	37	—	Push for digit
2	1	↓31	34	Fails	Next Canon
3	1	↓31	35	Fails	Next Canon
4	1	↓31	36	3↓1 digit	Pop
5	0	3↓1	37	3↓1 integer	Next Canon
6	0	↓31	38	—	Push for digit
7	1	↓31	34	Fails	Next Canon
8	1	↓31	35	Fails	Next Canon
9	1	↓31	36	3↓1 digit	Pop
10	0	3↓1	38	—	Push for integer
11	1	↓1	37	—	Push for digit
12	2	↓1	34	↓1 digit	Next Canon
13	2	↓1	35	Fails	Next Canon
14	2	↓1	36	Fails	Pop
15	1	1↓	37	1↓ integer	Next Canon
16	1	↓1	38	—	Push for digit
17	2	↓1	34	1↓ digit	Next Canon
18	2	↓1	35	Fails	Next Canon
19	2	↓1	36	Fails	Pop
20	1	1↓	38	—	Push for integer
21	2		37	—	Push for digit
22	3	↓	34	Fails	Next Canon
23	3	↓	35	Fails	Next Canon
24	3	↓	36	Fails	Pop
25	2	↓	37	Fails	Next Canon
26	2	↓	38	—	Push for digit
27	3	↓	34	Fails	Next Canon
28	3	↓	35	Fails	Next Canon
29	3	↓	36	Fails	Pop
30	2		38	Fails	Pop
31	1		38	1↓ integer	Pop
32	0		38	31↓ integer 3↓1 integer	Done

Flowchart of Simplified Algorithm



"PUSH" means save state, go to "ENTER".
"POP" means pop state, go to correct "RETURN".

At this point, the algorithm has arrived at two answers; i.e., that 3 and 31 are both integers. The first could not be immediately rejected because the algorithm has no global overview which informs it that there is no syntactic type following integer which would account for the rest of the string. At level zero however, we may eliminate such results, and the single assertion that 31 is indeed an integer remains.

We now consider the problem of left recursion. Suppose one wrote canon 38 in the following manner.

39. i integer ϕ d digit ⊢ id integer

The defined set integer has not been altered, but the algorithm will no longer function correctly. Note that whenever the program operates recursively to determine the definition of integer (steps 1, 10, 20), the length of the input string has been reduced by one character. Unless the scan proceeded from right to left, the program using the canon above would be caught in an endless loop, terminated only by the exhaustion of memory. Although it would be possible to devise a scheme to avoid the problem and still interpret the canon correctly, this would require some substantial effort which adds nothing to the scope or generality of this work. Instead, the canons are inspected for left recursion and rejected if it occurs. This constraint does not prevent the definition of any set of strings which could otherwise be defined.

The example brings out one other problem. At different points in the procedure (e.g. steps 42 and 43), the program must handle several possible answers which result from the various ways in which the canons may define the input. On a theoretical level this presents no problem, but in practice the manipulation of multiple large and nearly identical lists may exhaust memory. For this reason, one should follow two suggestions in using the system. Firstly, all syntactic types should be defined in as little context as possible, so that the legality of a particular string is immediately apparent, and does not depend on a construction occurring much further along in the input. In particular, the canonic system should not allow the input string to be parsed in several different ways, only to discover much later that only one is legal. To do so involves the risk of exhausting memory. Secondly, the canonic system should be unambiguous; that is, a particular string should be generated by only one production or path of application through the canons. Otherwise, both productions will give rise to results. Although the ambiguity could be eliminated by checking for identity among the results at any particular point, the comparisons would be extremely time consuming.

We turn now to an extension of the algorithm for the case in which we wish to consider evaluating a predicate of degree greater than one, for which one or more of the terms arenot known and are desired as translated output. The algorithm is presented at an arbitrary recursive level with input of arbitrary degree. For some of the input terms a character string is provided; some are merely marked "needed". Imagine a pointer positioned as before to the left of every term of the input set for which a character string is provided.

1. The program considers in sequence those canons directly defining the input in question, and performs the following steps (2 through 7) for each such canon.
2. The algorithm assembles a list of undefined variables which occur in those terms of the conclusion corresponding to "needed" terms in the input set. These are variables which would not normally be defined during the scan of the conclusion, but for which definitions must be obtained in order to generate the required translations. Variables appearing only in the premises of the canon and not in the conclusion are also added to the list.

3. The input strings provided are matched in sequence against the corresponding terms in the conclusion of the canon. The program skips conslusion terms corresponding to "needed" terms in the input set. If the item in the conclusion at any particular point is a terminal character, the algorithm performs step 4; if a variable, the algorithm performs step 5. When the scan of a term is complete, the program leaves the pointer where it rests and proceeds to the next term for which input is provided. When all such terms are scanned, the algorithm proceeds to step 6.

4. The item in the conclusion is a terminal character. It is compared with the character to the right of the pointer in the input string. If they are identical, the program returns to step 3 with the pointer shifted right one position. If they differ, the scan fails at this point and the algorithm returns to step 1 to pursue alternative definitions for the input.

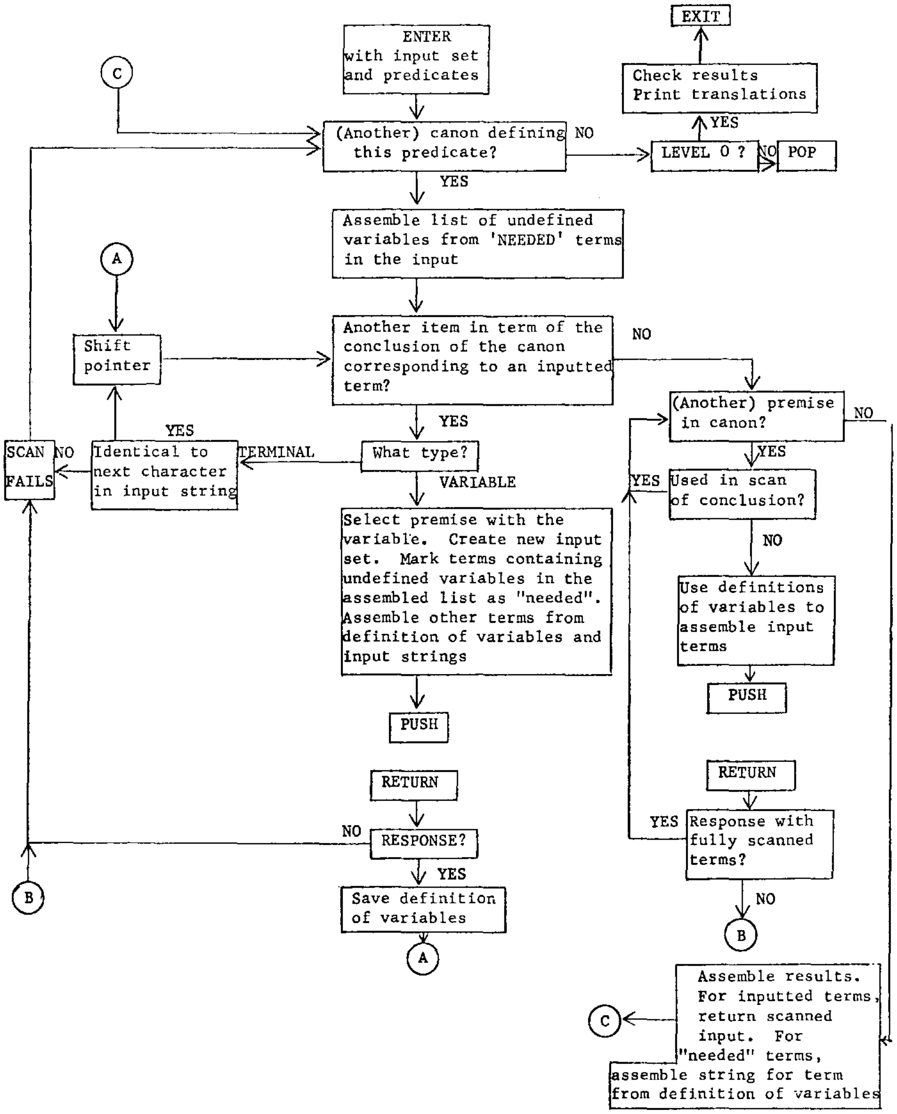
5. The item in the conclusion is a variable, and the algorithm must operate recursively to determine its definition. The program assembles a new input sequence from one of the premises in which the input appears. For the other terms in the premises, it assembles a character string if the variables therein have been defined. If one or more of the variables is undefined and in the "needed" list, it marks the term as "needed". Otherwise, the term is marked as unneeded. The program saves its state and returns to step 1 with the assembled input set for the chosen premise predicate. Upon return, if there is no response, the scan fails. If there is a response, the pointer of the input string is advanced accordingly, the definition of the undefined variables recorded, and the algorithm returns to step 3.

6. The scan of the conclusion is complete. Those premises which were not employed during the scan to generate definitions must now be verified. For these premises, the proper input strings for the terms are assembled

from the now-defined variables, and the algorithm operates recursively to determine whether the premise is satisfied. When all unchecked premises have been satisfied, the algorithm proceeds to the final step. If the return from recursion produces no response, or an input string not fully scanned, the scan fails and the algorithm returns to step 1 to consider any remaining canons.

7. If the scan succeeded, the results for the next higher level of recursion are assembled. For each term given as a string, the string is returned with the mental pointer moved to a position following the last character inspected in the conclusion scan. For each "needed" term, the definition of the term is assembled from the terminal characters and the now-defined variables in that term of the conclusion. If there are more canons to be considered, the algorithm returns to step 1. If not at level 0, the program then pops to the next higher level. If the zero recursion level has been reached, the evaluation is nearly complete. The results are checked to determine if there is a response in which all given terms have been fully scanned. If so, the "needed" terms are outputted. If not, there is a syntax error in the input. The expanded algorithm is presented as a flowchart below.

Flowchart of General Algorithm



A step-by-step example such as the previous table would be unduly lengthy when considering a non-trivial evaluation of a predicate of degree greater than one. Instead, consider as an example the action of the algorithm at the highest level of recursion as it seeks to determine whether an input string is a legal MINI MAD program. The only relevant canon is the last one.

40. $\langle s_{<} p_{<} r \rangle$ program with label lists $\langle r_{<} s \rangle$ in
$\vdash$ p END OF PROGRAM * MINI MAD program

The algorithm is presented with an input string which is possibly a member of the set MINI MAD program. Before beginning to scan the input, the program determines that s and r cannot be defined in terms of the input, and places these variables in the undefined list. It then begins the match of the input string against the conclusion of the canon. Since the item in the conclusion is a variable, it turns to the first premise, which contains p as a variable, in order to determine the definition of p in terms of the input string. Since s and r are in the undefined list, it marks these terms as "needed", and operates recursively to determine whether p is valid, and to produce s and r. The algorithm is presented at the next lower level with an ordered triplet in which the first and third elements are "needed" flags, and the second element an input string. If the input is indeed valid, excluding the requirement that all reference labels be defined, the algorithm will scan the input string at progressively deeper levels of recursion, eventually parsing out the statement labels, the various statements, etc. Since the first and third terms of program with label lists are "needed", it will build up these terms from the various statement and reference labels in the program, as directed by the canons which define program with label lists. Eventually, the algorithm will return to level zero. If there is no response returned from the lower level, the scan failed. If there is a response, it will consist of the input string with the pointer shifted to the right, and the accompanying lists which comprise the first and third terms of program with label lists. The remaining part of the input string is then checked to see whether it consists of END OF PROGRAM*. S and r are now defined. In order to verify the second premise, the algorithm assembles an input set from r and s, and operates recursively to determine if the two lists satisfy the relationship in. Upon return, if there is a response, the program checks to see that both terms are fully scanned; that is, that the definitions of r and s agree in both premises. Since both premises are now satisfied, the algorithm returns the scanned input string as a response. The program is at level 0, and control is given to a final routine which insures that, if there is a response, the input string has been fully scanned. The routine prints out a message to the effect that the input was or was not legal MINI MAD.

We turn now to the problems which may be encountered in evaluating the input in this manner. The potentially most disastrous problem is that of deciding how to generate the definition of variables not defined by the input. In the example above, there is no deterministic way of discovering from the one canon alone why the algorithm should not employ the second premise to generate the label lists. In this case, both terms of an input set would be marked "needed", and the canon would operate recursively to determine the members of the set. The definitions would be inserted, one at a time, into the first premise until the correct ordered pair for the particular input were found. Unfortunately in is an infinite set. Thus, if both terms are marked "needed", the algorithm sets about generating all possible members of the set and speedily exhausts memory. On the other hand, when the definitions for r and s are determined in conjunction with the scan of p as terms in program with label lists, only one ordered pair of label lists will be produced and inserted in the second premise. A similar but less serious problem might arise in determining the definition of p, if there were more than one premise containing p. Again, the choice of one premise over the other as a vehicle for determining the definition of p might result in a markedly different number of returned responses. These problems have been solved by transferring the decision to the user, who indicates how the definition of a variable should be determined by marking one appearance of the variable in the premises with a prefixed dollar sign. When the program encounters the variable in the conclusion, it will employ the premise in which the variable appears with the dollar sign as the vehicle to determine its definition. If there is no dollar sign, the program uses the premise in which the variable first occurs. Similarly, when considering the other terms of the chosen premise, the algorithm will mark the term as needed only if the variables therein are prefixed with the dollar sign, or if there is no other term in which they appear.

Another simplification is introduced in order to ease the programming effort. The restriction that premise terms contain one and only one variable reduces the complexity of the list manipulation which the program must perform. Again, this does not prevent the definition of sets whose definition is otherwise possibly. The premises in the canonic system which defines MINI MAD contain one and only one variable. An important point is that with the restriction we have placed on canonic systems we have in no way diminished their power.

This completes the description of the algorithm on the procedural level. The details of the use of the program, with examples, are described in appendix 3. We now turn to the intriguing question of the practicality of the canonic translator as a useful compiler.

The present program is wholly experimental, and we intend to use it to study the translation process. Three limitations exclude it from serious consideration as a practical device.

1. Speed. The program runs, conservatively, over 1000 times more slowly than a normal compiler.

2. Limitations on input. The program cannot accomodate large quantities of input data.

3. Error indications. If the scan fails, the program pinpoints the last character inspected in the input string, but goes no further. Thus, only one syntax error is detected per compilation.

I feel these limitations can be overcome, and that an implementation of the algorithm might be extremely useful in acting as a trial compiler in the design of a language, or as a regular compiler for lesser used languages where the additional efficiency of a dedicated compiler is not worth the effort necessary to produce one. I shall not consider the use of the algorithm for other language systems, such as the proof of theorems in boolean algebra. The further restrictions imposed on the generality of the algorithm in order to overcome the three limitations will probably reduce its usefullness in other more exotic areas. The proposals follow in order of increasing returns and commensurate restrictions on the algorithm.

1. Redesign and rewrite the program in assembly language. The program as it now stands is the MAD language in neither elegantly designed nor brilliantly executed. The pressure of time and the necessity to have the program work no matter how clumsily, prevented extensive streamlining.

2. Develop, perhaps in conjunction with proposal 1, a list processing system or data structure designed specifically for the algorithm. The SLIP list-processing system is elegantly designed, but its generality necessarily reduces its efficiency for this task. Measure 1 and 2 might provide a five-fold increase in speed, and a doubling of input handling

capacity.

3. Presently, all strings must be members of defined sets in order for premises to be asserted. Consider placing the left of the assertion sign premises which are true if and only if the definition of the variables are *not* members of the defined sets. Presently, it requires on the order of 26 2/2 canons to define the predicate *differ* for all letters of the alphabet. By defining a predicate *same*, as below, one could reduce this to 27 canons.

$$\vdash \langle A_< A\rangle_{\triangle} \langle B_< B\rangle_{\triangle} \ \ldots\ {}_{\triangle}\langle Z_< Z\rangle;\ \underline{\text{same}}\ \ \langle x_< y\rangle \widetilde{\underline{\text{same}}} \vdash \langle x_< y\rangle \underline{\text{differ}}$$

The sign $\sim$ indicates that the ordered pair $x_< y$ must *not* be a member of the set *same* in order to be a member of the set *differ*. This procedure would involve problems in originally defining variables, but could be used in premises which would only be verified after the variables have been defined. A moderate increase in speed would result, but the mathematical basis for canonic systems might well be destroyed. The possible implications of such a modification are vast and unexplored.

4. The compilation of a program never produces two different translations. This fact raises questions about the efficiency of handling multiple results at many points in the procedure (e.g. in the example for the simplified algorithm). A program, at any point in the scan of the source statement, is either possibly syntactically valid or definitely invalid. The source statements cannot be construed in several different syntactically valid ways. Consider establishing the rule that the algorithm, at any point in the recursion, returns only the first valid definition it discovers for the predicate. Assume the definition of integer were as follows.

41. d *digit* ꝗ i *integer* $\vdash$ di *integer*

42. d *digit* $\vdash$ d *integer*

Note that the recursive definition precedes the simpler canon, and the program considers it first. The action of the algorithm will be such that it continually operates recursively,eliminating a digit at each level, until it encounters a character other than a digit. The algorithm then "backs-up" one level, considers the alternative definition, and returns only one answer - an integer of the longest possible length, which is the definition actually desired. The implications of such a restriction are vast. By suitably positioning the non-recursive canons, one immediately eliminates more than half of the searching the program must perform. More importantly, such a rule eliminates all the list manipulation and duplication the program must presently execute. The manipulations are largely responsible for the complexity and inefficiency of the present implementation. Finally, such a restriction eliminates much "back-tracking", and makes it possible to contemplate a single, top-to-bottom pass of the input from auxiliary storage. Likewise, only one set of translation and "needed" lists must be built up, and this makes it possible to arrange the lists in a more conventional and more efficient format. The careful and imaginative implementation of this restriction might improve the speed of compilation by a factor of 50, and make the input capacity of the program comparable to that of conventional compilers. The usefulness of the translator for more general purposes would be, however, severly restricted.

5. One might consider using external subroutines to perform those functions (e.g. in and notin) clumsily handled by an algorithm which must essentially reverse the canonic production of the defined strings. If, as a result of proposal 4, the lists were arranged in a more conventional fashion, such subroutines might be easily implemented.

6. Finally, "system predicates" might be useful. The implementation of the algorithm would consider such elementary predicates as letter, digit

and differ to be understood, so that they need not be defined. Determining that A differs from B by testing whether or not B is one of the other 25 letters is hardly an efficient procedure. Such a provision might greatly speed the compilation.

We have not considered the uses of the algorithm in areas other than language translation, and the implementation of some of these measures, particularly 4, would severely hamper the ability of the algorithm to perform the intent of the canonic system. Other measures, particularly 1 and 6, might still prove useful. Ihave also avoided proposing a means of dealing with the problem of error indications. This problem might well be the most difficult to solve, but should probably consist of mechanism whereby the algorithm backtracks one syntactic type (e.g. statement) from the one in which the error was detected, skips the syntactic type, and proceeds from there on. Such a procedure might well produce fast and efficient syntax error elimination similar to that produced by a normal compiler.

Canonic systems are extremely powerful mechanisms for the definition of complicated strings. The areas in which canonic systems are applicable, and the possibilities for future study, are both vast and exciting. The possibility of a truly practical generalized compiler implemented through canonic systems deserves further investigation.

Appendix 1.

A.Canonic system specification of the syntax of MINI MAD.

1. Digit $\vdash 0_{\Delta} 1_{\Delta} 2_{\Delta} \ldots {}_{\Delta}8_{\Delta}9\ \underline{\text{digit}}$

2. Integer $d\ \underline{\text{digit}} \vdash d\ \underline{\text{integer}}$

   $d\ \underline{\text{digit}} \pitchfork i\ \underline{\text{integer}} \vdash di\ \underline{\text{integer}}$

3. Label $\vdash A_{\Delta}\ B_{\Delta}\ C\ \underline{\text{label}}$

4. Differ $\vdash \langle A_{<}B\rangle_{\Delta}\ \langle A_{<}C\rangle\ \ \langle B_{<}C\rangle\ \underline{\text{differ}}$

   $\langle x_{<}y\rangle\ \underline{\text{differ}}\ -\ \langle y_{<}x\rangle\ \underline{\text{differ}}$

5. Notin $\ell\ \underline{\text{label}} \vdash \langle \ell_{<}\Lambda\rangle\underline{\text{notin}}$

   $\langle x_{<}y\rangle\ \underline{\text{notin}} \pitchfork \langle x_{<}\ell\rangle\ \underline{\text{differ}} \vdash \langle x_{<}\ell, y\rangle\ \underline{\text{notin}}$

6. In $\vdash \langle \Lambda_{<}\Lambda\rangle\ \underline{\text{in}}$

   $\langle x_{<}y\rangle\ \underline{\text{in}} \pitchfork \ell\ \underline{\text{label}} \vdash \langle x_{<}\ell, y\rangle\ \underline{\text{in}}$

   $\langle x_{<}y\rangle\ \underline{\text{in}} \pitchfork \ \underline{\text{label}} \vdash \langle \ell, x_{<}\ell, y\rangle\ \underline{\text{in}}$

   $\langle x_{<}y\rangle\ \underline{\text{in}} \pitchfork \langle z_{<}y\rangle\ \underline{\text{in}} \vdash \langle xz_{<}y\rangle\ \underline{\text{in}}$

7. Variable $\vdash X_{\Delta}\ Y_{\Delta}\ Z\ \underline{\text{variable}}$

8. Expression $v\ \underline{\text{variable}} \vdash v\ \underline{\text{expression}}$

   $i\ \underline{\text{integer}} \vdash i\ \underline{\text{expression}}$

   $v\ \underline{\text{variable}} \pitchfork x\ \underline{\text{expression}} \vdash v{+}x\ \underline{\text{expression}}$

   $i\ \underline{\text{integer}} \pitchfork x\ \underline{\text{expression}} \vdash i{+}x\ \underline{\text{expression}}$

9. Conditional $\vdash \Lambda\ \ \underline{\text{conditional}}$

   $x\ \underline{\text{expression}} \pitchfork y\ \underline{\text{expression}} \vdash \text{WHENEVER } x \text{ .E. } y,\ \underline{\text{conditional}}$

10. Program with label lists

    $\vdash \langle \Lambda_{<}\Lambda_{<}\Lambda\rangle\ \ \underline{\text{program with label lists}}$

    $\langle s_{<}p_{<}r\rangle\ \underline{\text{program with label lists}} \pitchfork v\ \underline{\text{variable}} \pitchfork x\ \underline{\text{expression}} \pitchfork$

    $c\ \underline{\text{conditional}} \vdash \langle s \qquad \text{CV=X * } p_{<}r\rangle\ \underline{\text{program with label lists}}$

$\langle s_< p_< r\rangle$ program with label lists ɸ $\langle \ell_< s\rangle$ notin ɸ
v variable ɸ x expression ɸ c conditional ⊦
$\langle \ell, s_<$ cv=x * p r$\rangle$ program with label lists

$\langle s_< p_< r\rangle$ program with label lists ɸ m label ɸ c conditional
⊦ $\langle s_<$ c TRANSFER TO m * $p_<$ m, r$\rangle$
program with label lists

$\langle s_< p_< r\rangle$ program with label lists ɸ m label ɸ $\langle \ell_< s\rangle$ notin
ɸ c conditional ⊦$\langle \ell$,$s_< \ell$ c TRANSFER TO m * $p_<$ m,s$\rangle$
program with label lists

11. MINI MAD program

$\langle s_< p_< r\rangle$ program with labellists ɸ $\langle r_< s\rangle$ in
⊦ p END OF PROGRAM * MINI MAD program

B. Canonic system specification of the syntax and translation of MINI MAD into PSEUDO FAP. The dollar sign in PSEUDO FAP indicates "this location".

1. Digit $\vdash\ 0 \triangle 1 \triangle 2 \triangle \ldots \triangle 8 \triangle 9\ \underline{\text{digit}}$

2. Integer $d\ \underline{\text{digit}} \vdash d\ \underline{\text{integer}}$
$d\ \underline{\text{digit}} \dashv i\ \underline{\text{integer}} \vdash di\ \underline{\text{integer}}$

3. Label $\vdash\ A \triangle B \triangle C\ \underline{\text{label}}$

4. Differ $\vdash\ \langle A_< B\rangle \triangle \langle A_< C\rangle \triangle \langle B_< C\rangle\ \underline{\text{differ}}$
$\langle x_< y\rangle\ \underline{\text{differ}} \vdash \langle y_< x\rangle\ \underline{\text{differ}}$

5. Notin $\ell\ \underline{\text{label}} \vdash \langle \ell_< \Lambda\rangle\ \underline{\text{notin}}$
$\langle x_< y\rangle\ \underline{\text{notin}} \dashv \langle x_< \ell\rangle\ \underline{\text{differ}} \vdash \langle x_< \ell,\ y\rangle\ \underline{\text{notin}}$

6. In $\vdash \langle \Lambda_< \Lambda\rangle\ \underline{\text{in}}$
$\langle x_< y\rangle\ \underline{\text{in}} \dashv \ell\ \underline{\text{label}} \vdash \langle x_< \ell,\ y\rangle\ \underline{\text{in}}$
$\langle x_< y\rangle\ \underline{\text{in}} \dashv \ell\ \underline{\text{label}} \vdash \langle \ell x_< \ell\ y\rangle\ \underline{\text{in}}$
$\langle x_< y\rangle\ \underline{\text{in}} \dashv \langle x_< y\rangle\ \underline{\text{in}} \vdash \langle xz_< y\rangle\ \underline{\text{in}}$

7. Variable $\vdash\ X \triangle Y \triangle Z\ \underline{\text{variable}}$

8. Expression $v\ \underline{\text{variable}} \vdash v_< \text{CLA}\quad v{*}\rangle\ \underline{\text{expression}}$
$i\ \underline{\text{integer}} \vdash i_< \text{CLA} = i\ {*}\rangle\ \underline{\text{expression}}$
$v\ \underline{\text{variable}} \dashv \langle x_< y\rangle\ \underline{\text{expression}} \vdash \langle v{+}x_< y\quad \text{ADD } v{*}\rangle\ \underline{\text{expression}}$
$i\ \underline{\text{integer}} \dashv \langle x_< y\rangle\ \underline{\text{expression}} \vdash \langle i{+}x_< y\quad \text{ADD } = i{*}\rangle\ \underline{\text{expression}}$

9. Conditional $\vdash \langle \Lambda_< \Lambda\rangle\ \underline{\text{conditional}}$

$\langle x_< y\rangle\ \underline{\text{expression}} \dashv \langle u_< v\rangle\ \underline{\text{expression}}$
$\vdash \langle \text{WHENEVER } x.\text{E}.\ u,\ _< y\quad \text{STO TMP*}$
$\text{SUB TMP*}\quad \text{TNZ \$+3}\rangle\ \underline{\text{conditional}}$

10. Program with translation

$\vdash \langle \Lambda_< \Lambda_< \Lambda_< \Lambda\rangle\ \underline{\text{program with translation}}$

<s< p<·r< t> program with translation ¢ v variable
¢<x< y> expression ¢ <c< d> conditional ⊢
<s< cv=x * p< r< y STO TNP *
d CLA TNP * STO V * t>
program with translation

<s< p< r< t> program with translation ⊢< ℓ< s > notin ¢
v variable ¢ <x y> expression ¢ <c< d> conditional
⊢ <ℓ , s< ℓ cv=x* p< r<
/ y STO TNP* d CLA TNP* STO V * t>

<s< p< r< t> program with translation ¢ m label <c< d> conditional ⊢
<s< c TRANSFER TO m * p< m, r< d TRA m * NOP * t>
program with translation

<s< p< r< t> program with translation ¢ m label ¢ < ℓ< s> notin ¢ c conditional
⊢<ℓ ,s< ℓ c TRANSFER TO m * p< m, s< ℓ d TRA m * NOP* t >
program with translation

11. MINI MAD - PSEUDO FAP

<s< p< r< t > program with translation ¢ <r< s> in
⊢ < p END OF PROGRAM*< t HLT*
TMP DEC* TNP DEC* END * >
MINI MAD - PSEUDO FAP

As an example, the program given previously in the text is reproduced below with the equivalent PSEUDO FAP program.

```
A     X = 15
B     X = X+1
      WHENEVER X .E. 123, TRANSFER TO A
      TRANSFER TO B

A     CLA =15
      STO X
B     CLA =1
      ADD X
      STO X
      CLA X
      STO TNP
      CLA =123
      SUB TNP
      TNZ $ + 3
      TRA A
      NOP
      TRA B
TMP   DEC
TNP   DEC
      HLT
      END
```

Appendix 2.

A canonic system specification for the syntax of SNOBOL.

The canonic system presented in this appendix defines the syntax of SNOBOL as implemented on the 7094 CTSS system at MIT. The language is used for string processing and contains statements for string matching, replacing, deleting and inserting. The language also has a few arithmetic capabilities. Those not familiar with the language may find reference 5 useful.

The canonic system is listed below. λ represents a space.

1. $\vdash A_\Delta B_\Delta C \ldots X_\Delta Y_\Delta Z\ \underline{\text{letter}}$
2. $\vdash 0_\Delta 1_\Delta 2 \ldots 7_\Delta 8_\Delta 9\ \underline{\text{digit}}$
3. $x\ \underline{\text{letter}} \dashv y\ \underline{\text{digit}} \vdash x_\Delta y_\Delta .\ \underline{\text{name character}}$
4. $x\ \underline{\text{name character}} \vdash x_\Delta ,_\Delta \text{'}_\Delta *_\Delta +_\Delta -_\Delta /_\Delta ?_\Delta =_\Delta \$\ \underline{\text{label character}}$
5. $x\ \underline{\text{name character}} \vdash x_\Delta ,_\Delta (_\Delta)_\Delta *_\Delta +_\Delta /_\Delta ?_\Delta =_\Delta \$\ \underline{\text{string character}}$
6. $x\ \underline{\text{string character}} \vdash x_\Delta \text{'}\ \underline{\text{character}}$
7. $\vdash +_\Delta -_\Delta /_\Delta *\ \underline{\text{operator}}$
8. $\vdash \downarrow\ \underline{\text{tab}}$
9. $\vdash$ ⤸ $\underline{\text{carriage return}}$
10. $x\ \underline{\text{spaces}} \vdash \lambda_\Delta\ x\lambda\ \underline{\text{spaces}}$
11. $a_\Delta b_\Delta c_\Delta d_\Delta e_\Delta f\ \underline{\text{name character}} \vdash a_\Delta ab_\Delta abc_\Delta abcd_\Delta abcde_\Delta abcdef\ \underline{\text{string name}}$
12. $x\ \underline{\text{string character}} \dashv y\ \underline{\text{string}} \vdash xy\ \underline{\text{string}}$
13. $x\ \underline{\text{string}} \vdash \text{'}x\text{'}\ \underline{\text{literal}}$
14. $x\ \underline{\text{letter}} \dashv y\ \underline{\text{digit}} \dashv z\ \underline{\text{label character}} \dashv a\ \underline{\text{label}} \vdash x_\Delta y_\Delta az\ \underline{\text{label}}$
15. $x\ \underline{\text{digit}} \dashv y\ \underline{\text{integer}} \vdash x_\Delta yx\ \underline{\text{integer}}$

16. x string name ¢ y literal ⊢ $x_\Delta y$ operand

17. $x_\Delta y$ operand ¢ z expression ¢ v operator ¢ s spaces
⊢ $xsvsy_\Delta\ xsvsz_\Delta\ zsvsy_\Delta\ (z)$ expression

18. x operand ¢ y expression ¢ z term ¢ s spaces
⊢ $x_\Delta\ y_\Delta\ zsx_\Delta\ zsy$ term

19. x term ⊢ $\Lambda_\Delta x$ concatenation

20. x string name ⊢ *x* variable name

21. $x_\Delta y$ string name ¢ z integer ⊢ $*x/y*_\Delta\ *x/'z'*$ fixed length name

22. x string name ⊢ *(x)* balanced name

23. x string name ¢ y literal ¢ $u_\Delta v$ term ¢ w indirect name
¢ s spaces ⊢ $\$x_\Delta\ \$y_\Delta\ \$(w)_\Delta\ \$(usw)_\Delta\ \$(wsu)_\Delta\ \$(uswsv)$ indirect name

24. $\langle A_< B\rangle_\Delta \langle A_< C\rangle_\Delta \cdots_\Delta \langle B_< C\rangle_\Delta \cdots_\Delta \langle =_< \$\rangle$ differ

25 $a_\Delta e_\Delta f$ string ¢ $\langle b_< c\rangle$ differ ⊢ $\langle ace_< abf\rangle$ different

26. $\langle x_< y\rangle$ different ⊢ $\langle y_< x\rangle$ different

27. x label ¢ y list ⊢ $\Lambda_\Delta yx\omega$ list

28. x list ⊢ $\langle \Lambda_< x\rangle$ in

29. $x_\Delta y$ list ¢ $\langle w_< xy\rangle$ in ¢ l label ⊢ $\langle wl\omega_< xl\omega y\rangle$ in

30. $\langle w_< xy\rangle$ in ¢ $\langle u_< xy\rangle$ in ⊢ $\langle wu_< xy\rangle$ in

31. x label ⊢ $\langle x_< \Lambda\rangle$ notin

32. $x_\Delta y$ label ¢ $\langle x_< y\rangle$ different ¢ $\langle x_< z\rangle$ notin ⊢ $\langle x_< zy\omega\rangle$ notin

33. x string name ¢ y concatenation ¢ s spaces
⊢ xs=sy assignment statement

34. x operand ¢ y expression ¢ u variable name ¢ v fixed length name
¢ w balanced name ¢ z indirect name ⊢ $u_\Delta\ v_\Delta\ w_\Delta\ x_\Delta\ y_\Delta\ z$
scan operand

35. x $\underline{\text{scan operand}}$ ₵z $\underline{\text{scan}}$ ₵s $\underline{\text{spaces}}$ $\vdash$ x_Δ xsz $\underline{\text{scan}}$

36. x $\underline{\text{operand}}$ ₵y $\underline{\text{concatenation}}$ ₵z $\underline{\text{scan}}$ ₵s $\underline{\text{spaces}}$
$\vdash$ xsz_Δ xszs=sy $\underline{\text{scan statement}}$

37. $\vdash$ -EJECT_Δ -LIST_Δ -NULLOP OP_Δ -PCC_Δ -SPACE_Δ -TITLE
$_\Delta$ -UNLIST $\underline{\text{control word}}$

38. x $\underline{\text{operand}}$ ₵a $\underline{\text{arguments}}$ $\vdash$ Λ_Δ x_Δ a,x_Δ x,a_Δ 'a'_Δ (a) $\underline{\text{arguments}}$

39. x $\underline{\text{string name}}$ ₵a $\underline{\text{arguments}}$ $\vdash$ x(a) $\underline{\text{string name}}$ ₵
x(a) $\underline{\text{system function}}$

40. x $\underline{\text{label}}$ ₵y $\underline{\text{indirect name}}$ $\vdash$ $\langle x_< x\omega\rangle_\Delta\langle y_<\Lambda\rangle$ $\underline{\text{reference label}}$

41. $\langle x_< y\rangle_\Delta\langle w_< z\rangle$ $\underline{\text{reference label}}$ $\vdash$ $\langle /(x)_< y\rangle_\Delta\langle /S(x)_< y\rangle_\Delta\langle /F(x)_< y\rangle$
$_\Delta\langle /S(x)F(w)_< yz\rangle$ $\underline{\text{branch}}$

42. x $\underline{\text{scan statement}}$ ₵y $\underline{\text{assignment statement}}$ ₵z $\underline{\text{system function}}$
$\langle u_< v\rangle$ $\underline{\text{branch}}$ ₵s $\underline{\text{spaces}}$ $\vdash$ $\langle x_<\Lambda\rangle_\Delta\langle y_<\Lambda\rangle_\Delta\langle xsu_< v\rangle$
$_\Delta\langle ysu_< v\rangle_\Delta\langle z_<\Lambda\rangle_\Delta\langle zsu_< v\rangle$ $\underline{\text{right hand side}}$

43. $\langle x_< y\rangle$ $\underline{\text{right hand side}}$ $\vdash$ $\langle \text{END}\downarrow x_< y\rangle_\Delta\langle \text{END}_<\Lambda\rangle$ $\underline{\text{end card}}$

44. $\vdash\langle\Lambda_<\Lambda_<\Lambda\rangle$ $\underline{\text{program string}}$

45. $\langle p_< q_< r\rangle$ $\underline{\text{program string}}$ ₵x $\underline{\text{control word}}$ $\vdash$ $\langle p_< qx\downarrow_< r\rangle$ $\underline{\text{program string}}$

46. $\langle p_< q_< r\rangle$ $\underline{\text{program string}}$ ₵ $\langle x_< y\rangle$ $\underline{\text{right hand side}}$ ₵u $\underline{\text{label}}$
₵ $\langle u_< p\rangle$ $\underline{\text{notin}}$ $\vdash$ $\langle p_< q\downarrow x\downarrow_< ry\rangle_\Delta\langle pu\omega_< qu\downarrow x\downarrow_< ry\rangle$ $\underline{\text{program string}}$

47. $\langle p_< q_< r\rangle$ $\underline{\text{program string}}$ ₵ $\langle \text{END}_< p\rangle$ $\underline{\text{notin}}$ ₵ $\langle x_< y\rangle$ $\underline{\text{end card}}$
₵ $\langle ry_< p\rangle$ $\underline{\text{in}}$ $\vdash$ $qx\downarrow$ $\underline{\text{program}}$

Appendix 3.

Use of Program.

The program which implements the algorithm allows the user to type in a series of canons defining a set of strings, followed by the input he wishes to have analyzed. The program then scans the input string or strings for correct syntax. If the input is syntactically correct, a message to this effect is printed. Further, if the input is defined as only one of several terms in the final predicate of the canonic system, the other terms corresponding to the input may be produced. If the scan fails, the program identifies the character in the input string which was the last character inspected.

The sequence of messages and the proper responses as the program executes on the MIT CTSS system are as follows.

INPUT CANONS.

A set of canons may now be input, subject to the restrictions described in the text and summarized briefly below.

1. Canons may contain only one conclusion.
2. The terms of the premise predicates may contain one and only one variable, and no terminal characters.
3. Left recursion in all terms of a predicate is not permitted. Partial left recursion evokes a warning message.

The user inputs the canons according to the following rules which implement the punctuation of the canonic system.

1. Strings of terminal characters must be enclosed in break characters

(' / or *).

2. The digit 1 and the digit 2 when not enclosed in breaks represent respectively a tab and a carriage return.

3. Letters represent variables. All variables used in the conclusion must be defined in the premises.

4. Predicate names must consist of six characters or less, and be enclosed in hyphens.

5. The terms of a predicate are separated by periods.

6. The premise remarks of a canon are separated by commas.

7. An equals sign replaces the assertion sign.

8. Spaces and carriage returns are ignored except when enclosed in breaks, but each line may not contain more than one canon.

The examples at the end of this appendix will serve to clarify the syntax rules.

After the last canon, the user types 'end' at the beginning of a line. The program responds with the following sequence after checking that all predicates used as premises are defined as conclusions.

```
CONSITENT SET OF CANONS.
LIST OF DEFINED PREDICATES AND DEGREES.
```

The predicates typed in are then listed in the order in which they first appeared.

```
INPUT OF SOURCE STRINGS.
TYPE FINAL PREDICATE.
```

The user responds by typing the predicate name which defines the input string he wishes the program to consider.

TYPE -NONEED-, -NEED- OR -INPUT- FOR EACH TERM.

TERM NUMBER n OF -predicate-

At this point the user declares which terms of the final predicate he wishes to input and which terms he desires as translation. 'Noneed' indicates that he wishes neither to input the term nor receive it as output. 'Need' indicates he wishes to receive the term as a translation. 'Input' means that he wishes to type in an input string for the term. In this case, the program responds.

INPUT STRING. EXTRA CARRIAGE RETURN INDICATES END.

The user may now type in input which will be verified for syntactic correctness, and for which the program will produce output corresponding to 'needed' terms. Carriage returns are counted as characters. If the user wishes instead to input card images, he may do so by typing in 80 characters or more. The input is truncated at 80 characters and in this case the carriage return will not be counted.

After all terms of the final predicate have been considered, the program types this message.

TYPE 0, 1 OR 2 FOR DEPTH OF COMMENTS.

The user responds by typing a single number. If 0, the program will print only the final results. If 1, it will remark on extraordinary conditions which occur. Typing 2 results in messages whenever the program "pops" or "pushes". A larger number will result in the output of various lists which comprise the intermediate results of the scan. These lists, while useful during program during program debugging, are rather incomprehensible except to those familiar with both the program and the SLIP system.

The program then types

SCAN BEGINS.

When the program returns to the zero level of recursion, it will type out the results of the analysis. If the scan succeeds, and if terms are 'needed', these terms are printed. If there is more than one translation, all will be printed. In the examples which follow, the execution time, which is printed in seconds at the end of the run, indicates the problems of execution speed to be overcome if one wishes to make a practical canonic translator.

There are three examples of canonic translation. The first is relatively simple. It illustrates a scheme for coding messages by replacing the letters in the message with their successors in the alphabet. The second example demonstrates the construction of an expression in MINI MAD and the corresponding PSEUDO FAP instructions. The third example, an extension of the second, demonstrates the construction of an assignment statement in MINI MAD and the translation into PSEUDO FAP. Note that no data cells were reserved, although this could have been easily implemented. A final example illustrates the error analysis of the program.

```
resume thesis
W 2121.4
INPUT CANONS.

= 'a'.'b' -pair-
= 'b'.'c' -pair-
= 'c'.'d' -pair-
= 'd'.'e' -pair-
= 'e'.'f' -pair-
= 'f'.'a' -pair-
x.y -pair- = x.y -code-
x.y -pair-, u.v -code- = xu.yv -code-
u.v -code- = u2 . v' is the coded message for 'u2 -messag-
end

CONSISTENT SET OF CANONS.

LIST OF DEFINED PREDICATES AND DEGREES.

  1.  -  PAIR-   2
  2.  -  CODE-   2
  3.  -MESSAG-   2

INPUT OF SOURCE STRINGS.
TYPE FINAL PREDICATE.
messag

TYPE -NONEED-, -NEED- OR -INPUT- FOR EACH TERM.

TERM NUMBER 1 OF -MESSAG-
input

INPUT STRING. EXTRA CARRIAGE RETURN INDICATES END.

abcdef


TERM NUMBER 2 OF -MESSAG-
need

TYPE 0, 1 OR 2 FOR DEPTH OF COMMENTS.
0

SCAN BEGINS.


SCAN SUCCESSFUL.
TRANSLATED OUTPUT (IF ANY) FOLLOWS.

TERM NUMBER 2.

BCDEFA IS THE CODED MESSAGE FOR ABCDEF

END OF RUN.
EXITM CALLED. GOODBYE.
R 7.150+6.983
```

```
resume thesis
W 2127.5
INPUT CANONS.

= 'x' -variab-
= 'y' -variab-
= 'z' -variab-
= '1' -digit-
= '2' -digit-
= '3' -digit-
d -digit- = d -integ-
d -digit-, i -integ- = di -integ-
i -integ- = i . '    cla ='i2 -expres-
v -variab- = v . '    cla 'v2 -expres-
i -integ-, x.y -expres- = i'+'x . y'    add ='i2 -expres-
WARNING- PARTIAL LEFT RECURSION IN LINE NUMBER 11
v -variab-, x.y -expres- = v'+'x . y'    add 'v2 -expres-
WARNING- PARTIAL LEFT RECURSION IN LINE NUMBER 12
x.y -expres- = x2 . 'this is the translation for 'x22
     y'    end'2 -exampl-
end

CONSISTENT SET OF CANONS.

LIST OF DEFINED PREDICATES AND DEGREES.

  1.  -VARIAB-   1
  2.  - DIGIT-   1
  3.  - INTEG-   1
  4.  -EXPRES-   2
  5.  -EXAMPL-   2

INPUT OF SOURCE STRINGS.
TYPE FINAL PREDICATE.
exampl

TYPE -NONEED-, -NEED- OR -INPUT- FOR EACH TERM.

TERM NUMBER 1 OF -EXAMPL-
input

INPUT STRING. EXTRA CARRIAGE RETURN INDICATES END.

x+123+y+321+z


TERM NUMBER 2 OF -EXAMPL-
need

TYPE 0, 1 OR 2 FOR DEPTH OF COMMENTS.
0
```

```
SCAN BEGINS.


SCAN SUCCESSFUL.
TRANSLATED OUTPUT (IF ANY) FOLLOWS.

TERM NUMBER 2.

THIS IS THE TRANSLATION FOR X+123+Y+321+Z

     CLA Z
     ADD =321
     ADD Y
     ADD =123
     ADD X
     END

END OF RUN.
EXITM CALLED. GOODBYE.
R 16.733+8.383
```

```
resume thesis
W 2150.5
INPUT CANONS.

= 'x' -variab-
= 'y' -variab-
= 'z' -variab-
= '1' -digit-
= '2' -digit-
= '3' -digit-
d -digit- = d -integ-
d -digit-, i -integ- = di -integ-
i -integ- = i . '    cla ='i2 -expres-
v -variab- = v . '    cla 'v2 -expres-
i -integ-, x.y -expres- = i'+'x . y'    add ='i2 -expres-
WARNING- PARTIAL LEFT RECURSION IN LINE NUMBER 11
v -variab-, x.y -expres- = v'+'x . y'    add 'v2 -expres-
WARNING- PARTIAL LEFT RECURSION IN LINE NUMBER 12
v -variab-, x.y -expres- = v'='x . y'    sto 'v2 -assign-
x.y -assign- = x2 . 'this is the translation for 'x22
      y'    end'2 -exampl-
end

CONSISTENT SET OF CANONS.

LIST OF DEFINED PREDICATES AND DEGREES.

  1.  -VARIAB-   1
  2.  - DIGIT-   1
  3.  - INTEG-   1
  4.  -EXPRES-   2
  5.  -ASSIGN-   2
  6.  -EXAMPL-   2

INPUT OF SOURCE STRINGS.
TYPE FINAL PREDICATE.
exampl
```

```
TYPE -NONEED-, -NEED- OR -INPUT- FOR EACH TERM.

TERM NUMBER 1 OF -EXAMPL-
Input

INPUT STRING. EXTRA CARRIAGE RETURN INDICATES END.

y=x+123+y+3211+z


TERM NUMBER 2 OF -EXAMPL-
need

TYPE 0, 1 OR 2 FOR DEPTH OF COMMENTS.
0

SCAN BEGINS.


SCAN SUCCESSFUL.
TRANSLATED OUTPUT (IF ANY) FOLLOWS.

TERM NUMBER 2.

THIS IS THE TRANSLATION FOR Y=X+123+Y+3211+Z

    CLA Z
    ADD =3211
    ADD Y
    ADD =123
    ADD X
    STO Y
    END

END OF RUN.
EXITM CALLED. GOODBYE.
R 18.866+8.400
```

```
resume thesis
W 2200.4
INPUT CANONS.

= 'this is a test sentence'2 -exampl-
end

CONSISTENT SET OF CANONS.

LIST OF DEFINED PREDICATES AND DEGREES.

  1.  -EXAMPL-    1

INPUT OF SOURCE STRINGS.
TYPE FINAL PREDICATE.
exampl

TYPE -NONEED-, -NEED- OR -INPUT- FOR EACH TERM.

TERM NUMBER 1 OF -EXAMPL-
input

INPUT STRING. EXTRA CARRIAGE RETURN INDICATES END.

this is not a test sentence


TYPE 0, 1 OR 2 FOR DEPTH OF COMMENTS.
0

SCAN BEGINS.


SCAN FAILED.  SYNTAX ERROR IN INPUT STRING(S).
NO TRANSLATED OUTPUT.

LAST CHARACTER INSPECTED IN TERM 1 WAS  N  IN MIDST OF FOLLOWING CONTEXT

THIS IS NOT A TEST SENTE

END OF RUN.
EXITM CALLED. GOODBYE.
R .583+2.766
```

Appendix 4.

Program Listing.

The program listing for the program which implements the canonic translation algorithm is contained in this appendix. The program may be divided into three parts: a preliminary phase which verifies the syntax of the canons typed in and assembles them into a SLIP list structure, the recursive scanning routine which forms the major part of the code, and a final routine which inspects and prints the results. Understanding the code requires a thorough comprehension of the SLIP system developed by Weizenbaum (7). The lack of elegance in the program is quite the fault of the author.

The following table identifying the major parts of the code may prove useful.

Label	Lines	Purpose of Code
NEWORD	57-74	Inputs a line from the typewriter, feeds characters one at a time to the canon-analyzing routine.
	107-285	Reads predicate names and makes various checks (left recursion, degree same as before, etc.) and assembles into list structure.
	383-395	Identifies next variable to be encountered should be marked as the one to use if the variable is needed in the later phase.
	396-434	Inputs variable and assembles into SLIP structure.
EVAL	444-472	Checks that all variables are defined.
PUTIN	505-574	Assembles list structure for input to scan program at zero level.

LUP00Q	592	Beginning of recursive routine. It is to this point that the program returns when "pushing".
	603-617	Makes an "object time" check for left recursion.
OUTCHK	621-671	Creates the 'needed' list of variables for which definitions must be found from other than the input string.
LUP008	677-697	Handles multiple results, each of which must be analyzed in turn.
	717-855	Compares input string with conclusion, "pushing" to find definition of variables if necessary.
PUSHIT	871-893	Saves state of program and returns to the beginning of the scan routine.
PRMCHK	897-977	Checks premises of a canon.
ASSMBL	981-1038	Assembles results of scan for next higher level before "popping".
POP	1042-1075	Uncovers the state of the program and goes to appropriate return routine.
POP1	1080-1164	Analyzes return resulting from "push" during scan of conclusion.
POP2	1165-1196	Analyzes return resulting from "push" during check of premises.
THKGOD	1200-1344	Outputs results of scan.
HERAUS	1348-1350	Exit.
	1355-1381	Obtains character from input data area, using pointer furnished by caller.

1385-1562 [illegible] answers using input routine at top.

PRTLST [illegible] subroutine to print out the content and structure of lists for debugging.

```
*********************************************************************************
    M5364         5163       THESIS           MAD FOR    M5364          5163       0
                R
                R
                R
                R CANONIC TRANSLATION PROGRAM.
                R THIS PROGRAM EMPLOYS A CANONIC SYSTEM AS A BASIS
                R FOR RECOGNIZING DEFINED SETS OF STRINGS. THE
                R FIRST PART OF THE PROGRAM VERIFIES THE SYNTAX OF
                R THE CANONIC SYSTEM WHICH IS INPUT AND
                R ASSEMBLES IT INTO A SLIP LIST STRUCTURE.
                R THE SECOND PART OF THE PROGRAM OPERATES RECURSIVELY
                R TO DETERMINE WHETHER A GIVEN STRING IS A MEMBER
                R OF A DEFINED SET OF STRINGS, OR WHETHER THE GIVEN
                R STRING IS ONE ELEMENT OF AN ORDERED N-TUPLE
                R WHICH IS A MEMBER OF A DEFINED SET.  IN THE LATTER
                R CASE, THE OTHER ELEMENTS OF THE N-TUPLE MAY BE
                R OUTPUT AS TRANSLATIONS.
                R
                R
                 N'S INTEGER
                 BOOLEAN EQUAL, EOLIND, STRTND, ERRS, NOTIN, IN, SOMERC, ALLRC
                1 ,LEMPTY, INP, DOLIND
                 DIMENSION BUFFER (14), INPUT (1000)
                R
                RINITIALIZATION OF SLIP SYSTEM.
                R
                 INITAS. (0)
    BEGINS       LIST. (SYSTEM)
                 LIST. (NAMES)
                 LINE = 0
                 DEFNUM = 0
                R
                RINITIALIZATION OF SYSTEM
                R
                 PRINT COMMENT $INPUT CANONS.$
                 PRINT COMMENT $ $
                 T'O SKIP
                R
                RINITIALIZATION PRIOR TO READING A CANON
                R
    LOOP1        IRALST. (VAR)
                 IRALST. ,(SVEVAR)
    SKIP         DEF = LIST. (9)
                 LIST. (VAR)
                 LIST. (SVEVAR)
                 NEED = LIST. (9)
                 DOLIND = 0B
                 COND = 1
                 EQUAL = 0B
                 EOLIND = 1B
    NEWPRM       PREM = LIST. (9)
                 TRMNUM = 0
    NEWTRM       TERM = LIST. (9)
                R
                RINPUT OF CANONS, CHARACTER BY CHARACTER, RIGHT
                RADJUSTED IN WORD
                R
    NEWORD       W'R .NOT. EOLIND, T'O GETW
    RDLINE       NUMB = RDFLXC. (BUFFER, 84)
```

```
           LINE = LINE + 1
           EOLIND = 0B
           POS = 6
           I = -1
           STRTND = 1B
GETW       W'R POS .GE. 6
                POS = 0
                I = I + 1
                WORT = BUFFER (I)
           E'L
           W'R NUMB .E. 0, T'O RDLINE
           NUMB = NUMB - 1
           W'R NUMB .E. 0, EOLIND = 1B
           POS = POS + 1
           WORD = WORT .RS. 30 .V. $     0$
           WORT = WORT .LS. 6
          R
          RCHECK TO SEE IF READING ANSWER TO QUESTIONS.
          R
           W'R COND .E. 8
                T'O RDANS
          R
          RCHECK FOR REMARK AND END CARDS
          R
           O'R STRTND
                STRTND = 0B
                W'R WORD .E. $     *$, T'O RDLINE
                W'R BUFFER (0) .A. 777777000000K .E. $END000$
                     W'R COND .E. 1, T'O EVAL
                     PRNTP. (ERR1)
                     V'S ERR1 = $LAST CANON IS INCOMPLETE$, 377777777777
          1K
                     T'O ERRIN
                E'L
           E'L
          R
          RCHECK TO SEE IF READING 'LITERAL' OF TERMINAL CHARACTERS
          R
           W'R COND .E. 6
                W'R WORD .E. BREAK
                     COND = 2
                O'E
                     NEWBOT. (WORD, TERM)
                     W'R EOLIND, NEWBOT. (606060606055K, TERM)
                E'L
                T'O NEWORD
           O'R WORD .E. $ $
                T'O NEWORD
          R
          RCHECK TO SEE IF READING PREDICATE
          R
           O'R COND .E. 7
                W'R WORD .NE. $     -$
                     LENGTH = LENGTH + 1
                     W'R LENGTH .E. 7
                          PRNTP. (ERR2)
                          V'S ERR2 = $TOO MANY CHARACTERS IN PREDICATE$
          1, 377777777777K
                          T'O ERRIN
                     E'L
          R
```

```
          RBUILD UP PREDICATE, CHARACTER BY CHARACTER
          R
                     NAME = NAME .LS. 6 .V. WORD .A. 000000000077K
                     T'O NEWORD
          R
          RSAVE PREDICATE NAME JUST READ IN
          R
                O'E
                     EQUIV = ITSVAL. (NAME, NAMES)
                     W'R EQUIV .E. 0
                          DEFNUM = DEFNUM + 1
                          EQUIV = DEFNUM .V. TRMNUM .LS. 18
                          NEWVAL. (NAME, EQUIV, NAMES)
                     E'L
                     CHKNUM = EQUIV .RS. 18
                     EQUIV = EQUIV .A. 777777K
          R
          RCHECK DEGREE OF PREDICATE
          R
                     W'R TRMNUM .NE. CHKNUM
                          P'T ERR15, NAME, LINE
                          V'S ERR15 = $H'DEGREE OF PREDICATE -',C6,
          1H'- IN LINE NUMBER',I3,H' NOT AS PREVIOUSLY DEFINED'*$
                     T'O ZAPALL
                     E'L
          R
          RIF CONCLUSION, MAKE VARIOUS CHECKS AND ADD CANON TO SYSTEM
          R
                     W'R EQUAL
          R
          RCHECK FOR LEFT RECURSION
          R
                          LRECUR = SEQRDR.(DEF)
LOOP3                     CHKPRM = SEQLR. (LRECUR, F)
                          W'R F .G. 0, T'O CHKILL
                          PRMPRM = TOP. (LSTNAM. (CHKPRM))
          1 .A. 777777K
                          W'R PRMPRM .NE. EQUIV, T'O LOOP3
                          CHECKC = SEQRDR. (PREM)
                          CHECKP = SEQRDR. (CHKPRM)
                          SOMERC = 0B
                          ALLRC = 1B
LOOP4                     TERMP = SEQLR. (CHECKP, F)
                          TERMC = SEQLR. (CHECKC, G)
                          W'R F .G. 0 .OR. G .G. 0
                               W'R SOMERC .AND. ALLRC
                                    PRNTP. (ERR11)
                                    V'S ERR11 = $COMPLETE LEFT RECURSION
          1$, 377777777777K
                                    T'O ERRIN
                               O'R SOMERC
                                    P'T ERR12, LINE
                                    V'S ERR12 = $H'WARNING- PARTIAL LEFT
          1 RECURSION IN LINE NUMBER',I3*$
                               E'L
                               T'O LOOP3
                          E'L
                          TERM1 = SEQRDR. (TERMP)
                          TERM2 = SEQRDR. (TERMC)
LOOP5                     PPPREM = SEQLR. (TERM1, F)
                          CONCLU = SEQLR. (TERM2, G)
```

```
                          W'R F .G. 0 .OR. G .G. 0, T'O LOOP4
                          W'R F .L. 0 .AND. G .L. 0
                               W'R PPPREM .E. CONCLU, T'O LOOP5
                          O'R F .E. 0 .AND. G .E. 0
                               W'R TOP. (LSTNAM. (CONCLU)) .E.
          1 CHKPRM .AND. PPPREM .E. CONCLU
                                    SOMERC = 1B
                                    T'O LOOP4
                               E'L
                          E'L
                          ALLRC = 0B
                          T'O LOOP4
          R
          RCHECK FOR ILLEGAL VARIABLE CONSTRUCTION, I.E.
          RA PREMISE TERM WITH MORE THAN A VARIABLE.
          R
CHKILL                    READP = SEQRDR. (DEF)
ADVANP                    PREMP = SEQLR. (READP, F)
                          W'R F .G. 0, T'O NOUSE
                          READT = SEQRDR. (PREMP)
ADVANT                    TERMP = SEQLR. (READT, F)
                          W'R F .G. 0, T'O ADVANP
                          NOTIN = 0B
                          IN = 0B
                          VARCNT = 0
                          READV = SEQRDR. (TERMP)
ADVANV                    VARIAB = SEQLR. (READV, F)
                          W'R F .G. 0
                               W'R VARCNT .G. 1
GOOF01                              PRNTP. (ERR16)
                                    V'S ERR16 = $VARIABLE NOT ISOLATED$
          1, 377777777777K
                                    T'O ERRIN
                               O'R VARCNT .L. 1
                                    PRNTP. (ERR22)
                                    V'S ERR22 = $TERM WITHOUT VARIABLE$
          1 , 377777777777K
                                    T'O ERRIN
                               E'L
                               T'O ADVANT
                          E'L
                          W'R F .L. 0, T'O GOOF01
                          VARIAB = TOP. (VARIAB)
                          VARCNT = VARCNT + 1
                          READC = SEQRDR. (SVEVAR)
LOOP6                     TERMV = SEQLR. (READC, F)
                          W'R F .G. 0
                               NOTIN = 1B
                               READL = SEQRDR. (NEED)
ADVANL                         TERML = SEQLR. (READL, F)
                               W'R F .G. 0
                                    NEWBOT. (VARIAB, NEED)
               P'T NOTTY, VARIAB
               V'S NOTTY = $H'NEED ',C6*$
                                    T'O ADVANV
                               O'R VARIAB .A. 7777777777K .E. TERML .A.
          1 7777777777K
                                    SUBST. (VARIAB .V. 77K10, SEQPTR.
          1 (READL))
                                    T'O ADVANV
                               E'L
```

```
                               T'O ADVANL
                          O'R TERMV .E. VARIAB
                               IN = 1B
                               T'O ADVANV
                          E'L
                          T'O LOOP6
          R
          RCHECK FOR UNUSED VARIABLE (ONE WHICH OCCURS ONLY ONCE
          RIN CANON)
          R
NOUSE                     READL = SEQRDR. (NEED)
ADVANN                    TERML = SEQLR. (READL, F)
                          W'R F .G. 0, T'O ADDCAN
                          W'R TERML .A. 77K10 .E. 77K10
                               TERML = TERML .A. 607777777777K
                               T'O ADVANN
                          E'L
                          P'T ERR17, TERML, LINE
                          V'S ERR17 = $H'WARNING- VARIABLE ',RC1,
          1H' IN LINE NUMBER',I3,H' UNUSED'*$
                          DELETE. (SEQPTR. (READL))
                          T'O ADVANN
ADDCAN                    MAKEDL. (PREM, DEF)
                          MAKEDL. (NEED, PREM)
                          EQU = ITSVAL. (EQUIV, SYSTEM)
                          W'R EQU .E. 0
                               EQU = LIST. (9)
                               NEWVAL. (EQUIV, EQU, SYSTEM)
                          E'L
                          NEWBOT. (DEF, EQU)
                          COND = 1
                          T'O LOOP1
          R
          RIF NOT CONCLUSION, SAVE PREMISE AND PREMISE PREDICATE
          R
                     O'E
                          NEWBOT. (PREM, DEF)
                          TEMP = LIST. (9)
                          MAKEDL. (TEMP, PREM)
                          NEWBOT. (EQUIV .V. TRMNUM .LS. 18, TEMP)
                          COND = 3
                          T'O NEWPRM
                     E'L
                E'L
          R
          RCHECK FOR BREAK BETWEEN TERMS
          R
           C'R WORD .E. $     .$
                W'R COND .NE. 2
                     PRNTP. (ERR3)
                     V'S ERR3 = $MISPLACED PERIOD$, 377777777777K
                     T'O ERRIN
                O'E
                     NEWBOT. (TERM, PREM)
                     TRMNUM = TRMNUM + 1
                     COND = 4
                     T'O NEWTRM
                E'L
          R
          RCHECK FOR BEGINNING OF NAME
```

```
R
 C'R WORD .E. $      -$
      W'R COND .NE. 2
           PRNTP. (ERR4)
           V'S ERR4 = $MISPLACED HYPHEN$, 377777777777K
           T'O ERRIN
      O'E
           COND = 7
           LENGTH = 0
           NAME = $ $
           NEWBOT. (TERM, PREM)
           TRMNUM = TRMNUM + 1
           T'O NEWORD
      E'L
R
RCHECK FOR BEGINNING OF TERMINAL CHARACTER 'LITERAL'
R
 C'R WORD .E. $      '$ .OR. WORD .E. $      *$ .OR. WORD .E.
1 $     /$
      W'R COND .E. 3
           P'T ERR10, WORD, LINE
           V'S ERR10 = $H.MISPLACED '.,RC1,H.' IN LINE NUMBER.
1,I3*$
           T'O ZAPALL
      O'E
           BREAK = WORD
           COND = 6
           T'O NEWORD
      E'L
R
RCHECK FOR COMMA AFTER PREDICATE
R
 C'R WORD .E. $      ,$
      W'R COND .NE. 3
           PRNTP. (ERR5)
           V'S ERR5 = $MISPLACED COMMA$, 377777777777K
           T'O ERRIN
      O'E
           COND = 4
           T'O NEWORD
      E'L
R
RCHECK FOR EQUALS, BEGINNING OF CONCLUSION
R
 C'R WORD .E. $      =$
      W'R COND .NE. 3 .AND. COND .NE. 1
           PRNTP. (ERR6)
           V'S ERR6 = $MISPLACED EQUALS SIGN$, 377777777777K
           T'O ERRIN
      O'E
           EQUAL = 18
           COND = 4
           T'O NEWORD
      E'L
R
RCHECK FOR TAB
R
 C'R WORD .E. $      1$
      W'R COND .E. 3
           PRNTP. (ERR13)
           V'S ERR13 = $MISPLACED TAB$, 377777777777K
```

```
           T'O ERRIN
      O'E
           NEWBOT. (606060606072K, TERM)
           COND = 2
           T'O NEWORD
      E'L
R
RCHECK FOR CARRIAGE RETURN
R
 C'R WORD .E. $     2$
      W'R COND .E. 3
           PRNTP. (ERR14)
           V'S ERR14 = $MISPLACED CARRIAGE RETURN$, 3777777777
177K
           T'O ERRIN
      O'E
           NEWBOT. (606060606055K, TERM)
           COND = 2
           T'O NEWORD
      E'L
R
RCHECK FOR $. INDICATES VARIABLE NEXT ENCOUNTERED
R SHOULD BE MARKED FOR NEED.
R
 C'R WORD .E. 606060606053K
      W'R COND .E. 3 .OR. EQUAL
           PRNTP. (ERR25)
           V'S ERR25 = $MISPLACED DOLLAR SIGN$, 377777777777K
           T'O ERRIN
      O'E
           DOLIND = 1B
           T'O NEWORD
      E'L
R
RASSUME CHARACTER IS VARIABLE
R
 O'E
      W'R COND .E. 3
           PRNTP. (ERR7)
           V'S ERR7 = $MISPLACED VARIABLE$, 377777777777K
           T'O ERRIN
      O'E
           COND = 2
           VARIAB = ITSVAL. (WORD, VAR)
           W'R EQUAL
                W'R VARIAB .E. 0
                     PRNTP. (ERR8)
                     V'S ERR8 = $UNDEFINED VARIABLE$, 37777777
17777K
                     T'O ERRIN
                E'L
                NEWBOT. (VARIAB, TERM)
                NEWBOT. (WORD, SVEVAR)
           O'E
                W'R VARIAB .E. 0
                     VARIAB = LIST. (9)
                     TEMP = LIST. (9)
                     MAKEDL. (TEMP, VARIAB)
                     NEWBOT. (WORD, VARIAB)
                     NEWVAL. (WORD, VARIAB, VAR)
```

```
                          O'R DOLIND
                               POPTOP. (LSTNAM. (VARIAB))
                          O'E
                               T'O ONLYON
                          E'L
                          NEWBOT. (PREM, LSTNAM. (VARIAB))
ONLYON                    NEWBOT. (VARIAB, TERM)
                          DOLIND = 0B
                     E'L
                     T'O NEWORD
                E'L
           E'L
          R
          RIN CASE OF ERROR, CANON IS ERASED AND MAY BE RECONSTRUCTED
          R
ERRIN      P'T ERR, LINE
           V'S ERR = $H' IN LINE NUMBER',I3*$
ZAPALL     IRALST. (TERM)
           IRALST. (PREM)
           IRALST. (DEF)
           T'O LOOP1
          R
          RVARIOUS ERROR CHECKS FOLLOW
          R
          RCHECK TO SEE IF ALL NAMES ARE DEFINED
          R
EVAL       ERRS = 0B
           DLIST = LSTNAM. (NAMES)
           SEQCHK = SEQRDR. (DLIST)
LOOP2      NAME = SEQLR. (SEQCHK, F)
           DEFNUM = SEQLR. (SEQCHK, TEMP) .A. 777777K
           W'R F .G. 0
                W'R ERRS
                     PRNTP. (COMM2)
                     V'S COMM2 = $PLEASE DEFINE ABOVE PREDICATES.$, 7777
          177777777K
                     T'O LOOP1
                E'L
                PRINT COMMENT $ $
                PRNTP. (COMM1)
                V'S COMM1 = $CONSISTENT SET OF CANONS.$, 777777777777K
                T'O TYPOUT
           E'L
           DEFCHK = ITSVAL. (DEFNUM, SYSTEM)
           W'R DEFCHK .E. 0
                ERRS = 1B
                P'T ERR9, NAME
                V'S ERR9 = $C6, H' UNDEFINED'*$
           E'L
           T'O LOOP2
          R
          RPRINT LIST OF PREDICATES.
          R
TYPOUT     PRINT COMMENT $ $
           PRINT COMMENT $LIST OF DEFINED PREDICATES AND DEGREES.$
           PRINT COMMENT $ $
           I = 0
FNDNMB     I = I + 1
           SEQCHK = SEQRDR. (LSTNAM. (NAMES))
SPCNMB     NAME = SEQLR. (SEQCHK, F)
```

```
           DEFNUM = SEQLR. (SEQCHK, G)
           PRMNUM = DEFNUM .RS. 18
           DEFNUM = DEFNUM .A. 777777K
           W'R F .G. 0, T'O PUTIN
           W'R DEFNUM .E. I
                P'T NOTE3, I, NAME, PRMNUM
                V'S NOTE3 = $I3,H'.  -',C6,H'-',I4*$
                T'O ENDNMB
           E'L
           T'O SPCNMB
          R
          RINPUT OF SOURCE STRINGS AND 'NEED' FLAGS.
          RA POINTER TO THE INPUT STRING IS USED IN
          RTHE LIST, RATHER THAN THE INPUT ITSELF.
          RTHE ADDRESS PORTION OF THE WORD CONTAINS THE
          RNUMBER OF THE LAST CHARACTER INPUTTED
          RAND THE DECREMENT CONTAINS THE NUMBER OF
          RTHE FIRST.  THOSE PARTS OF THE STRINGS
          RDERIVED FROM THE CANONIC DEFINTIONS
          RARE LEFT AS SINGLE CHARACTERS IN A SLIP
          RCELL.
          R
PUTIN      LIST. (MAXINP)
           IRALST. (NAMES)
           PRINT COMMENT $ $
           PRINT COMMENT $INPUT OF SOURCE STRINGS.$
RETRY      PRINT COMMENT $TYPE FINAL PREDICATE.$
           READIN. (NAME)
           EQUIV = ITSVAL. (NAME, NAMES)
           CHKNUM = EQUIV .RS. 18
           EQUIV = EQUIV .A. 777777K
           W'R EQUIV .E. 0
                P'T COMM4, NAME
                V'S COMM4 = $H'-',C6,H'- NOT FOUND'*$
                T'O RETRY
           E'L
           LIST. (SEARCH)
           INP = 0B
           LNECNT = 0
           PRINT COMMENT $ $
           PRINT COMMENT $TYPE -NONEED-, -NEED- OR -INPUT- FOR EACH TERM
          1.$
           THROUGH READY, FOR TRMNUM = 1, 1, TRMNUM .G. CHKNUM
                PRINT COMMENT $ $
                P'T COMM3, TRMNUM, NAME
                V'S COMM3 = $H'TERM NUMBER',I2,H' OF -',C6,H'-'*$
                READIN. (ANSWER)
                W'R ANSWER .E. $  NEED$
                     NEWBOT. ($NEED$, SEARCH)
                     T'O READY
                O'R ANSWER .E. $NONEED$
                     NEWBOT. ($PLEASE$, SEARCH)
                     T'O READY
                O'R ANSWER .E. $ INPUT$
                     INP = 1B
                     PRINT COMMENT $ $
                     PRINT COMMENT $INPUT STRING. EXTRA CARRIAGE RETURN
          1INDICATES END.$
                     PRINT COMMENT $ $
                     SAVI = (LNECNT * 6 + 1) .LS. 18
                     TEMP = LIST. (9)
```

```
                     NEWBOT. (TEMP, SEARCH)
                     NEWBOT. (LIST.(9), TEMP)
                     TEMP1 = LIST. (0)
                     NEWBOT. (TEMP1, TEMP)
RDINP                W'R LNECNT .E. 960
                          PRINT COMMENT $CANNOT ACCEPT MORE INPUT.$
                          T'O SVEINP
                     O'R LNECNT .G. 960
                          PRINT COMMENT $ONLY ONE MORE LINE OF INPUT.$
                     E'L
                     NUMB = RDFLXC. (INPUT (LNECNT), 120)
                     W'R NUMB .LE. 1
SVEINP                    SAV1 = SAV1 .V. (LNECNT * 6)
                          NEWBOT. (SAV1, TEMP1)
                          NEWBOT. (LNECNT * 6, MAXINP)
                          T'O READY
                     O'R NUMB .G. 80
                          NUMB = 80
                     E'L
                     REMNUM = NUMB - (NUMB/6)*6
                     LNECNT = LNECNT + (NUMB + 5)/6
                     HOLD = INPUT (LNECNT - 1)
                     W'R REMNUM .E. 0, T'O RDINP
                     THROUGH FILL, FOR N = 0, 1, N .GE. 6 - REMNUM
FILL                 HOLD = HOLD .V. 77K .LS. (N*6)
                     INPUT (LNECNT - 1) = HOLD
                     T'O RDINP
                 *
                     TRMNUM = TRMNUM - 1
                     T'O READY
                 *
READY            CONTINUE
                 PRINT COMMENT $ $
                 PRINT COMMENT $TYPE 0, 1 OR 2 FOR DEPTH OF COMMENTS.$
                 READIN. (ANSWER)
                 SWITCH = ANSWER .A. 7K
          W'R .NOT. INP .AND. SWITCH .G. 0
                 PRINT COMMENT $WARNING- NO INPUT.$
          E'L
          W'R SWITCH .G. 4, PRTLST. ($SYSTEM$, SYSTEM)
        *
        [illegible] OF INPUT STRING(S).
        *
SCAN      [illegible] (BLANK)
          [illegible] (BLNKS)
          PRINT COMMENT $$
          PRINT COMMENT $SCAN BEGINS.$
          PRINT COMMENT $$
EQP2      [illegible] NORM. (ITSVAL. (EQUIV, SYSTEM))
          W'R SWITCH .G. 2
                 PRTLST. ($SEARCH$, SEARCH)
          E'L
          [illegible]
          [illegible]
          [illegible] (BLANK2)
          [illegible] 2
EQP3      [illegible]
EQP3      [illegible] DEW. (DEF, I)
          W'R [illegible], T'O [illegible]
        *
        [illegible] RESUMING.
```

```
          R
           CHKR = SEQRDR. (STACKB)
RECURR     CHK1 = SEQLR. (CHKR, F)
           CHK2 = SEQLR. (CHKR, TEMP)
           W'R F .G. 0, T'O OUTCHK
           W'R CHK1 .NE. DEFINE, T'O RECURR
           W'R LSTEQL. (SEARCH, CHK2) .E. 0
                W'R SWITCH .G. 0
                PRINT COMMENT $LEFT RECURSION DETECTED.$
                E'L
                T'O LUP002
           E'L
           T'O RECURR
          R
          RDEVELOPE 'NEED' LIST.
          R
OUTCHK     FIND = LSSCPY. (SEARCH)
           TEMP1 = LIST. (9)
           MAKEDL. (TEMP1, FIND)
           NEWTOP. (SEQRDR. (FIND), TEMP1)
           TEMP = LSTNAM. (DEFINE)
           NEED = LSSCPY. (LSTNAM. (TEMP))
           PREM = SEQRDR. (TEMP)
           LOOK = SEQRDR. (FIND)
           LIST. (NONEED)
LUP003     PRMISE = SEQLR. (PREM, F)
           W'R F .G. 0, T'O PRTNED
           SEE = SEQLR. (LOOK, F)
           FNDTRM = SEQRDR. (PRMISE)
LUP005     VARIAB = SEQLR. (FNDTRM, G)
           W'R G .G. 0, T'O LUP003
           W'R G .E. 0
                VARIAB = TOP. (VARIAB)
                W'R SEE .E. $NEED$
                     NEWBOT. (VARIAB, NEED)
                O'R F .E. 0
                     NEWBOT. (VARIAB, NONEED)
                E'L
           E'L
           T'O LUP005
PRTNED     W'R LEMPTY. (NONEED), T'O LUP006
           TEMP1 = POPTOP. (NONEED)
           FNDTRM = SEQRDR. (NEED)
LUP004     VARIAB = SEQLR. (FNDTRM, F)
           W'R F .G. 0, T'O PRTNED
           W'R VARIAB .NE. TEMP1, T'O LUP004
           DELETE. (SEQPTR. (FNDTRM))
           T'O LUP004
LUP006     IRALST. (NONEED)
           W'R SWITCH .LE. 1, T'O STRTSC
           TEMP1 = SEQRDR. (NEED)
LUP010     TEMP2 = SEQLR. (TEMP1, F)
           W'R F .G. 0, T'O STRTSC
           P'T NOTEJ, TEMP2
           V'S NOTEJ = $H'NEED ',RC1,H'.'*$
           T'O LUP010
          R
          RGET CONCLUSION OF CANON.
          R
STRTSC     NEWBOT. (FIND, STACK2)
           W'R SWITCH .G. 4, PRTLST. ($NEED$, NEED)
```

```
           CONCL = SEQRDR. (TEMP)
LUP007     TERM = SEQLR. (CONCL, F)
           W'R F .G. 0, T'O PRMCHK
           EOLIND = 1B
           IN = 0B
           INP = 0B
          R
          RGET NEXT TERM OF CONCLUSION.
          R
           PIECE = SEQRDR. (TERM)
           T'O LUP011
LUP008     W'R IN, T'O GETINA
LUP009     IN = 0B
           W'R LEMPTY. (STACK1)
                W'R INP, T'O LUP007
                EOLIND = 0B
LUP011          CHAR = SEQLR. (PIECE, G)
                W'R G .G. 0
                     W'R EOLIND
                          INP = 1B
                     O'E
                          T'O LUP007
                     E'L
                E'L
          R
          RCHECK TO SEE IF SCAN HAS FAILED.
          R
                W'R LEMPTY. (STACK2), T'O LUP001
                TEMP = STACK1
                STACK1 = STACK2
                STACK2 = TEMP
           E'L
           FIND = POPTOP. (STACK1)
           SEE = LSTNAM. (FIND)
           READS = POPTOP. (SEE)
           W'R EOLIND
                SEQLR. (READS, F)
                W'R F .L. 0 .OR. INP
                     NEWTOP. (READS, SEE)
                     NEWBOT. (FIND, STACK2)
                     INP = 1B
                     T'O LUP009
                E'L
           E'L
           TEMP = CONT. (SEQPTR. (READS) + 1)
           HOLDP = TOP. (TEMP)
           HOLDT = BOT. (TEMP)
           W'R G .L. 0
          R
          RTERMINAL CHARACTER IN CONCLUSION. CHECK STRING.
          R
LUP015          W'R LEMPTY. (HOLDT), T'O NGOOD
                WORD = POPTOP. (HOLDT)
                STRTND = 0B
                W'R WORD .L. 0
                     W'R CHAR .E. WORD
                          NEWBOT. (WORD, HOLDP)
                          NEWBOT. (FIND, STACK2)
                          NEWTOP. (READS, SEE)
                          T'O LUP008
```

```
                     O'E
                          T'O NGOOD
                     E'L
                O'E
                     OBJECT = CHARAC. (WORD)
                     W'R OBJECT .E. CHAR
LUP019                    TEMP1 = WORD .A. 777777K6
                          W'R LEMPTY. (HOLDP), T'O TRAOVR
                          TEMP = POPBOT. (HOLDP)
                          W'R TEMP .G. 0 .AND. (TEMP .A. 777777K)
          1 .E. (WORD .RS. 18) - 1, T'O SKPOVR
                          NEWBOT. (TEMP, HOLDP)
TRAOVR                    TEMP = TEMP1 .V. (WORD .RS. 18) - 1
SKPOVR                    TEMP = TEMP + 1
                          NEWBOT. (TEMP, HOLDP)
                          W'R TEMP1 .GE. WORD .LS. 18, T'O JMPOVR
                          WORD = WORD + 1K6
                          NEWTOP. (WORD, HOLDT)
JMPOVR                    W'R STRTND, T'O LUP015
                          NEWBOT. (FIND, STACK2)
                          NEWTOP. (READS, SEE)
                          T'O LUP008
                     O'R OBJECT .E. $00NULL$
                          STRTND = 1B
                          T'O LUP019
                     O'R OBJECT .E. $000END$
                          T'O LUP015
                     O'E
NGOOD                     IRALST. (FIND)
                          W'R IN
                               CHAR = SAVECH
                               G = SAVEG
                          E'L
                          T'O LUP009
                     E'L
                E'L
          R
          RVARIABLE IN CONCLUSION.
          RCHECK TO SEE IF VARIABLE PREVIOUSLY DEFINED.
          R
           C'E
                VARIAB = TOP. (CHAR)
                DLIST = ITSVAL. (VARIAB, FIND)
                W'R DLIST .NE. 0
                     SAVECH = CHAR
                     SAVEG = G
                     IN = 1B
                     G = -1
                     DLIST = SEQRDR. (DLIST)
LUP021               CHAR = SEQLR. (DLIST, F)
                     W'R F .G. 0, T'O GETOUT
                     W'R CHAR .L. 0, T'O LUP027
                     I = CHAR
LUP023               CHAR = CHARAC. (I)
                     I = I + 1K6
                     W'R CHAR .E. $000END$, T'O LUP021
                     W'R CHAR .E. $00NULL$, T'O LUP023
                     ALLRC = 1B
                     T'O LUP015
LUP027               ALLRC = 0B
```

```
                     T'O LUP015
GETINA               FIND = POPBOT. (STACK2)
                     READS = POPTOP. (SEE)
                     W'R ALLRC, T'O LUP023
                     T'O LUP021
GETOUT               CHAR = SAVECH
                     G = SAVEG
                     NEWBOT. (FIND, STACK2)
                     NEWTOP. (READS, SEE)
                     T'O LUP009
                E'L
          R
          RVARIABLE IS NOT YET DEFINED, SO PROGRAM
          R MUST SEARCH RECURSIVELY. SELECT PREMISE WITH WHICH
          R TO SEARCH FOR VARIABLE.
          R
                PRPNTR = TOP. (LSTNAM. (CHAR))
                PRMNUM = TOP. (LSTNAM. (PRPNTR)) .A. 777777K
          R
          RCHECK OTHER TERMS (AND VARIABLES) IN CHOSEN PREMISE.
          R
                LIST. (PUSHES)
                REMPTR = SEQRDR. (PRPNTR)
LUP031          TERM = SEQLR. (REMPTR, F)
                W'R F .G. 0, T'O PUSH1
                TEMP = TOP. (TERM)
                ZIEL = TOP. (TEMP)
          R
          RINSERT STRING FOR VARIABLE PRESENTLY SOUGHT.
          R
                W'R ZIEL .E. VARIAB
                     TEMP = LIST. (9)
                     NEWBOT. (TEMP, PUSHES)
                     TEMP2 = LIST. (9)
                     NEWBOT. (TEMP2, TEMP)
                     TEMP1 = LSSCPY. (HOLDT)
                     NEWBOT. (TEMP1, TEMP)
                     ABANDN. (TEMP1)
                O'E
          R
          RSEE IF OTHER VARIABLES PREVIOUSLY DEFINED.
          R
                     ISITDF = ITSVAL. (ZIEL, SEE)
                     W'R ISITDF .NE. 0
                          TEMP1 = LSSCPY. (ISITDF)
                          TEMP2 = LIST. (9)
                          TEMP = LIST. (9)
                          NEWBOT. (TEMP2, TEMP)
                          NEWBOT. (TEMP1, TEMP)
                          ABANDN. (TEMP1)
                          NEWBOT. (TEMP, PUSHES)
                     O'E
          R
          RDECIDE WHETHER TO FLAG AS 'NEED' OR 'PLEASE'.
          R
                          W'R PRPNTR .NE. TOP. (LSTNAM. (TEMP)),
          1 T'O LUP037
                          NDPTR = SEQRDR. (NEED)
LUP035                    CKNEED = SEQLR. (NDPTR, F)
                          W'R F .G. 0
```

```
LUP037                             NEWBOT. ($PLEASE$, PUSHES)
                              O'R ZIEL .E. CKNEED
                                   NEWBOT. ($NEED$, PUSHES)
                              O'E
                                   T'O LUP035
                              E'L
                         E'L
                    E'L
                    T'O LUP031
               E'L
          R
          RINFORMATION FOR RECURSION ASSEMBLED, SO SAVE STUFF
          RFOR THE PUSH.
          R
PUSH1      SOMERC = 0B
           W'R SWITCH .G. 1
                P'T NOTE1, PRMNUM
                V'S NOTE1 = $H'SCAN PUSH FOR',I3*$
           E'L
           T'O PUSHIT
PUSH2      SOMERC = 1B
           W'R SWITCH .G. 1
                P'T NOTEA, PRMNUM
                V'S NOTEA = $H'PREMISE PUSH FOR',I3*$
           E'L
PUSHIT     NEWTOP. (DEF, STACKA)
           NEWTOP. (EQUIV, STACKA)
           NEWTOP. (NEED, STACKA)
           NEWTOP. (SEARCH, STACKB)
           NEWTOP. (DEFINE, STACKB)
           NEWTOP. (STACK1, STACKA)
           NEWTOP. (STACK2, STACKA)
           NEWTOP. (CONCL, STACKA)
           NEWTOP. (PIECE, STACKA)
           NEWTOP. (FIND, STACKA)
           NEWTOP. (SEE, STACKA)
           NEWTOP. (READS, STACKA)
           NEWTOP. (HOLDP, STACKA)
           NEWTOP. (HOLDT, STACKA)
           NEWTOP. (CHAR, STACKA)
           NEWTOP. (VARIAB, STACKA)
           NEWTOP. (PRPNTR, STACKA)
           NEWTOP. (ANSWER, STACKA)
           NEWTOP. (SOMERC, STACKA)
           NEWTOP. (EOLIND, STACKA)
           EQUIV = PRMNUM
           SEARCH = PUSHES
           T'O LUP000
          R
          RCHECK WHETHER PREMISE CONDITIONS ARE SATISFIED.
          R
PRMCHK     PIECE = SEQRDR. (DEFINE)
LUP051     W'R LEMPTY. (STACK1)
                W'R LEMPTY. (STACK2), T'O LUP001
                PRPNTR = SEQLR. (PIECE, F)
                W'R F .G. 0, T'O ASSMBL
                TEMP = STACK1
                STACK1 = STACK2
                STACK2 = TEMP
           E'L
```

```
           FIND = POPTOP. (STACK1)
           SEE = LSTNAM. (FIND)
           READS = POPTOP. (SEE)
           PRMNUM = TOP. (LSTNAM. (PRPNTR)) .A. 777777K
           DLIST = ITSVAL. (PRPNTR .A. 77777K, FIND)
          R
          RPREMISE HAS NOT BEEN PREVIOUSLY VERIFIED WHILE
          RSEARCHING CONCLUSION.
          R
           W'R DLIST .E. 0
                LIST. (PUSHES)
                TERM = SEQRDR. (PRPNTR)
LUP053          TOPS = SEQLR. (TERM, F)
                W'R F .G. 0, T'O PUSH2
                TOPS = TOP. (TOP. (TOPS))
                DLIST = ITSVAL. (TOPS, FIND)
                W'R DLIST .E. 0
          R
          RVARIABLE NOT YET DEFINED. INSERT 'NEED'
          R
                     NEWBOT. ($NEED$, PUSHES)
                     W'R SWITCH .G. 0
                          P'T NOTED
                          V'S NOTED = $H-'NEED' REQUEST IN PREMISE CHEC
          1K.-*$
                     E'L
                     T'O LUP053
                E'L
                TEMP1 = LIST. (9)
                NEWBOT. (TEMP1, PUSHES)
                TEMP2 = LIST. (9)
                NEWBOT. (TEMP2, TEMP1)
                TEMP2 = LSSCPY. (DLIST)
                NEWBOT. (TEMP2, TEMP1)
                ABANDN. (TEMP2)
                T'O LUP053
           C'E
          R
          RPREMISE HAS BEEN PREVIOUSLY GENERATED IN SCAN
          RCF CONCLUSION.
          R
                SOMERC = 0B
                PUSHES = LSSCPY. (DLIST)
                TERM = SEQRDR. (PUSHES)
                TEMP = SEQRDR. (PRPNTR)
LUP057          TOPS = SEQLR. (TERM, F)
                TEMP3 = SEQLR. (TEMP, H)
                W'R F .G. 0
                     W'R SOMERC, T'O PUSH2
                     IRALST. (PUSHES)
                     NEWTOP. (READS, SEE)
                     NEWBOT. (FIND, STACK2)
                     T'O LUP051
                O'R F .L. 0
                     SOMERC = 1B
                     TEMP3 = TOP. (TOP. (TEMP3))
                     DLIST = ITSVAL. (TEMP3, FIND)
                     W'R DLIST .E. 0
                          P'T NOTED
                          SUBST. ($NEED$, SEQPTR. (TERM))
```

```
                          T'O LUP057
                     E'L
                     TEMP1 = LIST. (9)
                     SUBST. (TEMP1, SEQPTR. (TERM))
                     TEMP2 = LIST. (9)
                     NEWBOT. (TEMP2, TEMP1)
                     TEMP2 = LSSCPY. (DLIST)
                     NEWBOT. (TEMP2, TEMP1)
                     ABANDN. (TEMP2)
                E'L
                T'O LUP057
           E'L
          R
          RASSEMBLE CONCLUSION TO BE TRANSMITTED UPSTAIRS
          R
ASSMBL     W'R LEMPTY. (STACK2), T'O LUP001
           FIND = POPTOP. (STACK2)
           READS = POPTOP. (LSTNAM. (FIND))
           TERM = SEQRDR. (FIND)
           PREM = SEQRDR. (LSTNAM. (DEFINE))
LUP067     TOPS = SEQLR. (TERM, G)
           CONCL = SEQLR. (PREM, F)
          R
          RASSEMBLED, ADD TO ANSWER AND RETURN.
          R
           W'R F .G. 0
                NODLST. (FIND)
                NEWBOT. (FIND, ANSWER)
                T'O ASSMBL
          R
          RIF TERM SCANNED, SKIP IT.
          R
           O'R G .E. 0
                T'O LUP067
          R
          RIF NEED OR PLEASE, ASSEMBLE.
          R
           O'R G .L. 0
                SOMERC = 0B
                TEMP = LIST. (9)
                TEMP1 = SEQRDR. (CONCL)
LUP071          TEMP2 = SEQLR. (TEMP1, F)
                W'R F .G. 0
                     W'R SOMERC .OR. TOPS .E. $NEED$
                          TEMP1 = LIST. (9)
                          SUBST. (TEMP1, SEQPTR. (TERM))
                          NEWBOT. (TEMP, TEMP1)
                          TEMP2 = LIST. (9)
                          NEWBOT. (TEMP2, TEMP1)
                     O'E
                          IRALST. (TEMP)
                     E'L
                     T'O LUP067
                O'R F .E. 0
                     SOMERC = 1B
                     DLIST = TOP. (TEMP2)
                     ANTWRT = ITSVAL. (DLIST, FIND)
                     W'R ANTWRT .E. 0
                          W'R TOPS .E. $NEED$
                               PRINT COMMENT $'NEED' ERROR.$
```

```
                          E'L
                          IRALST. (FIND)
                          T'O ASSMBL
                     E'L
                     TEMP3 = LSSCPY. (ANTWRT)
                     INLSTR. (TEMP3, (CONT. (TEMP .A.
          1 77777K)) .RS. 18)
                     IRALST. (TEMP3)
                O'R F .L. 0
                     NEWBOT. (TEMP2, TEMP)
                E'L
                T'O LUP071
           E'L
          R
          RPCP-UP ROUTINE
          R
POP        W'R SWITCH .G. 2
                PRTLST. ($ANSWER$, ANSWER)
           E'L
           IRALST. (STACK1)
           IRALST. (STACK2)
           W'R LEMPTY. (STACKA), T'O THKGOD
           IRALST. (SEARCH)
           RTRN1 = ANSWER
           ECLIND = POPTOP. (STACKA) .E. 1
           SCMERC = POPTOP. (STACKA) .E. 1
           ANSWER = POPTOP. (STACKA)
           PRPNTR = POPTOP. (STACKA)
           VARIAB = POPTOP. (STACKA)
           CHAR = POPTOP. (STACKA)
           HCLDT = POPTOP. (STACKA)
           HCLDP = POPTOP. (STACKA)
           READS = POPTOP. (STACKA)
           SEE = POPTOP. (STACKA)
           FIND = POPTCP. (STACKA)
           PIECE = POPTOP. (STACKA)
           CCNCL = POPTOP. (STACKA)
           STACK2 = POPTOP. (STACKA)
           STACK1 = POPTOP. (STACKA)
           CEFINE = POPTOP. (STACKB)
           SEARCH = POPTOP. (STACK8)
           NEED = POPTCP. (STACKA)
           ECUIV = POPTOP. (STACKA)
           DEF = POPTOP. (STACKA)
           G = 0
           W'R SWITCH .G. 1
                P'T NOTE2, EQUIV
                V'S NOTE2 = $H'POP BACK TO',I3,H'.'*$
           E'L
           W'R SOMERC, T'O POP2
          R
          RRETURN TO SCAN OF CONCLUSION AFTER PUSHING
          RFCR DEFINITION OF A VARIABLE.
          R
POP1       W'R LEMPTY. (RTRN1)
                IRALST. (RTRN1)
                IRALST. (FIND)
                INP = 0B
           W'R SWITCH .L. 4, T'O LUP009
           PRTLST. ($STACK1$, STACK1)
```

```
           PRTLST. ($STACK2$, STACK2)
                T'O LUP009
           E'L
           FNDCPY = LSSCPY. (FIND)
           TEMP = SEQRDR. (FIND)
           TEMP3 = SEQRDR. (FNDCPY)
LUP079     TEMP1 = SEQLR. (TEMP, F)
           TEMP2 = SEQLR. (TEMP3, H)
           W'R F .G. 0
                T'O LUP080
           O'E
                W'R TEMP .E. READS
                     CPYHDP = TOP. (TEMP2)
                     CPYRDS = TEMP3
                     LINKS = CONT. (TEMP2 .A. 77777K) .RS. 18
                E'L
                T'O LUP079
           E'L
          R
          RSAVE THE RETURN ANSWER, AND DEFINE VARIABLES
          RAND PREDICATES AS GIVEN FROM PUSH.
          R
LUP080     TEMP1 = POPTOP. (RTRN1)
           TEMP3 = SEQRDR. (TEMP1)
           TEMP2 = SEQRDR. (PRPNTR)
LUP081     TEMP4 = SEQLR. (TEMP3, H)
           TEMP5 = SEQLR. (TEMP2, F)
           W'R F .G. 0
                NEWVAL. (PRPNTR .A. 77777K, TEMP1, FNDCPY)
                ABANDN. (TEMP1)
                NEWTOP. (CPYRDS, LSTNAM. (FNDCPY))
                NEWBOT. (FNDCPY, STACK2)
                T'O POP1
           O'R H .L. 0
                W'R TEMP4 .E. $NEED$
                     PRINT COMMENT $'NEED' ERROR.$
                E'L
                T'O LUP081
           O'E
                TMPVAR = TOP. (TOP. (TEMP5))
                PRVDEF = ITSVAL. (TMPVAR, FNDCPY)
          R
          RVARIABLE PREVIOUSLY DEFINED. COMPARE DEFINITIONS.
          R
                W'R PRVDEF .NE. 0
                     W'R LSTEQL. (PRVDEF, TOP. (TEMP4)) .NE. 0
                          IRALST. (FNDCPY)
                          IRALST. (TEMP1)
                          T'O POP1
                     E'L
                O'E
          R
          RADD DEFINITION.
          R
                     NEWVAL. (TMPVAR, TOP. (TEMP4), FNDCPY)
                     W'R VARIAB .E. TMPVAR
                          SUBST. (POPBOT. (TEMP4), LINKS)
                          CHKO = LSSCPY. (TOP. (TEMP4))
                          NEWTOP. (LIST. (9), TEMP4)
                          W'R LEMPTY. (CHKO), T'O LUP081
```

[redacted]

```
THKGD2     W'R LEMPTY. (ANSWER)
              W'R ALLRC, T'O HERAUS
              W'R SOMERC
                  PRINT COMMENT $SCAN COMPLETED.  SYNTAX ERROR IN INP
          1UT STRING. $
           PRINT COMMENT $PART(S) OF INPUT OR NEED STRING(S) NOT SCANNED
          1.$
                  T'O LUP150
              O'E
                  PRINT COMMENT $SCAN FAILED.  SYNTAX ERROR IN INPUT
          1STRING(S).$
              E'L
LUP150        PRINT COMMENT $NO TRANSLATED OUTPUT.$
              CHKNUM = 0
              MAX1 = 0
              CONCHK = SEQRDR. (SEARCH)
LUPERR        SEECHK = SEQLR. (CONCHK, F)
              W'R F .G. 0, T'O HERAUS
              CHKNUM = CHKNUM + 1
              W'R F .L. 0, T'O LUPERR
              I = SEQLR. (MAXCHK, F)
              OLDMAX = MAX1
              MAX1 = ¬I .A. 777777K
              MAX2 = I .RS. 18
              PRINT COMMENT $ $
              W'R CHARAC. (I + 1K6) .E. $00NUL($ .AND. MAX1 - MAX2
          1 .L. 6
                  P'T NOTE4, CHKNUM
                  V'S NOTE4 = $H'INPUT TERM',I2,H' COMPLETELY SCANNED
          1.'*$
              O'E
                  MAX3 = CHARAC. (I)
                  P'T NOTE5, CHKNUM, MAX3
                  V'S NOTE5 = $H'LAST CHARACTER INSPECTED IN TERM'
          1,I2,H' WAS  ',RC1,H'  IN MIDST OF FOLLOWING CONTEXT.'*$
                  PRINT COMMENT $ $
                  LINE1 = (MAX2 - 1)/6 - 2
                  LINE2 = (OLDMAX + 5)/6 - 1
                  LINE3 = (MAX1 - 1)/6
                  THROUGH ERRLUP, FOR I = 0, 1, I .E. 5
                      W'R LINE1 + I .LE. LINE2 .OR. LINE1 + I .G.
          1 LINE3
                          BUFFER(I) = 575757575757K
                      O'E
                          BUFFER(I) = INPUT(LINE1 + I)
                      E'L
ERRLUP                CONTINUE
                  P'T NOTE6, BUFFER(0),...,BUFFER(4)
                  V'S NOTE6 = $5C6*$
              E'L
              T'O LUPERR
           C'E
              SOMERC = 1B
              HOLD = POPTOP. (ANSWER)
              ENDCHK = SEQRDR. (HOLD)
              TEMP4 = SEQRDR. (SEARCH)
LUPSEE        SEECHK = SEQLR. (ENDCHK, F)
              TEMP5 = SEQLR. (TEMP4, H)
              W'R F .G. 0, T'O ALLOVR
              W'R TEMP5 .E. $PLEASE$, T'O LUPSEE
```

```
                TEMP = BOT. (SEECHK)
                W'R .NOT. LEMPTY. (TEMP)
                     TEMP1 = POPTOP. (TEMP)
                     W'R .NOT. LEMPTY. (TEMP), T'O THKGD1
                     TEMP2 = TEMP1 .RS. 18
                     TEMP3 = TEMP1 .A. 777777K
                     TEMP1 = CHARAC. (TEMP1)
                     W'R TEMP1 .NE. $000END$ .AND. TEMP1 .NE.
          1 $00NULL$ .OR. TEMP3 - TEMP2 .G. 5, T'O THKGD1
                E'L
                T'O LUPSEE
           E'L
          R
          RSCAN WAS SUCCESFUL.  PRINT OUT 'NEEDED' TERMS.
          R
ALLOVR     W'R ALLRC
                PRINT COMMENT $ $
                PRINT COMMENT $ADDITIONAL SUCCESSFUL SCAN.$
                T'O ALLGNE
           O'E
                PRINT COMMENT $SCAN SUCCESSFUL.$
                ALLRC = 1B
ALLGNE          PRINT COMMENT $TRANSLATED OUTPUT (IF ANY) FOLLOWS.$
                CONCHK = SEQRDR. (SEARCH)
                TRMNUM = 0
LUPOUT          CONCL = SEQLR. (CONCHK, F)
                SEECHK = SEQLR. (ENDCHK, G)
                TRMNUM = TRMNUM + 1
                W'R F .G. 0, T'O THKGD1
                W'R CONCL .NE. $NEED$, T'O LUPOUT
                PRINT COMMENT $ $
                W'R SEECHK .E. $NEED$, PRINT COMMENT $'NEED' ERR$
                P'T NOTE7, TRMNUM
                V'S NOTE7 = $H'TERM NUMBER',I2,H'.'*$
                PRINT COMMENT $ $
                SEECHK = TOP. (SEECHK)
                INP = 0B
LUP100          THROUGH LUP101, FOR I = 0, 1, I .E. 14
LUP101          BUFFER(I) = 575757575757K
                BUFFER(14) = 777777777777K
                I = 0
                G = 30
                WRDCNT = 0
CHKEMP          W'R G .LE. -6
                     G = 30
                     WRDCNT = WRDCNT + 1
                E'L
                W'R I .E. 80
CROUT                PRNTP. (BUFFER (0))
                     T'O LUP100
                E'L
                W'R INP
LUP105               INP = 1B
LUP107               TEMP1 = CHARAC. (TEMP)
                     TEMP = TEMP + 1K6
                     W'R TEMP1 .E. $00NULL$, T'O LUP107
                     W'R TEMP1 .E. $000END$, T'O LUP109
                     T'O LUP113
                O'E
LUP109               W'R LEMPTY. (SEECHK)
```

[redacted]

```
           E'O READIN.
           CEND = 8
           DUMMY = $ $
           T'O RDLINE
RDANS      W'R WORD .E. $      -$ .OR. WORD .E. $        $, T'O SKIPIT
           DUMMY = DUMMY .LS. 6 .V. WORD .A. 77K
SKIPIT     W'R EOLINO, FUNCTION RETURN
           T'O NEWORD
           E'N
           E'N
```

```
*******************************************************************************
   M5364         5163       PRTLST          MAD FOR   M5364          5163       05,
             EXTERNAL FUNCTION (NAME, LSTOUT)
             N'S INTEGER
             BOOLEAN LEMPTY
             E'O PRTLST.
             PRINT COMMENT $ $
             P'T NOTEBB, NAME, GETMEM. (0)
             V'S NOTEBB = $C6,H' MEM=',I6*$
             I = 0
             LIST. (STACK)
             LISSNM = LSTOUT
  START      NUMBER = ITSVAL. (LISSNM, STACK)
             W'R NUMBER .NE. 0
                  P'T NOTE2, NUMBER
                  V'S NOTE2 = $H'LIST',I3*$
  AROUND          W'R LEMPTY. (STACK)
                       PRINT COMMENT $ $
                       IRALST. (STACK)
                       FUNCTION RETURN
                  O'E
                       S = POPTOP. (STACK)
                       POINT = POPTOP. (STACK)
                       NUMB = POINT .A. 777777K
                       POINT = POINT .RS. 18
                       W'R POINT .E. 1, T'O RETURN
                       T'O GOBACK
                  E'L
             E'L
             I = I + 1
             NUMB = I
             NEWVAL. (LISSNM, NUMB, STACK)
             P'T NOTE3, NUMB
             V'S NOTE3 = $H'BEGIN',I3,H'.'*$
             S = SEQRDR. (LISSNM)
             L = LSTNAM. (LISSNM)
             W'R L .NE. 0
                  PRINT COMMENT $DLIST.$
                  NEWTOP. (NUMB .V. 1K6, STACK)
                  NEWTOP. (S, STACK)
                  LISSNM = L
                  T'O START
  RETURN          PRINT COMMENT $END DLIST.$
             O'E
                  PRINT COMMENT $NO DLIST.$
             E'L
  GOBACK     W = SEQLR. (S, F)
             W'R F .G. 0
                  P'T NOTE6, NUMB
                  V'S NOTE6 = $H'END',I3,H'.'*$
                  T'O AROUND
             O'R F .E. 0
                  W'R W .A. 7000007K5 .NE. 0, T'O READER
                  PRINT COMMENT $LIST NAME.$
                  NEWTOP. (NUMB, STACK)
                  NEWTOP. (S, STACK)
                  LISSNM = W
                  T'O START
             O'E
  READER          P'T NOTE5, W, W
                  V'S NOTE5 = $H. '.,C6,H.'    '.,K12,H.'.*$
```

```
      T'O GOBACK
E'L
E'N
```

Bibliography

1. Cheatham, T. E. and Kirk Sattley, Syntax-Directed Compiling, Proceedings 1964 Spring Joint Computer Conference, pp. 31-57, American Federation of Information Processing Societies (1964).

2. Donovan, John J., Investigations in Simulation and Simulation Languages, Ph.D. Thesis, Yale Univesity, New Haven, Connecticut; Fall, 1966.

3. Donovan, John J. and Henry Ledgard, A Formal System for the Specification of the Syntax and Translation of Computer Languages, M.I.T., 1967.

4. Post, E. L.,"Formal Reductions of the General Combinatorial Decision Problem", American Journal of Mathematics, Vol. 65, pp. 197-215; 1943.

5. Shea, Dorothy, CTSS SNOBOL User's Manual, Project MAC Memo MAC-M-307-1, Project MAC, M.I.T., Cambridge, Mass.; October, 1966.

6. Smullyan, R. M., Theory of Formal Systems, Princeton University Press, Princeton, New Jersey; 1961.

7. Weizenbaum, J.,"Symmetric List Processor", Communications of the ACM, Vol. 6, No. 9; September, 1963.

8. Weizenbaum, J., The Symbolic SLIP-Mad System, Project MAC, M.I.T., Cambridge, Mass; September, 1965.

CS-TR Scanning Project
Document Control Form

Date : 12/ 11 / 95

Report # LCS-TR-46

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: 84 (91-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:

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:
IMAGE MAP: (1- 84) UN#'ED TITLE & ACKN, PAGES 3-78, UN#BLK 79-84
(85-91) SCANCONTROL, COVER, FUNDING AGENT, DOD, TRGT'S (3)

Scanning Agent Signoff:
Date Received: 12/ 11 / 95 Date Scanned: 12/ 22/ 95 Date Returned: 12/ 28/ 95

Scanning Agent Signature: Michael W. Cook

Rev 9/94 DS/LCS Document Control Form cstrform.vsd

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Massachusetts Institute of Technology
Project MAC

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
None

3. REPORT TITLE
A Canonic Translator

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Bachelor's Thesis, Electrical Engineering, June 1967

5. AUTHOR(S) *(Last name, first name, initial)*
Alsop, Joseph W.

6. REPORT DATE
November 1967

7a. TOTAL NO. OF PAGES
84

7b. NO. OF REFS
8

8a. CONTRACT OR GRANT NO.
Office of Naval Research, Nonr-4102(01)
b. PROJECT NO.
NR 048-189
c.
RR 003-09-01
d.

9a. ORIGINATOR'S REPORT NUMBER(S)
MAC-TR-46 (THESIS)

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES
Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES
None

12. SPONSORING MILITARY ACTIVITY
Advanced Research Projects Agency
3D-200 Pentagon
Washington, D. C. 20301

13. ABSTRACT

This thesis presents an algorithm to recognize and translate sets of character strings specified by canonic systems. The ability of canonic systems to define the context sensitive features of strings and to specify their translation allows the algorithm to recognize and translate real computer languages. It is also applicable in other language systems.

Canonic systems are discussed, and several examples of their use are given. The algorithm is described, and examples of canonic translation are presented using a program implementation.

14. KEY WORDS

Canonic systems
Canonic translators
Computers
Machine-aided cognition
Multiple-access computers
On-line computers
Time-sharing
Time-shared computers
Translators

DD FORM 1 NOV 65 1473 (M.I.T.)

UNCLASSIFIED
Security Classification