AFCRL-62-549
AUGUST 1962

63-2-2

Research Report

A Phrase-Structure Language Translator

A. L. BASTIAN, JR.

COMPUTER AND MATHEMATICAL SCIENCES LABORATORY PROJECT 4641

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES, OFFICE OF AEROSPACE RESEARCH, UNITED STATES AIR FORCE, L.G. HANSCOM FIELD, MASS.

Requests for additional copies by Agencies of the Department of Defense, their contractors, and other government agencies should be directed to the:

Armed Services Technical Information Agency
Arlington Hall Station
Arlington 12, Virginia

Department of Defense contractors must be established for ASTIA services, or have their 'need-to-know' certified by the cognizant military agency of their project or contract.

All other persons and organizations should apply to the:

U. S. DEPARTMENT OF COMMERCE
OFFICE OF TECHNICAL SERVICES,
WASHINGTON 25, D. C.

ERRATA

The following corrections apply to AFCRL Research Report 62-549, entitled "A Phrase-Structure Language Translator," and dated August 1962.

Page 3

Line 18, change 'tract' to 'track'

Line 27, change to read 'character $\mathcal{P}$ (sometimes written as $\mathbb{P}$ herein) is used for this purpose.'

Line 28, change 'Pn]]' to '$\mathcal{P}$n]]'

Line 31, change 'P_n' to '$\mathcal{P}_n$'

Page 4

Lines 1 through 4, change 'P' to '$\mathcal{P}$'

Line 15, change 'P' to '$\mathcal{P}$'

Page 7

Line 4, change 'P, F' to '$\mathcal{P}$, $\mathcal{F}$'

Lines 21, 22, 28, 29, and 30, change 'P' to '$\mathcal{P}$'

Page 8

Lines 1, 2, 4, 5, 8, and 14, change 'P1]]', 'P2]]', and 'Pn' to '$\mathcal{P}$1]]', '$\mathcal{P}$2]], and '$\mathcal{P}$n'

Page 11

Lines 6, 7, and 8, delete all between 'and' of Line 6 and 'go' of Line 8 (sentence will then read 'If STABS(i)$\neq$ GOAL we set SF to F and go to NEWS4.')

Page 12

Line 15, change 'F_n' to '$\mathcal{F}_n$'

Line 19, change 'F_1 and F_2' to '$\mathcal{F}_1$ and $\mathcal{F}_2$'

Line 26, change 'a P or F' to 'a $\mathcal{P}$ or $\mathcal{F}$'

Line 33, change to 'if a $\mathcal{P}$ we go to WASSP'

Line 34, change to 'if an $\mathcal{F}$ we go to WASSF'

Page 13

Lines 31, 34, and 35, change 'P' and 'F' to '$\mathcal{P}$' and '$\mathcal{F}$'

Page 22

Line 12, change 'EROES' to 'ERORS'

Page 34

Line 23, change to read 'where n is either 1, 2, or 3'

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
OFFICE OF AEROSPACE RESEARCH
UNITED STATES AIR FORCE
L.G. HANSCOM FIELD, MASS.

AFCRL-62-549
AUGUST 1962

Research Report

A Phrase-Structure Language Translator

A. L. BASTIAN, JR.

COMPUTER AND MATHEMATICAL SCIENCES LABORATORY PROJECT 4641

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES, OFFICE OF AEROSPACE RESEARCH, UNITED STATES AIR FORCE, L.G. HANSCOM FIELD, MASS.

Abstract

The author discusses a syntax-directed translator with several new features which result in much faster running time and the ability to handle longer input strings. The computer program for the translator is described in detail, and an example of its operation is provided.

Contents

		Page
	Abstract	iii
1.	Introduction	1
2.	General Statement of Purpose	2
3.	Syntactical Description of the Language	3
4.	Tabling the Syntactical Description	4
5.	OTP	7
6.	DIAGRAM	8
7.	TRANSYM	12
	Appendix 1	17
	Appendix 2	23
	Appendix 3	39
	Appendix 4	43

A Phrase-Structure Language Translator

1 INTRODUCTION

Approximately five months ago many articles began appearing in computer journals regarding work being done on compilers which use a syntactical description to direct the translation of an input language (phrase-structure) to the output language. Although various people use somewhat different methods, the translation processes are all relatively short compared to previous compiler-translators, are elegant in form, and can easily be programmed to accept changes in the input language. On the other hand, they tend to be much slower than their specialized ancestors and consume large amounts of storage during operation.

In order to study these new techniques one of them was programmed. While they were all quite similar, the methods of E. T. Irons were chosen as a starting point for two reasons:

a. His notation for specifying the translation process is compact and easily handled.

b. The linked list which is built up during the parsing of the input string and which contains all information for the final output string seems to be easier to work with than other systems.

The two basic programs, DIAGRAM and TRANSYM, took only three weeks from start of writing to production runs. These will be described in detail in this paper.

(Received for publication, 14 June 1962)

The notation of Mr. Irons was used wherever possible to clarify references to his works. The program TRANSYM is very much like the program of Mr. Irons, and is fairly straightforward. DIAGRAM has been changed considerably, however, so it now bears only a general resemblance to its progenitor. It was DIAGRAM (and programs of similar nature in other systems) which was eating up so much time and storage. The program DIAGRAM described herein operates from 100 to 500 times faster than any others we know of. It uses about 40 % less storage for its output string and about 90 % less storage for its syntactical tables. These programs have been enlarged into a system which is being used regularly now, and it is this system that is described in this report.

2 GENERAL STATEMENT OF PURPOSE

The system was designed in the usual monitor fashion; i.e., several different translations can be done in one run on the computer. Each translation will ordinarily consist of the following parts:

(1) Specifying the allowable syntactical units.

(2) Syntactical description of the language; i.e., how the 'small' syntactical units fit together to form 'big' units.

(3) Specifying whence the input is to come and how it is to be converted into the input string.

(4) Performing the translation process with parameters stating:

   a. the 'goal'; i.e., the syntactical unit the input string is expected to form.

   b. what output converter is to be used and where the output is to go. The translation process produces an output string which is put out in its final form (printed, punched, written on tape, etc.) by the output converter.

A syntactical unit can be either the name of a single unit in the input string or a name assigned to a group of input units. Thus, we might call an 'A' simply 'A' (although we could as easily call it a 'B'); bigger units could be, say, 'word,' 'number,' 'sentence,' 'sum,' etc. The purpose of Part (1) is to assign a number to each of these units. Currently these are numbered 000-777 (octal) giving a maximum of 512 distinct units. Only 125 of these have been used to date, so this appears to be a reasonable selection. When the syntactical descriptions are read in, they are translated into strings of these numbers. Digits, letters, and punctuation characters should be assigned numbers corresponding to their input representation in order to facilitate input conversion (in the IBM 7090 this means an 'A' is assigned 021, etc.). Part (3) is dependent upon the individual application, although it usually takes the form of a character-to-character conversion with a provision for dropping extra blanks. Parts (2) and (4) occupy most of the program space, and these will be dealt with in separate sections.

3 SYNTACTICAL DESCRIPTION OF THE LANGUAGE

In this section the basic form of the syntax descriptions will be given. The form presently used is actually somewhat different from this in order to save space in the output of DIAGRAM, and to allow special interruptions for debugging and other uses. The basic form is essentially that of Irons, and his notation will be used.

The syntactical description consists of a sequence of 'sentences,' each having the form:

$$\underbrace{S_m S_{m-1} S_{m-2} \cdots S_2 S_1}_{\text{components}} \underbrace{= : : S}_{\text{subject}} \underbrace{\Longrightarrow \{ P\,P\,P\,P \ldots P\,P \}}_{\text{definition}}$$

Each S_i is a 'component,' a syntactical unit; the unit S following the = : : is called the 'subject' of the sentence. The meaning associated with the part of the sentence before the $\Longrightarrow$ is that if a section of the input string is of the form $S_m S_{m-1} \ldots S_1$, then that section can be called an 'S'. The string { P P P . . . P P } is the 'definition' of the sentence; it specifies the output for S in terms of output from the components. Usually each P will be a character of the output string language; however, certain characters have special significance. These are { } ℙ ℱ / ;] ← .

The units { and } will occur only when syntactical descriptions are to be put out. A counter in the output string generator keeps tract of the number of { 's minus the number of } 's. Let us call this number b. When a { is encountered, a { is output and b + 1 replaces b. When a } is encountered, b is replaced by b - 1 if the latter is not negative and a } is output. If b - 1 is negative (b = 0 before), then the output string generator TRANSYM has reached the end of that definition string and nothing is output. TRANSYM usually begins its action with the character following the leftmost { in a definition string with b = 0.

Most definitions for a subject require information from the definitions of its components, sometimes with substitutions being made in these definitions. The character P is used for this purpose. Assuming that the counter b is 0, the characters Pn] , where n denotes an octal interger, signal TRANSYM to obtain and output the definition corresponding to S_n (counting left n from the = : :). If substitutions are to be made in the definition for S_n, then the form is

$$P_n ; x_1 \leftarrow PP \ldots P ; x_2 \leftarrow PP \ldots P ; \ldots ; x_\ell \leftarrow PP \ldots P]$$

signifying that if an x_i is encountered it will be replaced with what is to the right of the ← up to the next ; or] . The strings P P . . . P are in each case

completely general, and may contain more P's, etc. If $b = b_1 \neq 0$ and the character P is preceded by b_1 /'s, then the /'s are ignored and the operation proceeds as stated above. If P is preceded by either no /'s or by b_1 /'s, where $b_2 \neq b_1$, then the /'s, the P, and all the characters up to and including the ‖ are output as is. This feature, in combination with the { 's and } 's, enables syntactical descriptions to be output.

Occasionally it is useful or necessary to perform some special task in the output, such as outputting the sum of several numbers, or counting the occurrences of a certain character in a definition, etc. The character $\mathcal{F}$ is used to denote a special function of this type. Assuming that $b = 0$, the characters

$$\mathcal{F}_n \; ; \; PP \ldots P \; ; \; PP \ldots P \; ; \ldots ; \; PP \ldots P \,‖ \quad \text{and} \quad \mathcal{F}_n \,‖$$

(m groups)

cause TRANSYM to call the function $\mathcal{F}_n$ of m arguments and no arguments, respectively. These special functions must be coded separately and put into storage when the other programs are read in at the beginning of the run. If $b \neq 0$, the use of /'s is exactly the same as with the character P.

4 TABLING THE SYNTACTICAL DESCRIPTION

Consider the following descriptions:

$$\alpha_1 \alpha_2 \alpha_3 \alpha_4 \alpha_5 \; = :: \; \alpha_6 \longrightarrow \{ \ldots \}$$

$$\alpha_1 \alpha_2 \alpha_8 \alpha_9 \alpha_{10} = :: \; \alpha_{11} \longrightarrow \{ \ldots \}$$

$$\alpha_1 \alpha_2 \alpha_8 \alpha_9 \alpha_{10} = :: \; \alpha_{13} \longrightarrow \{ \ldots \}$$

$$\alpha_1 \alpha_{15} \; = :: \; \alpha_{16} \longrightarrow \{ \ldots \}$$

No information is lost and constructional differences are made more apparent if we change this to:

$$\begin{array}{l} \alpha_1 \alpha_2 \alpha_3 \alpha_4 \alpha_5 \alpha_6 \quad \{ \ldots \} \\ \quad\; \alpha_8 \alpha_9 \alpha_{10} \alpha_{11} \; \{ \ldots \} \\ \qquad\qquad\; \alpha_{13} \; \{ \ldots \} \\ \alpha_{15} \alpha_{16} \quad \{ \ldots \} \end{array}$$

The items to notice are the following:

a. The characters = : : and $\Longrightarrow$ can be omitted if we agree that the character { can never occur as one of the α's. Thus the subject is the α before the first { and the definition follows that { .

b. The method of examination of the input string is made much more evident.

To elaborate on part (b) let us table the above information as follows:

i	STC (i)	STABS (i)
2	15	x_2
3	8	x_3
4	0	x_4
5	0	x_5
6	0	x_6
7	β_1	{
8	0	x_8
9	0	x_9
10	0	x_{10}
11	13	x_{11}
12	β_2	{
13	0	x_{13}
14	β_3	{
15	0	x_{15}
16	0	x_{16}
17	β_4	{

t	STABD(t)
	}
β_1	}
β_2	}
β_3	}
β_4	}

α	TRAN(α)
α_1	2

Let us assume that the unit α_1 has been constructed from the input. The purpose of the TRAN table is to index the STABS table, which contains the syntax units of the descriptions. Thus, if we are to construct something which begins with an α_1, TRAN(α_1) = 2 gives us the starting point in STABS and STC for the descriptions which begin with α_1. STABS(2) gives the next unit to be constructed from the input. Since STABS(2) = α_2 (and α_2 is not a subject) DIAGRAM will attempt to build an α_2 from the input. If α_2 cannot be constructed, if α_3, α_4, or α_5 cannot be constructed, of if α_6 does not lead to the desired final form,

then DIAGRAM will attempt to construct STABS (STC(2)) = STABS(15) = α_{15}. If the input string does yield the construction $\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5$, then DIAGRAM will recognize that it has constructed an α_6 and note in an output string that the corresponding definition begins in STABD(STC(6 + 1)) = STABD(β_1). If α_6 is not the end result and further construction is needed, DIAGRAM will begin building the unit found in STABS (TRAN (α_6)). Thus we see that the syntax descriptions are broken up into groups; every sentence in a given group begins with the same syntactical unit. TRAN of this unit refers to the location in STABS where the rest of the first sentence of the group begins; STC provides a linkage between sentences within a group. STABD may be regarded as an extension of STABS, and in fact it could be included in STABS (Irons had them in a combined table), but since they need not be in storage at the same time (DIAGRAM is all through with STABS before TRANSYM begins on STABD) it was considered best to separate them in case storage was needed later.

Consider the following sentences:

$$x_1\ x_2\ x_3 = x_4$$
$$x_2\ x_5\ x_6 = x_3$$
$$x_4\ x_5\ x_6 = x_4$$
$$x_3\ x_7\ x_8 = x_9$$
$$x_9\ x_5\ x_6 = x_1$$

On occasion it is necessary for DIAGRAM to know if a unit x_j can be constructed beginning with the unit x_i. This information is contained in a Boolean table called the SUCCR (successor) table. The SUCCR table for the above sentences would be

x_i \ x_j	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
x_1	0	0	0	1	0	0	0	0	0
x_2	1	0	1	1	0	0	0	0	1
x_3	1	0	0	1	0	0	0	0	1
x_4	0	0	0	1	0	0	0	0	0
x_5	0	0	0	0	0	0	0	0	0
x_6	0	0	0	0	0	0	0	0	0
x_7	0	0	0	0	0	0	0	0	0
x_8	0	0	0	0	0	0	0	0	0
x_9	1	0	0	1	0	0	0	0	0

and we see that SUCCR(1,4) = 1; SUCCR(1,5) = 0, SUCCR(2,3) = 1; SUCCR(5,j)=0 for all j, etc.

The table S indexes the table B which contains the substitutions in effect in TRANSYM. Before outputting an ordinary character x (not a P, F, etc.) TRANSYM first checks to see if x is entered in S; if not, then x is output; if x is in S, then S(u) gives the location in B of the string to be output instead of x. S and B will be described in detail later.

5 OTP

OTP is the name of the output from DIAGRAM. It is also the input to TRANSYM. OTP is a list containing all the information of the structure of the input string according to the syntactical descriptions in STABS. Its form determines the operation of both DIAGRAM and TRANSYM. For this reason it is essential that the reader understand OTP completely before continuing.

Entries in OTP can be of two types: direct and indirect. The indirect entries will be preceded by a − to distinguish between the two. Each direct entry is the address of a definition in STABD, that is, it is a β_i. Each indirect entry is the address of a (direct) entry in OTP. Suppose we have the descriptions:

a = : : letter ⟹ { A }

b = : : letter ⟹ { B }

c = : : letter ⟹ { C }

letter = : : ltrgrp ⟹ { P 1]] }

ltrgrp letter = : : ltrgrp ⟹ { P 2]] P 1]] }

If we give DIAGRAM the input string

a b c

and ask it to construct a ltrgrp (letter group), then OTP would be

1. β_2 ⟶ B }
2. β_3 ⟶ C }
3. β_5 ⟶ P 2]] P 1]] }
4. -2
5. β_5 ⟶ P 2]] P 1]] }
6. -1
7. β_4 ⟶ P 1]] }
8. β_1 ⟶ A }

TRANSYM would be told to start with OTP(3). To find P 2]] we look in OTP(3 + 2), where another P 2]] sends us to OTP(5+2). There a P 1]] sends us to OTP(7+1) which points to A } . The A is output and the } is found, so we go back to OTP(7). We are done with the P 1]] so we move on to the } here also; this sends us back to OTP(5). To find P 1]] we look at OTP(5+1), which is indirect and jumps us back to OTP(1). Here we output the B, find the } , and return to OTP(5). TRANSYM finds another } here and so exits back to OTP(3). Having done the P 2]], we move on to the P 1]] which is found in OTP(3+1)= -2. This jumps us to OTP(2) where we output C and return to OTP(3). We run into the } and exit to the controlling program.

The reader should thoroughly understand the above before proceeding. It contains the essence of the operation of TRANSYM as well as the details of OTP. When TRANSYM is told to output the definition pointed to directly by OTP(k) and somewhere in that definiton Pn is encountered, TRANSYM calls itself recursively to output the definition pointed to by OTP(k+n), first jumping to the direct link pointed to by OTP(k+n) if it is indirect.

6. DIAGRAM

Before outlining the operation of DIAGRAM, a few comments are in order since it generally operates quite differently from the program given by Irons in ACM (January 1961 Communications). The most important change is the lookahead feature, which for problems of useful size and complexity results in an increase in speed by a factor of from 100 to 500; the greater the complexity the greater the increase in speed. Two other changes have resulted in about a 40 % decrease in the size of OTP.

DIAGRAM is a function of seven arguments: i, goal, param, error, S_2, S_3, S_5. STABS(i) is the unit in STABS being examined. The goal is the syntactical unit which is being constructed. Param refers to the output cell of the level which called DIAGRAM to construct the goal. Error is where DIAGRAM will return control if it cannot construct the goal. S_2 tells whether or not lookahead has already been performed or not. S_3 and S_5 control what is done with the output.

The first thing done by DIAGRAM is to save information it will need when it exits. Since it is a recursive function these items are put into a push down list. The quantities saved are: all the arguments; j, the input string pointer; k, the OTP pointer. In addition, space is left for storage in PDL (push down list) of SW and S_4, and the i for the last call is saved so that it can be restored upon exit. This eliminates the storage of i every time it is changed, since this is not needed unless another call is made. The quantities i, j, and k are saved in I, J, and K,

respectively. Each section of DIAGRAM is given a name for reference purposes. The action of each section is explained in the following word flow chart:

DIAGRAM (i, GOAL, PARAM, ERROR, S_2, S_3, S_5)

DIAGR: i $\longrightarrow$ I, j $\longrightarrow$ J, k $\longrightarrow$ K, save i of last call in PAST I. Save GOAL, PARAM, ERROR, S_2, S_3, and S_5.

START: Test STABS(i+1). If it contains { then STABS(i) is a subject, so jump to STRT1. Otherwise test S_2 to see if branching is possible: F - no, T - yes. If F then jump to STRT8. If T then continue to the lookahead section after saving i in I_1. The lookahead section is from STRT3 through STRT9.

STRT3: Test STC(i+1); if non-zero jump to STRT9.
Test STABS(i+2); if it does not contain { then replace i with i+1 and go back to STRT3.

STRT4: At this point the following is known

a) STABS(I_1) is not a subject.

b) STABS(I_2+1) is a subject (where I_2 is the final value of i from the above loop).

c) STC(i) = 0 for $I_1 + 1 \leq i \leq I_2 + 1$. That is, no branching is possible once we have passed STABS(I_1).

Thus we can test STABS(I_2+1), the subject, to see if it <u>is</u> the goal or if it can <u>lead</u> to the goal, that is, we see if STABS(I_2+1) = GOAL or SUCCR (STABS(I_2+1), GOAL) is true. If either is the case, we set S_2 to F and go on to STRT9. Otherwise, we reset i to I_1 and go to STRT1, since it would be useless to pursue this branch further.

STRT9: Reset i to I_1.

STRT8: When we arrive here we prepare to start construction of STABS(i) from the input since we know that either no branching is possible and we definitely want to get to the subject, or else branching is possible (STC(i) $\neq 0$ for some i between I_1 and the subject) and hence it is too early to make a decision. Certain decisions are made about possible future output relating to STABS(i) at this point. S_3 is tested. If it is F then we have previously noted that no output for this unit is needed, so S_4, an output indicator for this level, is also set to F. If S_3 is T then T(i), which is a quantity stored in STABS(i) along with the syntactical unit, is tested. If T(i)=1 S_4 is set to F. If T(i)=2 we have hit a special function to be performed immediately, followed by a jump to CONTN. This is a special feature for debugging aids and two-pass systems. In this case STABS(i) is the name of a function rather than a syntactical unit. S_4 is also set to F when T(i)=2. If T(i) $\neq$ 1 or 2 S_4 is set to T and we go on to SMALJ.

SMALJ: j is replaced with j+1 in preparation for looking at the next input. INPUT(j) is obtained and compared with STABS(i). If they are equal the input unit itself is the syntactical unit we desired, so SW is set to T. If not, SW is set to F. In any case, we need to know if we can build STABS(i) from the string of characters starting with INPUT(j), for even if SW=T we may be able to build a bigger STABS(i) than just INPUT(j) alone. Thus SUCCR (INPUT(j), STABS(i)) is performed. If it yields F (false) we skip to NOGO, if T we go on to STRT2.

STRT2: At this point we have the first character of STABS(i), so we call DIAGRAM (TRAN(INPUT(j)), STABS(i), OTCEL, NOGO, T, S_4, S_6) that is, we call DIAGRAM to proceed with the construction with STABS(i) as goal, the output cell of this level as param, NOGO as the error exit, S_2 is true (since we know nothing about the branching for the new sentence we are going to be working on), S_4 is the S_3 for this level, and S_6 is obtained as follows: the value of S_6 is F if either S_5 = F or T(i) = 3, T if S_5 = T and T(i) $\neq$ 3. S_5 and S_6 pertain to the output in a manner illustrated in a subsequent example and will not be dealt with here. If DIAGRAM is successful it returns to CONTN, where we go on the STABS(i+1).

NOGO: We arrive here when we cannot build STABS(i) starting with INPUT(j). SW is tested; if SW=T, then INPUT(j)=STABS(i) and we go to CONTN. If SW=F, then we must abandon this branch. This is done at RETRA.

RETRA: J $\longrightarrow$ j, K $\longrightarrow$ k; that is, reset the input and output indexes to the values they had at the beginning of this call. We have discovered that STABS(i) could not be constructed, so we reset j and k and proceed to STRT1.

STRT1: STC(i) $\longrightarrow$ i, and test i. That is, we replace i with STC(i) and then test this new value of i. If it is non-zero branching is possible, so set S_2 to T and go back to START to try to construct the alternate syntactical element STABS(i). If i is 0, reset i to I, the original value of i, and go to NEWST.

(NEWST): When we arrive here we know that nothing can be constructed. If STABS(I) was a subject, or could not be built, we looked at STABS (STC (STC(...STC(I)...))) until no more branches were left. Now we loop back through the same branches, but this time we look for subjects instead of components to be constructed. For every subject we attempt to start the construction corresponding to a sentence starting with that subject. We proceed as follows:

NEWST: Is STABS(i+1) = { ? If not, we jump to NPTH1. Otherwise STABS(i) is a subject, and we put STC(i+1) into OTCEL, the output cell. This is the index pointing to the definition in the STABD table which corresponds to the sentence just assembled. Next set S_4 to T and test STABS(i) to see if it is the goal we are seeking. If STABS(i) $\neq$ GOAL we set SW to F and test S_2. If S_2 is F then we have already performed the SUCCR of NEWS4 and found it true when we were doing the lookahead, so skip to NEWS3. If S_2 is T, we go to NEWS4. If STABS(i) = GOAL, SW is set to T and -K is put into PARAM, which is the output cell of the level which called for the construction of the current goal. Proceed to NEWS4.

NEWS4: Perform SUCCR (STABS(i), GOAL). If no, jump to NOPTH; if yes, proceed to NEWS3.

NEWS3: SUCCR was true, so we can build the goal starting with STABS(i). If STABS(i) = GOAL this means the construction of GOAL is recursive. We perform

DIAGRAM (TRAN (STABS(i)), GOAL, PARAM, NOPTH, T, S_3, T) to try to proceed with the construction of the goal. If this call is successful, go to FOUND; otherwise to NOPTH.

NOPTH: GOAL cannot be constructed starting from STABS(i), so we test SW to see if STABS(i) = GOAL (thus SW saves us from having to perform this test twice). If SW is true, go to FOUND; if false, go to NPTH1.

NPTH1: Replace i with STC(i) and test If the new value of i is non-zero, go to NEWST. If it is zero, reset j to J, k to K, and go to ERROR, the error exit for this call.

CONTN: STABS(i) has been found, so we call DIAGRAM to continue to STABS(i+1). So we perform

DIAGRAM (i+1, GOAL, PARAM, RETRA, S_2, S_3,T)

The error exit is to RETRA, the normal return is to FOUND.

FOUND: If S_3, S_4, and S_5 are all true, OTCEL is put into OUTPUT(k), which is the $k^{\underline{th}}$ element of OTP, and k is replaced with k+1. If S_3 or S_4 is false, no output is put out and k is not changed. If S_5 is false OTCEL is put into PARAM and k is not changed. After any of these possibilities a normal return is executed.

7 TRANSYM

Like DIAGRAM, TRANSYM is a recursive function, although it generally does not use nearly as many recursions. Its input is OTP and its output is a string which is generally fed character by character directly into the output converter. A line is printed, card is punched, etc., as soon as enough characters are generated, thus preventing a build-up of characters waiting to be output. Some of the characters are treated differently from others in that they cause special actions. Examples of these are: output current card, skip to Column 8, etc. These are dealt with in the appendixes.

The arguments of TRANSYM are N, RR, PUTIKER, and P. RR is the address of a direct link in OTP, STABD(N) is the first definition unit TRANSYM is to look at on this call, and PUTIKER is a Boolean variable controlling the output. If PUTIKER is T, the output goes to the output converter; if F, the output is put into OTPX, a temporary storage area for holding output to be later transferred to the substitution table or output which serves as the arguments for the special functions F_n. P is described below.

TRANSYM reserves certain information in a stack (i.e., push-down space) just as DIAGRAM does. These are the following nine quantities: N, RR, PUTIKER, EXIT, VKEEP, TE, SUBXKEEP, SUBLKEEP, P, and a quantity x_o which is really part of F_1 and F_2 and will be discussed in the appendixes. EXIT is exactly what the name implies and P is the counter for the left and right metabraces = (# of { 's) - (# of } 's) - 1. The -1 appears since P=0 at the start of a definition and we start with the character following the initial { . SUBXKEEP and SUBLKEEP (hereafter referred to as S_o and U_o) are the addresses of the last-used spaces in the substitution list and the index to that list, respectively. TE is a cell used for the saving of (essentially) the n found after a P or F. VKEEP is used for remembering the value V, the OTPX index, for later use.

TRANSYM initially sets t equal to the value N and goes to PI.

PI: A test is made of STABD(t). If the character is a { we go to WASLB,
if a } we go to WASRB,
if a / we go to WASPR,
if a P we go to WASSP,
if an F we go to WASSF,
if a ; we go to RETRN
if a]] we go to RETRN, and if the character is none of these (it is called a 'simple symbol'), we go to SIMSM.

SIMSM: OUTSUB(t) is performed; t+1 replaces t and N, and we go back to PI.

OUTSUB(t): This subroutine checks the substitution indexing list S(u) to see if the character STABD(t) occurs; if it does, the appropriate substitution is made. The mechanism is as follows:

OUTSB: U is set equal to U_0 (from SUBLKEEP).

OTS3: Test u; if u=0 output STABD(t) and return to calling routine. Otherwise compare STABD(t) with S(u); if not equal, decrease u by 1 and go back to OTS3. If equal, then there is a substitution to be made. The list A, which runs parallel to S, contains the address of the first character of the substitution string.

OTS2: We set s=A(u). s is the variable running through the substitution strings which are stored in area B.

OTS4: B(s) is tested to see if it is the character ENDSYM (octal 777 is reserved for this special character). If it is, an exit to the calling routine is performed. If not, then B(s) is output, s+1 replaces s, and we go back to OTS4.

OUTPUT(x): This subroutine tests the current value of PUTIKER. If T, x is fed to the converter. If F, x is put into OTPX(v) and v+1 replaces v.

There is also a special subroutine OTPTN, which is the same as OUTPUT except that it assumes the value F for PUTIKER.

WASLB: Increase P by 1; output a { ; replace t and N with t+1; return to PI.

WASRB: Test P; if 0 go to RETRN; if not zero decrease P by 1, output a }; replace t and N with t+1, and return to PI.

RETRN: We have come to the end of this call of TRANSYM, so an exit is performed.

WASPR: Proceed to count the number of primes. q is set equal to 1, and t+1 replaces t. STABD(t) is tested; if it is a ', q and t are increased by one and the process repeated. When a character other than a prime is finally encountered, q is compared with P. If q=P, STABD(t) is tested. If it is a P or F a jump is performed to WSSPI or WSSFI, respectively; otherwise, an error is noted. If $q \neq P$, then q primes are output, and we go to WSPR4.

WSPR4: The P (or F) is output along with all other characters up to and including the]] which matches the P (or F). t is increased by one with every character that is output. The final value of t, then, is such that STABD(t) is the character after the]]. N is set to this final value of t and control is transferred back to PI.

WASSP: We come here knowing that STABD(t) = ρ and that there are no primes before it. So P is tested. If P $\neq$ 0, transfer to WSPR4. If P = 0, go on to WSSP1.

WSSP1: The value of the octal integer n following ρ is obtained, added to RR, and the sum stored in TE. This is the address in OTP of the link pointing to the definition of ρ_n. The link may be either direct or indirect; this will be examined later. t is increased to point to the character after n.

WSSP3: STABD(t) is examined. If STABD(t) =]], control is transferred to DOP; if STABD(t) = ; we proceed; otherwise, an error condition is noted. At this point VKEEP (hereafter referred to as V_o) is set equal to v, the current value of the OTPX index. Then t is increased by one and STABD(t), the character for which the substitution is to be made, is put into OTPX(v) and v+1 replaces v (this is done through the OTPTN (STABD(t)) subroutine). Again t is increased, and STABD(t) is checked to see that it is the character $\Leftarrow$. If it is, t is increased by one and we proceed; if not, an error condition is noted. At this point STABD(t) is the first character of the string which is to be substituted for the character in OTPX(V_o). Now this string is arbitrary; that is to say, it may be null (so that STABD(t) = ; or]]), or it may be simple characters, but it may contain ρ's and $\mathcal{F}$'s which will need further translation. For this reason it is necessary to have TRANSYM analyze the string. This is done by performing TRANSYM (t, RR, F, P), which starts translating with STABD(t) and continues until it gets to either ; or]] , whereupon it returns to the present call. All output from this translation is put into OTPX starting at V_o+1 since PUTIKER = F. Note that all substitutions which are to be made as a result of previous calls are done during this translation, so that the string in OTPX is in need of no further analysis. OTPTN (ENDSYM) is performed, thus putting the character ENDSYM at the end of the string in OTPX. We have now to transfer this string from OTPX to the area B and put the proper entries into S and A. This is done as follows:

a.) Set v_1 equal to the current value of v,
set v_2 equal to v_o+1, and
set s equal to s_o .

b.) Increase s by one.

c.) Put OTPX(v_2) in B(s).

d.) Increase v_2 by one and test. If it is less than v_1 , go back to (b); if $v_2 = v_1$, go to (e).

e.) Replace U_o with U_o+1.

f.) Put OTPX (v_o) in S (U_o) and put S_o in A (U_o).

g.) Replace S_o with S.

Thus we see that U_o and S_o always point to the last used cells in S and A, and B, respectively, while v points to the next vacant cell in OTPX.

Since we are now done with the string in OTPX, v is reset to the value v_o, and a transfer to WSSP3 is executed. Note that t is left pointing to the ; or]] following the substitution string analyzed by the TRANSYM call mentioned above.

DOP: N is set equal to t+1, which points to the character following the]]. This is for resetting t after the next call of TRANSYM. We are now ready to get the definition corresponding to $\mathcal{P}_n$. To do this we first examine OTP(TE). If this is indirect (i.e., negative) then we set TE = |OTP(TE)|. If it is direct we leave it alone. OTP(TE) now points directly to the first character in the definition for $\mathcal{P}_n$. Thus we perform TRANSYM (OTP(TE), TE, PUTIKER, 0), which gets the appropriate output and puts it in OTPX or gives it to the output converter, according to the current value for PUTIKER. After performing this we go to PI2.

WASSF: At this point it is known that STABD(t) = $\mathcal{F}$ and that no primes immediately precede this $\mathcal{F}$. Therefore P is tested. If P ≠ 0, transfer to WSPR4. If P = 0, go on to WSSF1.

WSSF1: The value of n is obtained, added to FUNBASE, and the sum stored in TE. This is the address of the first instruction for $\mathcal{F}_n$, and control will be transferred there after the arguments (if there are any) have been translated. We set v_o to the current value of v and go to WSSF2.

WSSF2: Test STABD(t). If it is]], go to DOF. If it is ; go on; otherwise indicate error. t is increased by one and TRANSYM (t, RR, F, P) is called. This translates the argument string and puts the output in OTPX, then returns with t pointing to the ; or]] at the end of the argument. OTPTN (ENDSYM) is performed, thus placing ENDSYM at the end of the completely formed argument in OTPX. Control is then transferred back to the beginning of WSSF2.

DOF: v is reset to v_o, N is set to t+1, and a transfer to the address in TE is executed. v_o is in the accumulator (or some other known place), and this tells $\mathcal{F}_n$ where the arguments begin in OTPX. $\mathcal{F}_n$ further knows that ENDSYM's separate the arguments. $\mathcal{F}_n$ returns to PI2.

PI2: Set t = N and go to PI.

Appendix 1

SPECIFIC INSTRUCTIONS FOR OPERATION OF PRESENT SYSTEM

The present system is not in a form which is considered useful for general programmers. Special functions and modifications for input-output control must be put in as mod-cards using the SOS system. This is difficult for users who do not know the IBM 7090 very well. Several methods for making the system easier to use are presently being studied, and results of these studies will be supplied later. This paper describes only that which is currently in use.

The general procedure for running is

a) a definition 1 call.

b) a syntax generator call.

c) a definition 2 call.

d) a diagram call (this calls transym if it is successful, otherwise comes back to the directing program).

The directing program, which has been named SEER for no particular reason, recognizes the following control cards:

DEFN1
SNTXG
DEFN2
DIAGR, GOAL, PUNCH,
DIAGR, GOAL, PRINT,
DONE
DUMP
LOG1
LOG2
LOG3

Each begins in card column 8. The only variable in the above is GOAL, for which is substituted the name of the actual syntactical element to be found by diagram. We will study the action of the programs called when these cards are encountered.

DEFN1

The routine DEFN1 sets up a 1-1 correspondence between the names to be used for syntactical elements and the nine-bit binary numbers used by the machine to refer to those elements. These names will be used only in the SNTXG part for inputting the descriptions, except for the elements which occur as goals on the DIAGR call cards.

The input to DEFN1 starts on the card following the call card. A name is read in and assigned the number currently in a counter; then the counter is advanced one and the next name input, etc. A name may consist of any number of characters, although only the first six are regarded as significant. A name is terminated by a comma, and blanks following the comma and preceding the next name are ignored. Column 72 of one card is regarded as contiguous with column 1 of the next. While =, +, −, and / are themselves ligitimate names, no name of more than one character may begin with one of them. However, they may occur within a name. Of course a comma "," may not occur anywhere within a name. The reasons for the +, −, and / being excluded from the first character of a name will be evident from SNTXG, but the = character has a special significance in DEFN1, for it is used to set the counter. It may be followed by one of two characters, O or D. This letter is in turn followed by from one to three digits. If the letter was an O the digits are regarded as octal, if a D, as decimal. The digits are followed by a comma. Thus =O0, =O177, =D10, =D511, set the counter to 000, 177, 012, and 777, respectively (all octal). Note that 511 is the largest number which can follow =D, and 777 that can follow =O.

DEFN1 is ended by putting the character string DFIEND in as the last name. When DEFN1 finds this name it immediately returns control to SEER.

SNTXG

The problems that arose in reading in the syntax description were mostly the result of the small character set available. This was the case particularly in the definition part of the descriptions. The best way to explain the method used is to give an example. Consider the following syntax statements:

symbol = sum = : : stmnt $\Longrightarrow$ { $\mathbb{P}_1$ ‖ STO $\mathbb{P}_3$ ‖ $\mathbb{R}$ }

symbol x term = : : term $\Longrightarrow$ { $\mathbb{P}_1$ ‖ XCA $\mathbb{R}$ FMP $\mathbb{P}_3$ ‖ $\mathbb{R}$ }

symbol x symbol = : : term $\Longrightarrow$ { LDQ $\mathbb{P}_3$ ‖ $\mathbb{R}$ RMP $\mathbb{P}_1$ ‖ $\mathbb{R}$ }

symbol = : : sum $\Longrightarrow$ { CLA $\mathbb{P}_1$ ‖ $\mathbb{R}$ }

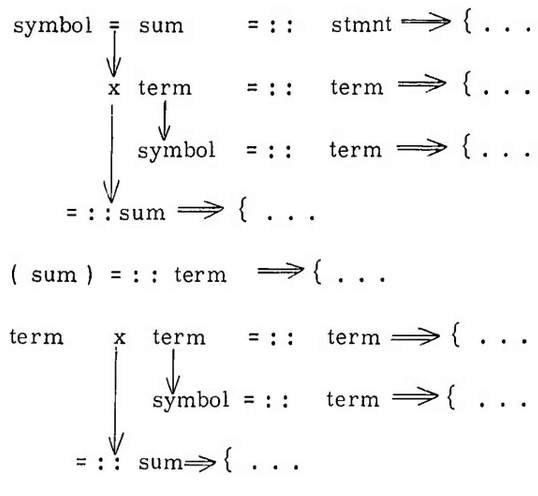
(sum) = : : term $\Longrightarrow$ { $\mathbb{P}_2$ ‖ }

term x term = : : term $\Longrightarrow$ { $\mathbb{P}_3$ ‖ STO T $\mathbb{R}$ $\mathbb{P}_1$ } T $\overleftarrow{\mathcal{T}_1}$ ‖ ‖ XCA $\mathbb{R}$ FMP T $\mathbb{R}$ }

term x symbol = : : term $\Longrightarrow$ { $\mathbb{P}_3$ ‖ XCA $\mathbb{R}$ FMP $\mathbb{P}_1$ ‖ $\mathbb{R}$ }

term = : : sum $\Longrightarrow$ { $\mathbb{P}_1$ }

where $\mathbb{R}$ stands for a carriage return (or end of line) signal to the output converter. Writing this so that the STC links are clearly evident we get



where the definitions have been omitted for clarity.

The input corresponding to these statements is

```
SYMBOL, =, SUM, = STMNT, (/P1/, STO/P3/, /*776)
       , x, TERM, = TERM, (/P1/, XCA/*776FMP/P3/, /*776)
       , , SYMBOL, = TERM, (LDQ/P3/, /*776FMP/P1/, /*776)
       , = SUM, (CLA/P1/, /*776)
(, SUM, ), = TERM, (/P2/, )
TERM, x, TERM, = TERM, (/P3/, STO T/*776/P|, T/./F|/, /, XCA
                                        /*776FMP T/*776)
    , , SYMBOL, = TERM, (/P3/, XCA/*776FMP /P|/, /*776)
    , = SUM, (/P|/, )
```

Note that consecutive syntactical elements are separated by commas, and that blanks between commas and between a comma and the name following it are ignored. Note that the commas specify where an STC link is to fall, since it points to the first name in a description (i.e., the first thing not blanks or commas) from the corresponding element in one of the preceding descriptions. This corresponding element is found by going upward until we find a non-null name which is preceded by the same number of commas. If the name is preceded by a -, T (i) is set to 1 for i corresponding to where that element is put into STABS. If preceded by a /, T (i) is set to 3. If an octal number is preceded by a +, the T (i) is set to 2, and the octal number is put into STABS; that is to say, the octal number is not considered as a name. This octal number corresponds to an interal function called by DIAGRAM. The name of the subject is preceded immediately by the character =,

and following the subject the definition begins with the character (, which may be preceded by an arbitrary number of blanks. Starting with that (, the characters are treated differently than those before the definition. The relationship is as follows:

/P	is translated to	𝕻	(750_8 internally)
/F	" " "	𝓕	(751)
(	" " "	{	(754)
)	" " "	}	(755)
'	" " "	'	(757)
,	" " "	;	(752)
/,	" " "	‖	(753)
/.	" " "	←	(756)
/*a	" " "	a	(where a is any character except 0-7. Thus /*(goes in as (, not {).
/*ddd	" " "	(ddd)	(where ddd is a three digit octal number)

If slash (/) is followed by any character other than P, F, comma, or *, the slash and the character go on into STABD as ordinary characters. Thus

/P1/,STO /P3/,/*776)

(remember that the initial { does not go into STABD)

becomes: 750 001 753 062 063 046 060 750 003 753 776 755

corresponding to: 𝕻 1 ‖ S T O 𝕻 3 ‖ 𝕽 }

The use of /* ddd is very important, since it enables us to use characters 100-747 and 760-776.

The characters 760-776 have been reserved for format control of the output. Of these only 773 to 776 have been assigned. The output converter recognizes these and performs the following operations accordingly:

776 output current line (card) and set up for next.

775 skip to column 2 of card.

774 skip to column 8.

773 skip to column 16.

At present the output converter recognizes no other characters but these. It simply tests the character to see that it is below 100; having ascertained this the initial 0 is dropped and the remaining two digits are put into the final output string. These two digit combinations are simply the binary coded decimal representations for the output alphabet. When 72 characters are obtained they are output as a line (or card), except that 776 also causes an output. If the output converter is fed a character between 100 and 772 it signals an error condition.

The end of the syntax descriptions is signaled by putting SNTEND anywhere on a card after the last description card.

DEFN2

In order to understand the action of this routine the reader must first learn more about the input converter. Ordinarily it simply takes the BCD representation for the input character (21 for A), appends an initial zero so that it has the required nine bits (021 for A), and puts this in the input string for DIAGRAM. However, the prime (', or 14 in BCD) causes a different action. The input is read in until the next prime is reached, and the characters are treated as forming a symbol. Thus 'ALPHA' is the symbol alpha, 'X1' is the symbol X1. The input converter then looks for this symbol (only the first 6 characters if it has more than 6 characters) in a table. When found, the corresponding number is put into the input string. This table is constructed as follows:

symbol
number
symbol
number
.
.
.
.
.

It is this table that DEFN2 constructs. The input, which starts anywhere on the card following the DEFN2 card, consists of pairs. S, ddd, S,ddd, S,ddd, ... , where S is a symbol of arbitrary length, and ddd is a three digit octal number. Any number of blanks may precede an S, none may be between the comma and the octal number. A symbol is assigned the number which follows it. Thus ALPHA, 012, X1, 124 yields the table entries

0 ALPHA
0 1 2
0 0 0 0X1
1 2 4
.
.
.

so that the input 'X1' = 2+'ALPHA' results in the input string
1 2 4 0 1 3 0 0 2 0 2 0 0 1 2.

DEFN2 returns control to SEER when it discovers the symbol DF2END, so these cannot be used as the first six characters of an input symbol.

DIAGR, GOAL, PUNCH, (or DIAGR, GOAL, PRINT,)

This card causes the following operations to be performed

(a) The SUCCESSOR table is built up.

(b) The next card is checked. If it contains only ENDS in columns 8-11 and no other characters on the card, control is returned to SEER. Otherwise control goes to the input converter.

(c) The input converter puts the input into the input string, reading in new cards as necessary, until it hits a card having ENDM in columns 8-11 and the rest blank. Then we proceed to (d).

(d) DIAGRAM (TRAN(INPUT (1)), GOAL, SEER1, EROES, T, T, T) is called. This will result in the input being diagrammed and a normal return if everything is all right. However, if DIAGRAM is unsuccessful and it returns to ERORS, an error message is printed out and we go back up to (b). After a normal return TRANSYM (D(SEER1), (SEER1), T, 0) is called. (SEER1) is simply the number (changed to plus) put into SEER1 by DIAGRAM, and is the address of the direct link in OTP which points to the definition TRANSYM is to start with. D(SEER1) is the index of the first character in that definition. After TRANSYM ends its action, return to (b) above.

DONE

This card returns control to the SOS monitor.

DUMP

This card causes a complete dump of core, using the SOS macro instruction "CORE."

LOG1, LOG2, and LOG3

These cards are used to print comments on line. LOG1 causes the card it is on plus the next card to be printed, then returns to SEER. LOG2 is the same except that following the second line it causes an eject. LOG3 is the same as LOG2 except that following the eject the computer executes a HPR (halt and proceed) instruction. Pushing the start returns control to SEER. This can be used to give instructions to the operator.

Appendix 2

Consider the following syntactical statements:

```
A, = LETTER,(A)
B, = LETTER,(B)
C, = LETTER,(C)
0, = DIGIT,(0)
1, = DIGIT,(1)
2, = DIGIT,(2)

    LETTER, DIGIT, = SYMBOL, (/P2/,P1/,)
    SYMBOL, =, SUM, = STMNT,(/P1/,STO/P3/,/*776)
            , x, TERM, = TERM,(/P1/,XCA/*776FMP/P3/,/*776)
            ,  , SYMBOL = TERM,(LDQ/P3/,/*776FMP/P1/,/*776)
            ,+ , TERM, = SUM,(/P1/,FAD/P3/,/*776)
            ,  , SYMBOL = SUM,(CLA/P3/,/*776FAD/P1/,/*776)
            ,= , SUM,(CLA/P1/,/*776)
    TERM, x, TERM, = TERM,(/P3/,STO T/*776/P1,T/./F1/,/,XCA/*776FMP
                                                        T/*776)
          ,  , SYMBOL, = TERM,(/P3/,XCA/*776FMP/P1/,/*776)
          , =  SUM,(/P1/,)
    SUM,  +, TERM, = SUM,(/P1/,STO T/*776/P3,T/./F1/,/,FAD T/*776)
        ,  , SYMBOL,= SUM,(/P3/,FAD/P1/,/*776
    (,SUM,), = TERM,(/P2/,)
```

The function $\mathcal{F}_1$ generates a new symbol each time it is called. For example, it could generate z0001, z0002, z0003, etc. for consecutive calls.

The input we will consider is:

```
1 2 3  4 5  6  7 8 9 10 11  2 3 4 5 6 7 8  9  20 1 2 3 4 25
B2 = A1 + (B0 X ( A0 +C2) ) + B1 X A2.
```

The initial call of DIAGRAM would be

DIAGRAM(TRAN(INPUT(1)), STMNT, O_0, ERORS,TRUE,TRUE)

The tables generated for this syntax are the following, with STABS and STABD combined. Note that this means that when a direct link is put into an OTCEL, it will be i + 2 instead of STC (i+1).

TRAN [α]

α	
A	1
B	5
C	9
0	13
1	17
2	21
letter	25
symbol	35
term	133
sum	191
(	235
x	0
+	0
=	0
)	0
digit	0
stmnt	0

SUCCR [α, β]

α \ β	A	B	C	0	1	2	letter	symbol	terms	sum	stmnt	(	x	+	=	)	digit
A	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0
B	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0
C	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
letter	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
symbol	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
term	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
sum	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
(	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
digit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
stmnt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

STC(i) and STAB(i)

In the following, which is primarily a STAB table, almost all the STC entries are zero. For this reason the nonzero entries have simply been enclosed in parenthesis and placed after the corresponding STAB entry.

	0	1	2	3	4	5	6	7	8	9
00	-	letter	{ (3)	A	}	letter	{ (7)	B	}	letter
10	{(11)	C	}	digit	{ (15)	0	}	digit	{ (19)	1
20	}	digit	{ (23)	2	}	digit	symbol	{(28)	ℙ	2
30	][	ℙ	1	][	}	=(51)	sum	stmnt	{ (39)	ℙ
40	1	][	S	T	0	Δ	ℙ	3	][	ℝ
50	}	x(91)	term(71)	term	{ (55)	ℙ	1	][	X	C
60	A	ℝ	F	M	ℙ	Δ	ℙ	3	][	ℝ
70	}	symbol	term	{ (74)	L	D	Q	Δ	ℙ	3
80	][	ℝ	F	M	P	Δ	ℙ	1	][	ℝ
90	}	+(243)	term(107)	sum	{ (95)	ℙ	1	][	F	A
100	D	Δ	ℙ	3	][	ℝ	}	symbol	sum	{(110)
110	C	L	A	Δ	ℙ	3	][	ℝ	F	A
120	D	Δ	ℙ	1	][	ℝ	}			
130				x(185)	term(166)	term	{(137)	ℙ	3	][
140	S	T	0	Δ	T	ℝ	ℙ	1	;	T
150	←	ℱ	1	][	][	X	C	A	ℝ	F
160	M	ℙ	Δ	T	ℝ	}	symbol	term	{(169)	ℙ
170	3	][	X	C	A	ℝ	F	M	P	Δ
180	ℙ	1	][	ℝ	}	sum	{(187)	ℙ	1	][
190	}	+	term(220)	sum	{(195)	ℙ	1	][	S	T
200	0	Δ	T	ℝ	ℙ	3	;	T	←	ℱ
210	1	][	][	F	A	D	Δ	T	ℝ	}
220	symbol	sum	{(223)	ℙ	3	][	F	A	D	Δ
230	ℙ	1	][	ℝ	}	sum	)	term	{ (239)	ℙ
240	2	][	}	sum	{(245)	C	L	A	Δ	ℙ
250	1	][	ℝ	}						

OPERATIONS of DIAGRAM

In the following table L designates the recursion level, i is the position in STAB and STC, P is the parameter relating to indirect entries for OTP, EE is the error exit, NE is the normal exit, j is the input index, k is the output index, and S(a, b) stands for SUCCESSOR (a, b).

L	i	GOAL	P	EE	NE	STAB(i)	j	k	S_2	REMARKS
1	5	stmnt	0_0	ERORS	(SEER)	letter	1	1	T	letter is subject; STC(5)=0; 7 → 0_1 ; F → SW_1; S(letter, stmnt)=T, so go on.
2	25	stmnt	0_0	NOPATH	FOUND	digit	1 2	1	T F	F → S_2; 2 → j; input(2)=2; F → SW_2; S(2, digit)=T, so go build digit.
3	21	digit	0_2	NOGO	CONT.	digit	2	1 2	T	digit is subject; STC(21)=0; 23 → 0_3; -1 → 0_2; T → SW_3; 23 → output(1); 2 → k; normal return to continue.
2										CONTINUE
3	26	stmnt	0_0	RETRACE	FOUND	symbol	2	2	F	subject; STC(26)=0; 28 → 0_3 ; F → SW_3.
4	35	stmnt	0_0	NOPATH	FOUND	=	2	2	T	F → S_2; 3 → j; input(3) is = ; T → SW_4; S(=, =) is false; CONTINUE.
5	36	stmnt	0_0	RETRACE	FOUND	sum	3 4	2	F	4 → j; input(4)=A; F → SW_5; S(A, sum) is true; go construct sum.
6	1	sum	0_5	NOGO	CONT.	letter	4	2	T	subj.; STC(1)=0; 3 → 0_6; F → SW_6; S()=T.
7	25	sum	0_5	NOPATH	FOUND	digit	4 5	2	T F	F → S_2; 5 → j; input(5)=1; F → SW_7; S(1, digit)=T; go build digit.
8	17	digit	0_7	NOGO	CONT.	digit	5	2 3	T	subj.; STC(17)=0; 19 → 0_8; -2 → 0_7; T → SW_8; S(digit, digit)=F; 19 → output(2); 3 → k. Normal return.
7										CONTINUE.
8	26	sum	0_5	RETRACE	FOUND	symbol	5	3	F	subj.; STC(26)=0; 28 → 0_8; F → SW_8; S()=T.
9	35 51 91	sum	0_5	NOPATH	FOUND	= x +	5 6	3	T	Lookahead fails; STC(35)=51 → i; 6 → j; input(6)= +; but + ≠ x and S(+, x)=F, so STC(i)=91 → i; T → SW_9; S(+, +)=F; CONT.
10	92	sum	0_5	RETRACE	FOUND	term	6 7	3	T F	F → S_2; 7 → j; input(7)= (; F → SW_{10}; S((, term)=T; go build term.

OPERATIONS of DIAGRAM (Contd)

L	i	GOAL	P	EE	NE	STAB(i)	j	k	S_2	REMARKS
11	235	term	0_{10}	NOGO	CONT.	sum	7 8	3	T F	F → S_2; 8 →j; input(8)=B; F → SW_{11}; S(B, sum)=T; go build sum.
12	5	sum	0_{11}	NOGO	CONT.	letter	8	3	T	subj.; STC(5)=0; 7 → 0_{12}; F → SW_{12}; S()=T.
13	25	sum	0_{11}	NOPATH	FOUND	digit	8 9	3	T F	F → S_2; 9 → j; input(9)=0; F → SW_{13}; S(0; digit)=T; go build digit.
14	13	digit	0_{13}	NOGO	CONT.	digit	9	3 4	T	subj.; STC(13)=0; 15 → 0_{14}; -3 → 0_{13}; T → SW_{14}; S()=F; 15 → output(3); 4 →k.
13										CONTINUE
14	26	sum	0_{11}	RETRACE	FOUND	symbol	9	4	F	subj.; STC(26)=0; 28 → 0_{14}; F → SW_{14}; S(symbol, sum)=T, so go on.
15	35 51	sum	0_{11}	NOPATH	FOUND	= x	9 10	4	T	Lookahead causes STC(35)=51 → i; 10 → j; input(10)=x; T → SW_{15}; S(x, x)=F; CONT.
16	52	sum	0_{11}	RETRACE	FOUND	term	10 11	4	T F	F → S_2; 11 → j; input(11)= (; F → SW_{16}; S((, term)=T; go build term.
17	235	term	0_{16}	NOGO	CONT.	sum	11 12	4	T	F → S_2; 12 →j; input(12)=A; F → SW_{17}; S(A, sum)=T; go build sum.
18	1	sum	0_{17}	NOGO	CONT.	letter	12	4	T	subj.; STC (1)=0; 3 → 0_{18}; F → SW_{18}; S()=T.
19	25	sum	0_{17}	NOPATH	FOUND	digit	12 13	4	T F	F → S_2; 13 → j; input(13)=0; F → SW_{19}; S(0, digit)=T; go build digit.
20	13	digit	0_{19}	NOGO	CONT.	digit	13	4	T	subj.; STC(13)=0; 15 → 0_{20}; -4 → 0_{19}; T → SW_{20}; S( )=F; 15 → output(4); 5 →k.
19										CONTINUE.
20	26	sum	0_{17}	RETRACE	FOUND	symbol	13	5	F	subj.; STC(26)=0; 28 → 0_{20}; F → SW_{20}.
21	35 51 91	sum	0_{17}	NOPATH	FOUND	= x +	13 14	5	T	Lookahead causes STC(35)=51 →i; 14 → j; input(14)=+; F → SW_{21}; S(x, +)=F; so STC(51)=91 → i; T → SW_{21}; S( )=F; CONTINUE.
22	92	sum	0_{17}	RETRACE	FOUND	term	14 15	5	T F	F → S_2; 15 → j; input(15)=C; F → SW_{22}; S(C, term)=T, so go build term.
23	9	term	0_{22}	NOGO	CONT.	letter	15	5	T	11 → 0_{23}; F → SW_{23}; S( )=T; go on.
24	25	term	0_{22}	NOPATH	FOUND	digit	15 16	5	T F	F →S_2; 16 → j; input(16)=2; F → SW_{24}; S(2, digit)=T; go build digit.

OPERATIONS of DIAGRAM (Contd)

L	i	GOAL	P	EE	NE	STAB(i)	j	k	S_2	REMARKS
25	21	digit	0_{24}	NOGO	CONT.	digit	16	5 6	T	23 → 0_{25}; -5 → 0_{24}; T → SW_{25}; S( )=F; 23 → Output(5); 6 → k; normal exit.
24										CONTINUE
25	26	term	0_{22}	RETRACE	FOUND	symbol	16	6	F	28 → 0_{25}; F → SW_{25}; go on.
26	35 51 91 243	term	0_{22}	NOPATH	FOUND	= x + sum	16 17 16	6	T	Lookahead causes STC(35)=51 → i; 17 → j; input(17)=); S(),x)=F, so STC(51)=91 → i; similarly with plus, so STC(91)=243 → i; similarly for sum, only STC(243)=0, so 16 → j and error exit to NOPATH.
25							↓	5		SW_{25}=F → STC(26)=0; error exit to RETRACE.
24							15			error exit to NOPATH.
23							↓			error exit to NOGO.
22	92 107					symbol	14 15	5	F	STC(92)=107 → i; T → S_2; F → S_2; 15 → j; input(15)=C; F → SW_{22}; S( )=T; build symbol.
23	9	symbol	0_{22}	NOGO	CONT.	letter	15	5	T	11 → 0_{23}; F → SW_{23}; S( )=T; go on.
24	25	symbol	0_{22}	NOPATH	FOUND	digit	15 16	5	T F	F → S_2; 16 → j; input(16)=2; F → SW_{24}; S(2,digit)=T; go build digit.
25	21	digit	0_{24}	NOGO	CONT.	digit	16	5 6	T	23 → 0_{25}; -5 → 0_{24}; T → SW_{25}; 23 → output(5) 6 → k; normal exit to CONTINUE.
24										CONTINUE
25	26	symbol	0_{22}	RETRACE	FOUND	symbol	16	6 7	F	28 → 0_{25}; -6 → 0_{22}; T → SW_{25}; S( )=F; 28 → output (6); 7 → k; normal exit.
24								7		-5 → output(7); 8 → k; to FOUND (norm exit).
23								8		11 → output(8); 9 → k; to CONT. (norm exit).
22								9		CONTINUE
23	108	sum	0_{17}	RETRACE	FOUND	sum	16	9	F	110 → 0_{23}; -9 → 0_{17}; T → SW_{23}; S( )=T;
24	191	sum	0_{17}	NOPATH	FOUND	+	16 17	9	T F	F → S_2; 17 → j; input(17)=) ; F → SW_{24}; S( ),+)=F; STC(191)=0; error exit.
23							16	9		SW_{23}=T; so 110 → output(9); 10 → k; normx.
22								10		-6 → output(10); 11 → k; go to FOUND.

OPERATIONS of DIAGRAM (Contd)

L	i	GOAL	P	EE	NE	STAB(i)	j	k	S_2	REMARKS
21								11		0_{21}=garbage →output(11); 12 → k; to FOUND.
20								12		28 →output(12); 13 → k; go to FOUND.
19								13		0_{19}=-4 → output(13); 14 → k; go to FOUND.
18								14		0_{18}=3 → output(14); 15 → k; go to CONT.
17								15		CONTINUE
18	236	term	0_{16}	RETRACE	FOUND	)	16 17	15	F	17 → j; input(17)=); T →SW_{18}; S( ) ,))=F; CONTINUE
19	237	term	0_{16}	RETRACE	FOUND	term	17	15	F	subj.; 239 → 0_{19}; -15 → 0_{16}; T →SW_{19}; S(term, term)=T, so try to make larger.
20	133 185	term	0_{16}	NOPATH	FOUND	x sum	17 18 17	15	T	18 →j; input(18)=) ; STC(133)=185 →i; S(sum, term)=F; STC(185)=0; so error exit to NOPATH after resetting j to 17.
19								15 16		SW_{19}= T, so 239 → output(15); 16 → k; and normal exit to FOUND.
18								16		garbage → output(16); 17 →k: to FOUND.
17								17		0_{17}=-9 → output(17); 18 → k; to CONTINUE.
16								18		CONTINUE.
17	53	sum	0_{11}	RETRACE	FOUND	term	17	18	F	subj.; 55 → 0_{17}; F → SW_{17}; go on.
18	133 185	sum	0_{11}	NOPATH	FOUND	x sum	17 18 17	18	T	18 →j; input(18)= ; 17 →j; STC(133)= 185→ i; subj; 187 → 0_{18}; -18 → 0_{11}; T → SW_{18}; S(sum, sum)=T, so go on.
19	191	sum	0_{11}	NOPATH	FOUND	+	17	18	T	since input(18)=) and STC(191)=0, errorx
18								18		SW_{18}=T, so 187 → output(18); 19 → k; normx
17								19		0_{17}=55→ output(19); 20 → k; to FOUND.
16								20		0_{16}=-15 →output(20); 21 → k; to FOUND.
15								21		0_{15}=garb. → output(21); 22 → k; to FOUND.
14								22		0_{14}=28 → output(22); 23 → k; to FOUND.
13								23		0_{13}=-3→ output(23); 24 →k; to FOUND.
12								24		0_{12}=7 → output(24); 25 →k; to CONTINUE.

OPERATIONS of DIAGRAM (Contd)

L	i	GOAL	P	EE	NE	STAB(i)	j	k	S_2	REMARKS
11								25		CONTINUE
12	236	term	0_{10}	RETRACE	FOUND	)	17 18	25	F	18 → j; input(18)=) ; T → SW_{12}; S( )=F; CONTINUE.
13	237	term	0_{10}	RETRACE	FOUND	term	18	25	F	subj.; 239 → 0_{13}; -25 → 0_{10}; T→ SW_{13}; S( )=T, so try to make larger term.
14	133 185	term	0_{10}	NOPATH	FOUND	x sum	18 19 18	25	T F T	F → S_2; input(19)= + and S(+, x)=F, so T→ S_2; and STC(133)=185 →i; sum is subj. and S( )=F, and STC(185)=0, so error exit.
13							18	25		SW_{13}=T, so 239 → output(25); 26 → k;normx.
12								26		0_{12}=garb. → output(26); 27 → k; to FOUND.
11								27		0_{11}=-18 → output(27); 28 →k; go to CONT.
10								28		CONTINUE.
11	93	sum	0_5	RETRACE	FOUND	sum	18	28	F	subj.; 95 → 0_{11}; -28 → 0_5; T – SW_{11};S( )=T.
12	191	sum	0_5	NOPATH	FOUND	+	18	28	T	19 → j; input(19)=+; T → SW_{12}; S( )=F, CONT.
13	192	sum	0_5	RETRACE	FOUND	term	19 20	28	T F	F → S_2; 20 → j; input(20)=B; F→ SW_{13}; S(B, term)=T; go build term.
14	5	term	0_{13}	NOGO	CONT.	letter	20	28	T	subj.; 7 →0_{14}; F →SW_{14}; S( )=T; go on.
15	25	term	0_{13}	NOPATH	FOUND	digit	20 21	28	T F	F→ S_2; 21 → j; input(21)=1; F →SW_{15}; S(1, digit)=T; go build digit.
16	17	digit	0_{15}	NOGO	CONT.	digit	21	28 29	T	subj.; 19 → 0_{16}; -28 →0_{15}; S( )=F, so 19 → output(28); 29 → k; normx to CONT.
15										CONTINUE.
16	26	term	0_{13}	RETRACE	FOUND	symbol	21	29	F	subj.; 28 → 0_{16}; F →SW_{16}; go on.
17	35 51	term	0_{13}	NOPATH	FOUND	= x	21 22	29	T	Lookahead causes STC(35)=51 → i; 22 → j; input(22)=x; T →SW_{17}; S( )=F; CONT.
18	52	term	0_{13}	RETRACE	FOUND	term	22 23	29	T F	F →S_2; 23 → j; input(23)=A; F → SW_{18}; S(A, term)=T, so go guild term.
19	1	term	0_{18}	NOGO	CONT.	letter	23	29	T	subj.; 3 →0_{19}; F →SW_{19}; S( )=T; go on.
20	25	term	0_{18}	NOPATH	FOUND	digit	23 24	29	T F	F → S_2; 24 →j; input(24)=2; F→ SW_{20}; S( )=T; so go build digit.

OPERATIONS of DIAGRAM (Contd)

L	i	GOAL	P	EE	NE	STAB(i)	j	k	S_2	REMARKS
21	21	digit	0_{20}	NOGO	CONT.	digit	24	29	T	subj.; 23 → 0_{21}; -29 → 0_{20}; T → SW_{21};
								30		S( )=F; 23 → output(29); 30 → k; normx.
20										CONTINUE
21	26	term	0_{18}	RETRACE	FOUND	symbol	24	30	F	subj.; 28 → 0_{21}; F → SW_{21}; go on.
22	35	term	0_{18}	NOPATH	FOUND	=	24	30	T	Lookahead causes STC(35)=51 → i; 25 → j
	51					x	25			input(25)= . ; S(.,x)=F, so STC(51)= 91 → i;
	91					+				S(.,+)=F; so STC(91)=243 → i;
	243					sum	24			subj.; SW_{22}=F and S( )=F; so errorx.
21							24	30		SW_{21}=F; STC(26)=0; errorx to RETRACE.
20							24	30		STC(25)=0; 23 → j; 29 → k; errorx.
19							23	29		STC(1)=0; SW_{19}=F; errorx to NOGO.
18	52						22	29	T	SW_{18}=F; STC(52)=71 → i; F → SW_{18};
	71					symbol	23	29	F	S(A, symbol)=T; go build symbol.
19	1	symbol	0_{18}	NOGO	CONT.	letter	23	29	T	subj.; 3 → 0_{19}; F → SW_{19}; S( )=T; go on.
20	25	symbol	0_{18}	NOPATH	FOUND	digit	23	29	T	F → S_2; 24 → j; input(24)=2; F → SW_{20};
							24		F	S(2, digit)=T; go build digit.
21	21	digit	0_{20}	NOGO	CONT.	digit	24	29	T	subj.; 23 → 0_{21}; -29 → 0_{20}; T → SW_{21};
								30		S( )=F; 23 → output(29); 30 → k; normx.
20										CONTINUE.
21	26	symbol	0_{18}	RETRACE	FOUND	symbol	24	30	F	subj.; 28 → 0_{21}; -30 → 0_{18} T → SW_{21};
								31		S( )=F; 28 → output(30); 31 → k; normx.
20								31		-29 → output(31); 32 → k; normx to Fnd.
19								32		3 → output(32); 33 → k; normx to CONT.
18	71	term	0_{13}				24	33	F	CONTINUE.
19	72	term	0_{13}	RETRACE	FOUND	term	24	33	F	subj.; 74 → 0_{19}; -33 → 0_{13}; T → SW_{19};
										S(term, term)=T, so try to build term.
20	133	term	0_{13}	NOPATH	FOUND	x	24	33	T	Since input(25)= . we cannot form bigger
	185									term; error exit to NOPATH.
19								33		SW_{19}=T, so 74 → output(33); 34 → k.
18								34		0_{18}=-30 → output(34); 35 → k; to FOUND.

OPERATIONS of DIAGRAM (Contd)

L	i	GOAL	P	EE	NE	STAB(i)	j	k	S_2	REMARKS
17								35		garbage → output(35); 36 → k; to FOUND.
16								36		28 → output(36); 37 → k; normx to FOUND.
15								37		-28 → output(37); 38 → k; to FOUND.
14								38		7 → output(38); 39 → k; normx to CONT.
13	192	sum	0_5				24	39	F	CONTINUE.
14	193	sum	0_5	RETRACE	FOUND	sum	24	39	F	195 → 0_{14}; -39 → 0_5; T → SW_{14}; S( )=T.
15	191	sum	0_5	NOPATH	FOUND	+	24	39	T	Cannot form bigger sum; to NOPATH.
14								39		SW_{14}=T, so 195 → output(39); 40 → k.
13								40		-33 → output(40); 41 → k; to FOUND.
12								41		garbage → output(41); 42 → k; to FOUND.
11								42		95 → output(42); 43 → k; normx to FOUND.
10								43		0_{10}=-25 → output(43); 44 → k; to FOUND.
9								44		garbage → output(44); 45 → k; to FOUND.
8								45		28 → output(45); 46 → k; normx to FOUND.
7								46		-2 → output(46); 47 → k; normx to FOUND.
6								47		0_6=3 → output(47); 48 → k; TO CONTINUE.
5	36	stmnt	0_o				24	48	F	CONTINUE.
6	37	stmnt	0_o	RETRACE	FOUND	stmnt	24	48	F	subj.; 39 → 0_6; -48 → 0_o; T → SW_6; S( )= F;
5								49		39 → output(48): 49 → k; to FOUND.
4								49		0_5=-39 → output(49); 50 → k; to FOUND.
3								50		garbage → output(50); 51 → k; to FOUND.
2								51		28 → output(51); 52 → k; normx to FOUND.
1								52		-1 → output(52); 53 → k; normx to FOUND.
								53		0_1=7 → output(53); 54 → k; to SEER.
SEER								54		GO CALL TRANSYM(39, 48, T, 0).

OTP (Output from DIAGRAM, Input to TRANSYM)

1	23	28	19
2	19	29	23
3	15	30	28
4	15	31	-29
5	23	32	3
6	28	33	74
7	-5	34	-30
8	11	35	—
9	110	36	28
10	-6	37	-28
11	—	38	7
12	28	39	195
13	-4	40	-33
14	3	41	—
15	239	42	95
16	—	43	-25
17	-9	44	—
18	187	45	28
19	55	46	-2
20	-15	47	3
21	—	48	39
22	28	49	-39
23	-3	50	—
24	7	51	28
25	239	52	-1
26	—	53	7
27	-18		

Output from Transym

```
LDQ     B1
FMP     A2
STO     T
CLA     A0
FAD     C2
XCA
FMP     B0
FAD     A1
FAD     T
STO     B2
```

Several things should be noted at this point.

a) OTP (x) for x = 11, 16, 21, 26, 35, 41, 44, and 50 contain no information, since these outputs were caused by recognition of +'s, x's, and)'s . This is a special case of the following general problem. Consider the syntactical statement:

$$\alpha_4, \alpha_3, \alpha_2, \alpha_1, = \alpha, \{ G(\mathbb{P}_1, \mathbb{P}_2, \mathbb{P}_3, \mathbb{P}_4) \}$$

where G stands for the fact that the definition for α is in general a function of α_1, α_2, α_3, and α_4 . Any output for α_4 will have been determined before this statement is examined. It will always have a direct link put in OTP for it if it is necessary, nothing otherwise. So the following remarks apply only to α_3, α_2, and α_1 . Suppose the definition for α does not depend on that of α_n in any way, where α_n is either 1, 2, or 3. Then it is desirable to waste no OTP space on these definitions. This is precisely what is accomplished by setting T(i) = 1 for this particular element. If α_n is a simple character (such as +) which is simply recognized directly from the input, no entry is put in OTP by the call of DIAGRAM which does this recognizing. If α_n is not a simple character, then the parameters S_3 and S_4 are used to carry the "no output" information to the calls of DIAGRAM which construct α_n . This results in no indirect entry being put into OTP for α_n , and further no entries are put into OTP for any output which would normally result during the building of α_n . This normally results in a 25% to 30% saving in OTP space used, depending on the syntax. At present a — is put in front of a syntactical unit for which no OTP is needed. Thus if G does not depend on the definition of α_2, we write $\alpha_4, \alpha_3, -\alpha_2, \alpha_1, = \alpha, \{G(\mathbb{P}_1, \mathbb{P}_2, \mathbb{P}_3)\}$

where now $\mathbb{P}_1$ will get the definition of α_1 ,
$\mathbb{P}_2$ will get the definition of α_3 ,
and $\mathbb{P}_3$ will get the definition of α_4 .

Note that it is $\mathbb{P}_4$ which is eliminated, and that the definitions referred to by $\mathbb{P}_2$ and $\mathbb{P}_3$ are changed. A brief consideration of the construction of OTP will show why this is true.

b) We see that it is not necessary that OTP(x) for x = 7, 13, 23, 37, 46, and 52 be indirect. We could just as well replace these negative values of OTP with the direct values to which they point, and eliminate the latter, since they are no longer used. Thus OTP (7) would be changed to 23 and OTP(5) would be deleted. In general this change can be made for all indirect links which point to direct links which point to simple definitions, i.e., definitions which contain no $\mathbb{P}$'s . In our example we see that this situation occurs for "digit" in the sentence

LETTER, DIGIT, = SYMBOL, (...

The process described above is carried out by the sections of DIAGRAM which set up and test S_5 and S_6 . When it is known that a syntactical element will normally result in the chain described above, a slash is written before that element, and this in turn causes T(i) to be set equal to 3 for that element.

This device usually results in a 15% to 20% saving in OTP space. Combined with the elimination device described in part a) above, a saving of 40% to 50% has been obtained for OTP space. Since this is by far the largest table used, these changes were felt to be of great significance. Indeed, we have been able to handle input strings roughly twice as long as those translated before the new developments were added.

The changes in the syntax input which make T(i) = 1 and 3 in the appropriate places are these:

```
LETTER, /DIGIT, = SYMBOL,
SYMBOL, - =, SUM, = STMNT,
        , - x, TERM, = TERM,
        ,    , SYMBOL, = TERM,
        , - +, TERM,= SUM,
        ,    , SYMBOL,= SUM,
        , = SUM,
TERM, - x,TERM,= TERM,
      ,    ,SYMBOL,= TERM,
      , =  ,SUM,
SUM, - +,TERM,= SUM,
    ,    ,SYMBOL,= SUM,
( , SUM, -),= TERM,
```

with appropriate changes in the definitions.

This revised syntax (along with A, = LETTER, etc.) results in the following OTP list:

1	28	20	23
2	23	21	3
3	11	22	74
4	110	23	-19
5	-1	24	28
6	28	25	19
7	15	26	7
8	3	27	195
9	239	28	-22
10	-4	29	95
11	187	30	-17
12	55	31	28
13	-9	32	19
14	28	33	3
15	15	34	39
16	7	35	-27
17	239	36	28
18	-11	37	23
19	28	38	7

This results in a 28% saving even in this simple example.

c) In following the diagramming in the example, the reader should have noticed that twice DIAGRAM wasted a considerable amount of time constructing something it later could not use. It tried to make a term starting with C2 and again with A2. These false starts can be corrected by changing the order in which units are constructed or by changing the syntax. In general it can be said that, in order to produce fast scanning of the input, a great deal of care should be exercised in putting together the syntax tables. This is not generally critical, however. The syntax which corrects the false starts for our example is:

```
A, = LETTER, (A)
B, = LETTER, (B)
C, = LETTER, (C)
0, = DIGIT, (0)
1, = DIGIT, (1)
2, = DIGIT, (2)
LETTER, /DIGIT, = SYMBOL, (/P2/, /P1/,)
```

```
SYMBOL, -=, SUM, = STMNT, (/P1/,STO /P2/,/*776)
        , - x,SYMBOL,= TERM, (LDQ /P2/,/*776FMP  /P1/,/*776)
        ,   ,TERM, = TERM,(/P1/,XCA/*776FMP  /P2/,/*776)
        , - +,SYMBOL, -+,= YSUM,(FAD  /P2/,/*776FAD  /P1/,/*776)
        ,   ,        , -x,= NSUM, (FMP  /P1/,FAD  /P2/,/*776)
        ,   ,        , = SUM,(CLA  /P2/,/*776FAD  /P1/,/*776)
        ,   , TERM,= SUM(/P1/,FAD  /P2/,/*776)
        ,= SUM,(CLA  /P1/,/*776)
TERM,-x,SYMBOL,= TERM,(/P2/,XCA/*776FMP  /P1/,/*776)
     ,  ,TERM,= TERM,(/P2/,STO  T/*776/P1, T/./F1/,/,XCA/*776
                                                FMP  T/*776)
     ,= SUM,(/P1/,)
SUM,-+,SYMBOL,-+,= YSUM,(FAD  /P1/,/*776STO  T/*776,/P2,T/./F1/,/,
                         FAD  T/*776)
   ,  ,        ,-x,= NSUM,(FMP  /P1/,STO  T/*776/P2,T/./F1/,/,FAD T/*776)
   ,  ,        , = SUM,(/P2/,FAD  /P1/,/*776)
   ,  ,TERM,= SUM,(/P1/,STO  T/*776/P2,T/./F1/,/,FAD  T/*776)
(,SUM,-),= TERM,(/P1/,)
YSUM,SYMBOL,-+,= YSUM,(FAD  /P1/,/*776/P2/,)
     ,        ,-x,= NSUM,(FMP  /P1/,/*776/P2/,)
     ,        ,= SUM,(CLA  /P1/,/*776/P2/,)
     ,TERM,= SUM,(/P1/,/P2/,)
NSUM,SYMBOL,-x,= NSUM,(FMP  /P1/,/*776XCA/*776/P2/,)
     ,        ,= TERM,(LDQ  /P1/,/*776/P2/,)
     ,TERM,= TERM,(/P1/,XCA  /*776/P2/,)
```

For our sample problem this generates

```
        LDQ     A2
        FMP     B1
        STO     T
        CLA     A0
        FAD     C2
        XCA
        FMP     B0
        FAD     A1
        FAD     T
        STO     B2
```

which is almost identical to the previous result.

Appendix 3

TABLES USED BY DIAGRAM AND TRANSYM

STABS

This table contains the syntax used by DIAGRAM to diagram the input string. The IBM 7090 word is a 36-bit word divided into 4 groups; these are (starting at the left) a 3-bit prefix, a 15-bit decrement, a 3-bit tag, and a 15-bit address. Using this notation and defining the base location of STABS to be $STABS_0$, STAB [i] = address of ($STABS_0$ + i), STC [i] = decrement of ($STABS_0$ + i), and T [i] = tag of ($STABS_0$ + i). If STAB [i] is the subject of a sentence, then the word in $STABS_0$ + i + 1 will have a 1 in the first bit of the prefix (this is also the sign bit, so we could say that ($STABS_0$ + i + 1) is negative), and the decrement will contain the location in the table STABD of the definition corresponding to that sentence.

STABD

This table contains the definitions used by TRANSYM to translate the output string OTP, which is put out by DIAGRAM. This table has been squeezed down so that four 9-bit characters go in each word. However, each definition starts in a new word, the last word of the previous definition being filled in with 0's if necessary.

Thus the sentences

LETTER, LETTER, = LTRGRP, (/P2/,/P1/,)

,= LTRGRP, (/P1/,)

with LETTER represented internally as 105, LTRGRP as 653, ℙ (i.e. ''/P'') as 750,][(i.e. ''/,'') as 753, { (i.e. ''('') as 754, and } (i.e. '') '') as 755, would appear as $STABS_0$

$STABS_0$	+ α	0	α + 3	0	00105
	+ α + 1	0	00000	0	00653
	+ α + 2	4	β	0	00000
	+ α + 3	0	0000	0	00653
	+ α + 4	4	β + 2	0	00000
		.			
		.			
		.			
$STABD_0$		.			
		.			
		.			
$STABD_0$	+ β	750	002	753	750
	+ β + 1	001	753	755	000
	+ β + 2	750	001	753	755

TRANT

This table (400_8 = 256 words) can be thought of as indexing the STABS table. In the current usage both the decrement and address of the words in TRANT are used.

The decrement of ($TRANT_o + \gamma$) contains TRAN [2γ], where STAB [TRAN [2γ]] is the second element of the sentence which started with the syntax element 2γ , and the address of ($TRANT_o + \gamma$) contains TRAN [$2\gamma + 1$].

Using the previous example again, since $105_8 = 69_{10} = (2)(34) + 1$, the address portion of ($TRANT_o + 34$) would contain α .

INDXS and SUCRT

The function SUCCR [A1; A2] is used to see if it may be possible to form the syntactical element A2 from a section of the input string, having already ascertained that part (or all) of this section is syntactical element A1.

The $n^{\underline{th}}$ word of INDXS (1000_8 = 512 words) is of the form $a_{n1}\ a_{n2}\ b_n\ c_n$, where a_{n1} and a_{n2} each take 9 bits, b_n takes 13 bits, and c_n takes 5 bits. a_{n1} is the lowest syntactical element for which SUCCR [n; m] is true, and a_{n2} is the highest. b_n is the location in SUCRT of the word defining SUCCR [n; m] for $a_{n1} < m < a_{n2}$, and c_n the bit in that word for $a_{n1} + 1$. Thus if we want to see what syntactical elements could be built from element 103_8 , we look in INDXS $+ 67_{10}$, which could contain, say

777 000 0 00000 meaning 103 can lead to nothing (since it is our policy never to use 777 for a syntactical element), or

104 104 0 00000 meaning 103 can lead only to element 104, or

063 105 0 00672 meaning 103 can lead to 063 and 105, and that b_n = 000 000 000 1101_2 or $15_8 = 13_{10}$, and $c_n = 11010_2 = 32_8 = 26_{10}$, so that in words $SUCRT_o + 13_{10}$ and $SUCRT_o + 14_{10}$ we might have (starting at the $26^{\underline{th}}$ bit of $SUCRT_o + 13$ with the sign bit considered as bit 0)

$SUCRT_o + 13$ —————————— 0 0 1 1 0 0 1 1 1 0

$SUCRT_o + 14$ 1 1 0 1 1 1 1 ——————————

meaning that 103 can lead also to 066, 067, 072, 073, 074, 076, 077, 101, 102, 103, and 104, but not to 064, 065, 070, 071, 075, or 100.

INPUT STRING and the table INSYM

Each input character is allowed 9 bits, and four characters are put in each word of the input space. Each character on a card is read in its BCD representation (A is 21_8, $ is 53_8, * is 54_8, / is 61_8, + is 20_8 , and a blank is 60_8), and a 0 is added to the left (A is 021, $ is 053, etc.) for all characters except a '. Following a ' the characters up to and including the next ' are read in,

saving the first six. Then these six characters (or less if there were less than six between the ' ,s) are compared with the symbols in the table INSYM. This table consists of pairs of words, the first containing a symbol, the second a 9-bit representation for this symbol. If the input symbol matches a symbol in INSYM, then the corresponding 9 bits are put into the input string, otherwise an error stop occurs.

Thus, if the symbols IF and GO TO are assigned to 304 and 457 respectively, the card 'IF' (A = B1*C) 'GO TO' 357 is put into the input string (assuming this is the first card) as

000 304 074 021 | 013 022 001 054 | 457 003 005 007 | 060 060 ...

Note that 000 is put in for INPUT (0), since INPUT (1) is the first input looked at by DIAGRAM.

OTP

This table contains the output from DIAGRAM, which is also the input to TRANSYM. Output (i) occupies the left half of word $OTP_0 + {}^{i}/2$ if i is even, and the right half of word $OTP_0 + \frac{i-1}{2}$ if i is odd. The table is 4096 words long, which allows for 8191 outputs from DIAGRAM.

SUBL and ISUBIX

ISUBIX is the substitution table used by TRANSYM, and SUBL is the indexing table to ISUBIX. When TRANSYM is to output a simple character, the SUBL table is scanned (from latest entry back to first entry) to see if it contains that character. If not, that character is put out as is; if it is, then a link is obtained which gives the location in ISUBIX of the string to be substituted for that character. The physical structure of SUBL is as follows: the character is in the address portion of a word, with the link in the decrement. The substitution strings are four 9-bit characters per word with each new string starting in a new word. For example, suppose that TRANSYM is looking at the character 172, and in SUBL we have the word

000 107 000 172

And in ISUBIX + 107_8 we had (say)

	C	L	A	Δ
ISUBIX + 107_8	023	043	021	060
	S			
+ 110_8	062	777	000	000

Then the output would be CLA S .
The character 777 signals TRANSYM that that is the end of the substitution string.

OTPX String

This area, which is constructed and addressed exactly the same as the INPUT string, serves as a temporary storage space used when TRANSYM is constructing a substitution string, which will later be stored in ISUBIX, or a parameter string to be used by one of the functions $\mathcal{F}_i$.

Appendix 4

AVAILABLE FUNCTIONS OF TRANSYM

The functions $\mathcal{F}_n$ have been specified only for n = 1, 2, and 5, which have been patterned after similar functions used by Irons. All other values of n are at the user's disposal.

$\mathcal{F}_1$ This is a function of either one or no variables. It generates a new symbol everytime it is called. The symbols are of the form z1000, z10001,..., z9999. The first time it is used in a definition it must have a decimal number as an argument, this being the total number of times $\mathcal{F}_1$ occurs in the definition. The subroutine takes the current value of α, a constant at its disposal which is initially set at 1000, and stores this in α_o, a space in the TRANSYM push down space reserved for this. Then α is increased by the value of the argument of $\mathcal{F}_1$. Then the symbol $z\alpha_o$ is output and α_o is increased by one. Thereafter calls of $\mathcal{F}_1$ (which now must have no argument) only perform the last part of the above operation, that is outputting $z\alpha_o$ and increasing α_o by one.

$\mathcal{F}_2$ This is a function of one argument, a decimal number. Call this number β, and set $\gamma = \alpha_o - \beta$. Then $\mathcal{F}_2$ outputs $z\gamma$ and exits. Thus we see that $\mathcal{F}_2$; 1]] outputs the same symbol as $\mathcal{F}_1$ output during its last call on the same level of TRANSYM; $\mathcal{F}_2$; 2]] outputs the same symbol as $\mathcal{F}_1$ did the time before last, etc.

$\mathcal{F}_5$ $\mathcal{F}_5$ has two arguments, the first being a string of characters. This string may contain any number of characters. $\mathcal{F}_5$ simply counts how many times the character I (031) occurs, adds this to the second argument, and outputs this as a decimal digit string.

$\mathcal{F}_5$ ' A I * / 732 O I I ' 9]] thus has 12 as its output.

In all the above cases the output is put out under control of the call of TRANSYM which called it. Thus if PUTIKER is T, the output converter; if F, it goes to OTPX.

AF Cambridge Research Laboratories, Bedford, Mass. Electronics Research Directorate
A PHRASE-STRUCTURE LANGUAGE TRANSLATOR by A.L. Bastian, Jr. 43 pp. incl. illus. tables. August 1962. AFCRL-62-549 Unclassified report

The author discusses a syntax-directed translator with several new features which result in much faster running time and the ability to handle longer input strings. The computer program for the translator is described in detail, and an example of its operation is provided.

UNCLASSIFIED

Computers
Coding
Machine translation
Programming

I. Bastian, A.L., Jr.

UNCLASSIFIED

AF Cambridge Research Laboratories, Bedford, Mass. Electronics Research Directorate
A PHRASE-STRUCTURE LANGUAGE TRANSLATOR by A.L. Bastian, Jr. 43 pp. incl. illus. tables. August 1962. AFCRL-62-549 Unclassified report

The author discusses a syntax-directed translator with several new features which result in much faster running time and the ability to handle longer input strings. The computer program for the translator is described in detail, and an example of its operation is provided.

UNCLASSIFIED

Computers
Coding
Machine translation
Programming

I. Bastian, A.L., Jr.

UNCLASSIFIED

AF Cambridge Research Laboratories, Bedford, Mass. Electronics Research Directorate
A PHRASE-STRUCTURE LANGUAGE TRANSLATOR by A.L. Bastian, Jr. 43 pp. incl. illus. tables. August 1962. AFCRL-62-549 Unclassified report

The author discusses a syntax-directed translator with several new features which result in much faster running time and the ability to handle longer input strings. The computer program for the translator is described in detail, and an example of its operation is provided.

UNCLASSIFIED

Computers
Coding
Machine translation
Programming

I. Bastian, A.L., Jr.

UNCLASSIFIED

UNCLASSIFIED

UNCLASSIFIED

UNCLASSIFIED

UNCLASSIFIED

UNCLASSIFIED

UNCLASSIFIED

UNCLASSIFIED

UNCLASSIFIED