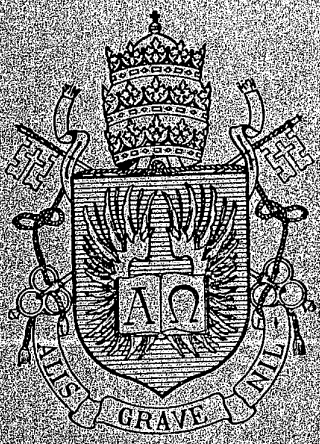
1999



PUC

Series: Monographs in Computer Science
and Computer Applications

Nº 1/69

CONVERSION FROM BNF TO SYNTAX-GRAPH, AND SYNTAX-GRAPH REDUCTIONS

by

RAPHAEL CHRYSOSTOMO BARBOSA DA SILVA

Computer Science Department - Rio Datacenter

CENTRO TECNICO CIENTIFICO
Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 209 — ZC 20
Rio de Janeiro — Brasil

CONVERSION FROM BNF TO SYNTAX-GRAPH, AND SYNTAX-GRAPH REDUCTIONS

RAPHAEL CHRYSOSTOMO BARBOSA DA SILVA
COMPUTER SCIENCE DEPARTMENT
PUC - RIO DE JANEIRO

INDEX

(A) THE PROBLEM

In Syntax Directed Compiling, the syntax of the source language is described in the form of some data structure upon which the Parser or Analyzer will work.
This paper will present a program that will convert BNF expressions into a data structure called the SYNTAX GRAPH.
After the conversion to this list-form structure, some reductions are made, which will contribute towards the reduction of parsing time and storage space.
This Syntax-Graph structure and Parsing mechanism is described in Prof. D.J. Cohen & C.C. Gotlieb, paper 1 .

(B) THE SYNTAX-GRAPH STRUCTURE

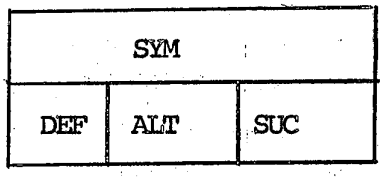
This graph is built with nodes that contain, each, the following fields:

1. SYM (or value of the node) - That will contain the BNF symbol wether it is terminal or not.
2. DEF- A pointer to the node corresponding to the definition of the type contained in SYM. If it is a terminal type, the DEF field will contain zero.
3. ALT - A pointer to the alternative definition or construction.

This field may also assume the value:
OK (zero) (or SIGMA) - when there will be no failing consequences if the type is not encountered,and there are no alternatives, beyond this one.
FAIL - (-1 or FI) If, when the type is not encountered and there is no other alternative and this type could not be missing in order to exist the instance ot the subject it defines.

4. SUC - a pointer to the next component in the alternative. When the node is an end of alternative node, this field will also assume the value OK (or SIGMA).

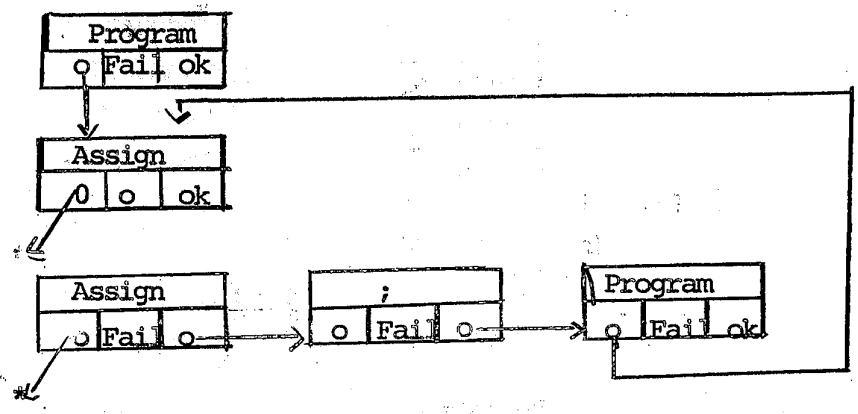
(C) THE NODE & EXAMPLE OF A GRAPH



A graph for the BNF expression:

Program > ::= <Assign> | Assign ; <Program>

would be: (without any reduction)



* Pointers to the definition of Assign that would be given in other (subsequent) BNF statements.

(D) THE BNF INPUT

In order to write a program that will read the BNF specification of the syntax of a language, it is necessary to define precisely what the program will expect to read.

For this description, I will use the BNF notation (and some comments), where the terminal symbols (or terminal types) will be enclosed in " " .

< Syntax Specification > :: = < Syntax Card> < BNF Cards > <End Card>

< Syntax Card > :: = < any comment > ".Syntax." < any comment >

< END Card > :: = < any comment > ".End Syntax." < any comment >

< BNF Cards > :: = < BNF Card > | < BNF Card > < BNF Cards >

< BNF Card > :: = <Defined Type > ":: =" <Definition >

< Definition > :: = < Alternative > | < Alternative > "|" <Definition >

< Alternative > :: = <Component > | < Component > < Alternative >

< Component > :: = < Defined Type > | < Terminal Type>

< Defined Type > :: = "<" < Letters > ">"

< Terminal Type > :: = <Any Char >

< Letters > :: = L

< Any Char > :: = A

< Any Comment > :: = C

L - a string of up to 6 letters

A - a string of up to 6 characters that may be (each) any letter, any digit or one of the following:

; = + - * / ' ( )

C - Any string of symbols, including a null string (Note: the keywords are located by means of the "INDEX" built in function in PL1).

(E) THE SYNTAX GRAPH REDUCTIONS

According to Pr. Cohen's paper [1], there are four types of syntax graph reductions, which are:

1. Factoring Reduction

   Alternatives with same first components will have only one of them used.

   Alternatives with same last components will use also the same node, provided that this node has no alternative link. (other than Fail).

   This reduction is of basic importance for a correct parsing besides the space-saving importance.

   In the program, it is done at the same time the Graph is being built.

2. Terminal Reduction

   Defined Types which are defined only by terminal types (all alternatives contain only terminal types), can be eliminated and replaced by their definition.

   This reduction will speed the parsing but will not always reduce the amount of space used.

   In the program it was skipped, that is, it was not made, because of the time factor, (that is I had to leave soon, and this was avoided only to save time for the other more important reductions).

3. Recursive Reduction

   Recursive Alternatives, and mainly the left-recursive Alternatives (the ones where the defined type appears as the leftmost component), will cause an infinite loop of the Parser if they are not eliminated.

   So, this reduction will eliminate all components which are equal to the defined type, if they are left or right justified in the alternatives.

(middle-recursive alternatives will not be reduced).
Different treatment will be given to left and right recursive components.
As one can see, this is a very basic reduction also.
It not only saves space, but is also fundamental for the performance of the Parser.

4. Redundancy Reduction
After all this work on the Graph, some nodes will be exactly the same, that is, they will have the same symbol and the same DEF, ALT and SUC links.
The redundancy Reduction will, then, eliminate all the copies and keep only one, making the appropriate change in the pointers.
This change of pointers means searching for all nodes that somehow point to the eliminated "copy" and make it point to the "original".

(F) THE PROGRAM OUTLINE

The Program

The program was structured according to the following scheme:

1. A main procedure (Syntax2), that contains, besides the common area description and entry point references, the main line of the building of the graph and reductions in the form of calls to the appropriate routines.
2. The main routines
   a) Graph
      That makes the conversion from BNF to syntax-Graph with factoring Reduction at the same time.
      Graph uses in different levels the following subroutines and functions:

      READ - a function that returns a jump value to graph.

      GLOT - the routine in charge of advancing the input string pointer to skip blanks, read new cards and detect the .Syntax , Endsyntax. cards and the end-of-file condition.

      CNS - the function that will construct a new node.

      DELETE - that will return the node pointed at by the parameter, to the AVAIL list.

      COPY - the function that will return a pointer to the copy of the node pointed at by the PARAMETER.

RECRSIV - a logical function that will return the value true if the component given as argument is equal to the defined type.

LETTER,SPECIAL - a logical function that will return the value true if the component or rather, the character, given as an argument is a letter or a special character (when special means letter or special) depending on the word used at invokation.

b) Putdefn

After the whole syntax Graph is built and factored , this recursive procedure, by iterative calls from the main procedure (Syntax2) will traverse each of the binary trees whose roots are in the symbol-table, and add the DEF link.
To find where the definition of the component is, it uses the function SEARCH that will return the symbol table pointer.

c) Recred

This is the subroutine that will do or control the recursive reduction process.
It scans the symbol-table and for each line uses the TRAVRSE subroutine that will use the REPLACE routine.
TRAVRSE - traverses the "binary trees" whose roots are argument, and, for each node, calls the REPLACE routine.
REPLACE - checks if the node is a recursive component (equal to the symbol in the symbol table in the same line as the root given as argument).
If it is not it returns.
If it is, it will take the appropriate action for elimination of the node, if it is a left or rignt component, or just

ignoring it if is a middle-component.

Two important precautions must be pointed.

1. The TRAVERSE routine traverse the trees in the inverse postfix form, in order to avoid a loop when it would get to the already reduced nodes if the traversal was done in any other form.

2. The REPLACE routine leaves the nodes free or loose,and does not return them to the AVAIL list.

This is done because, if the node is a last node, for instance, which was used as a common end for more than one alternative in the case of an end-to-end factoring reduction, it will be again returned to the avail list, what will be disastrous.

However, it will be left loose, and to provide for its later retrieval, there is the routine COLLECT that is called by the routine CNS whenever the AVAIL list is empty .

COLLECT - Searches all nodes, looking for any other nodes that may be pointing at them. If it is not pointed at by any other node, it will be added to the AVAIL list.

d) Redred

This is the Redundancy Reduction Routine

By the use of three nested do-loops and feature of "cleaning" the node name (or symbol) when it is deleted, it will return to the AVAIL list all nodes that are "copies" of another, and will change the pointers accordingly.

3. Common Areas

By this title I mean the areas which are defined (or declared) in the main procedure.

They are:

a) The NODE (100)

Containing - SYM - the symbol
DEF - the definition link
ALT - the alternative link
SUC - the successor link

Initially, all nodes are linked by the DEF link, in the avail list.

b) The STRING (82 characters)

The input area.

Notice that this "card" has 82 characters. The 81st is used to put an aditional "or" symbol (|), so that the scanning may be more standard.

The 82nd is a blanck character also for standardization (mainly for the GLOT routine).

c) The SYMTAB (100) (symbol table)

Containing - SM - the symbol field
PT - the pointer to the first component of the first alternative of the defined type whose symbol appears in the SM field.

Only defined types are put in this symbol-table.

It can also be looked as an array of roots.

d) SYMBOL

An intermediate buffer where the symbol from the input string is put , before the node is built.

e) FIRST (100)

Containing FP - the "first-pointer".

These pointers point to the first component of the current alternative (the one that is being handled), and, if factoring is being done, successive "first pointers"will point to successive factored components.

They are reset at each new alternative that begins different form the previous one.

f) LAST (100)

Containing LP - a "last pointer"

USEDALT - a logical variable that will indicate wether or not the node pointed at by LP has an alternative link or not.

USEDSUC - a logical variable that will be set on or true when the node pointed at LP has been used already for a last-to-last factoring reduction.

This vector contains all the last components of all definitions.

(G) GRAPH ROUTINE

1. The Method

/* Conversion from BNF to Syntax Graph with factoring */ Graph relies upon the information contained in the value returned by the READ routine. This value is used to jump to the appropriate section.

These sections are:

1. Defined type
   Action: the symbol is put in the symbol-table.
2. First Component of First Alternative
   Action: the component is linked to the symbol table, and put in the FIRST list.
3. First Component of other alternatives
   Action: Disables the recursive chek; Adds the component to the FIRST list.
   Links to previous first's ALT field, or if eugal, deletes the new node and sets the factoring flag (FACT) on.
4. Not First Component and Factoring
   Action: Since the FACT flag is also in the "common area" (in the main procedure), the read routine will check it and set the return value 4.
   The new component is added to the SUC link of the last (or current) FIRST, or if the SUC link is already occupied it will be added to the last ALT of the last first's successor. If however, the new node is equal to this last ALT, it will be deleted and the FACT flag will be on again.
   If the existent successor is a last component, it is checked for the USEDSUC and USEDALT flags and appropriate action taken.

5. Last Component

The last to last factoring is attempted.

If there is an equal component in the LAST vector, it is tagged with the USEDSUC flag and the current node will point to the one encountered in the LAST list, while the new node will point to the new (suc link) and the new node will be added to the LAST vector.

6. Intermediate Component

The current node will point to the new node by its successor link, SUC.

2. Graph Building Restrictions

Since this method reads and builds the graph using the alternatives in the order in which they are written, this order becomes very important.

1. The alternatives with the same starting component must be together (without any other intervening alternative) and

2. The recursive alternatives must be the last ones.

If a recursive alternative is written as the first one, the program will point out the fact and end execution.

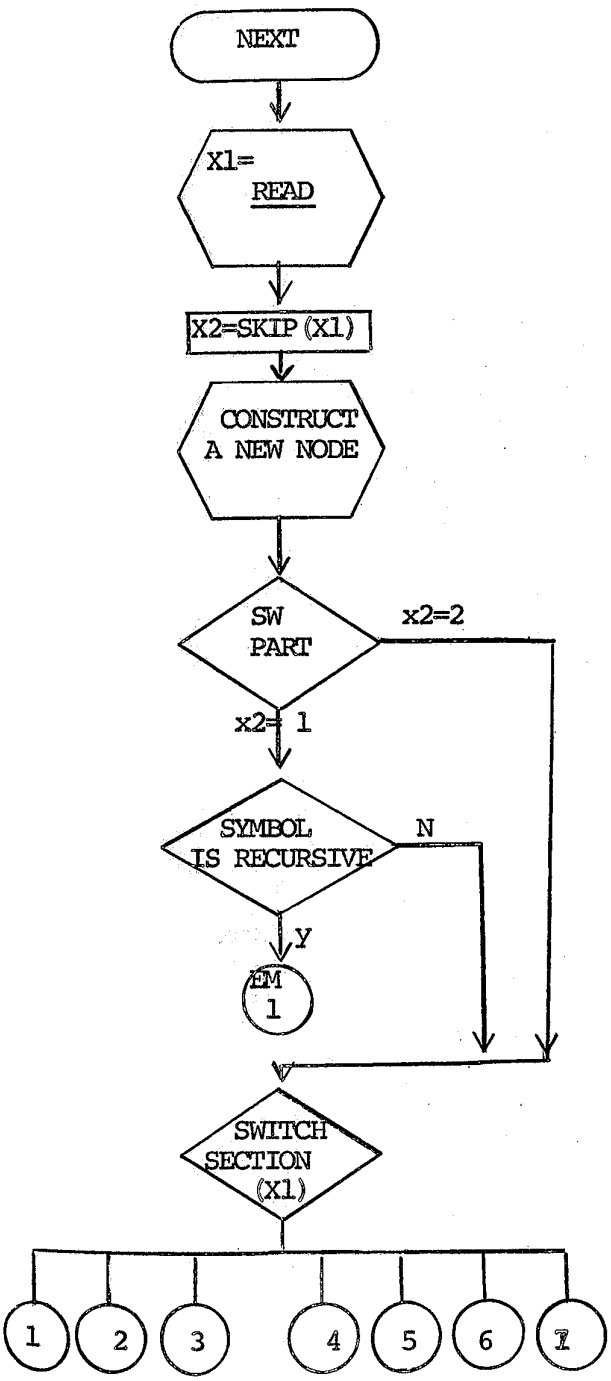
With regard to the first restriction, no detection is made.

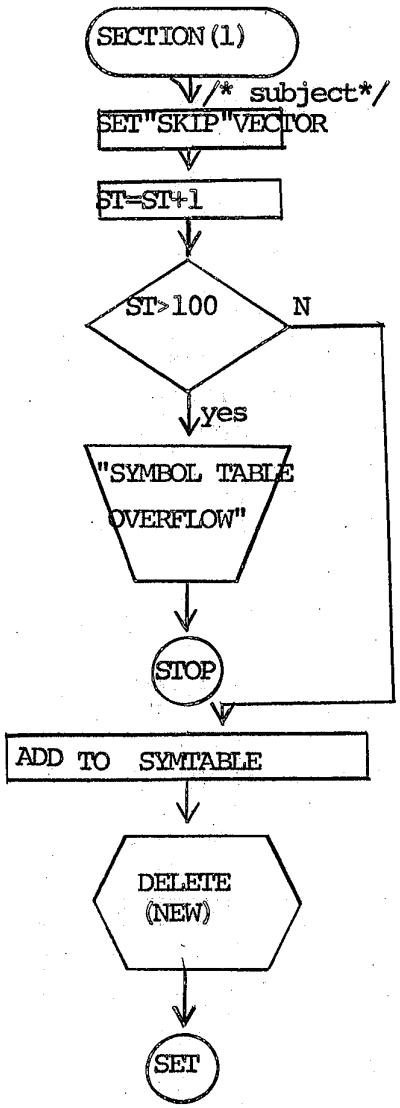
I must point out, however, that I have already designed a way of avoiding these restrictions (although they are not programmed due to lack of time):

There would be another step in the read-in process (the first one) that, for each BNF statement card, would create an array of alternatives (or pointers to the alternatives). The non-recursive alternatives would be sorted (and the sort key would be the whole alternative) to ensure the most effective factoring, and the same would be done with the recursive ones.

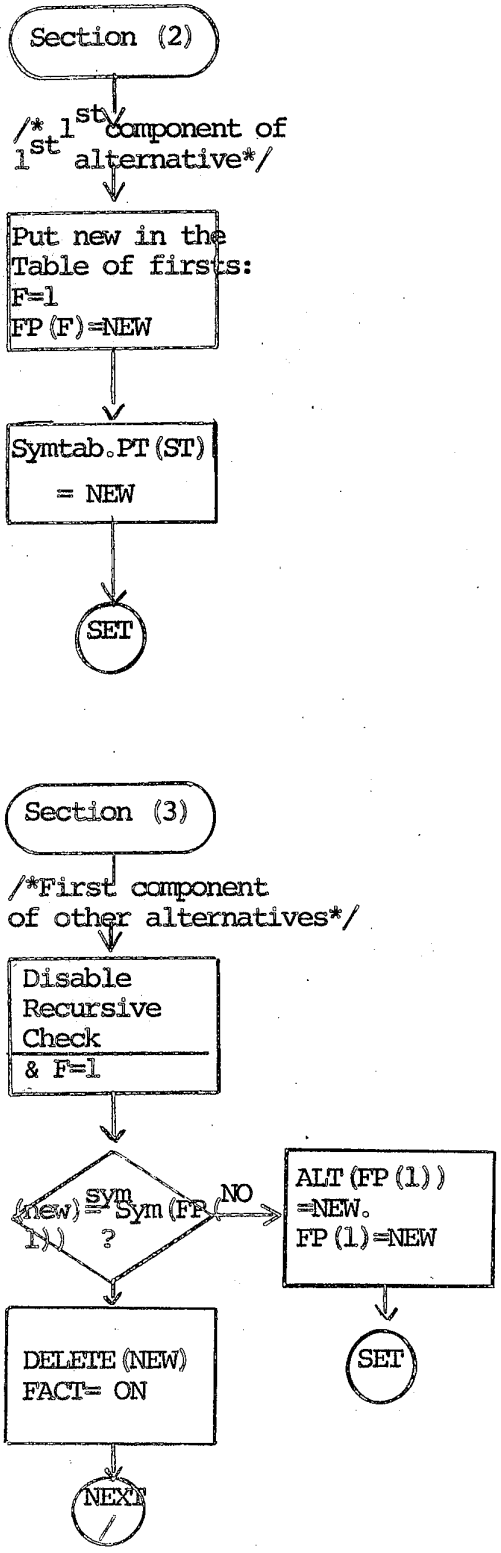
on the other hand, for the same purpose.
The read procedure would then follow the new sequence of alternatives that would be given to it by the "PREPARATION ROUTINE".
The only check (recursive) needed would be the verification if the alternative has only recursive components, in which case it would be eliminated.
This would eliminate the two restrictions listed above.

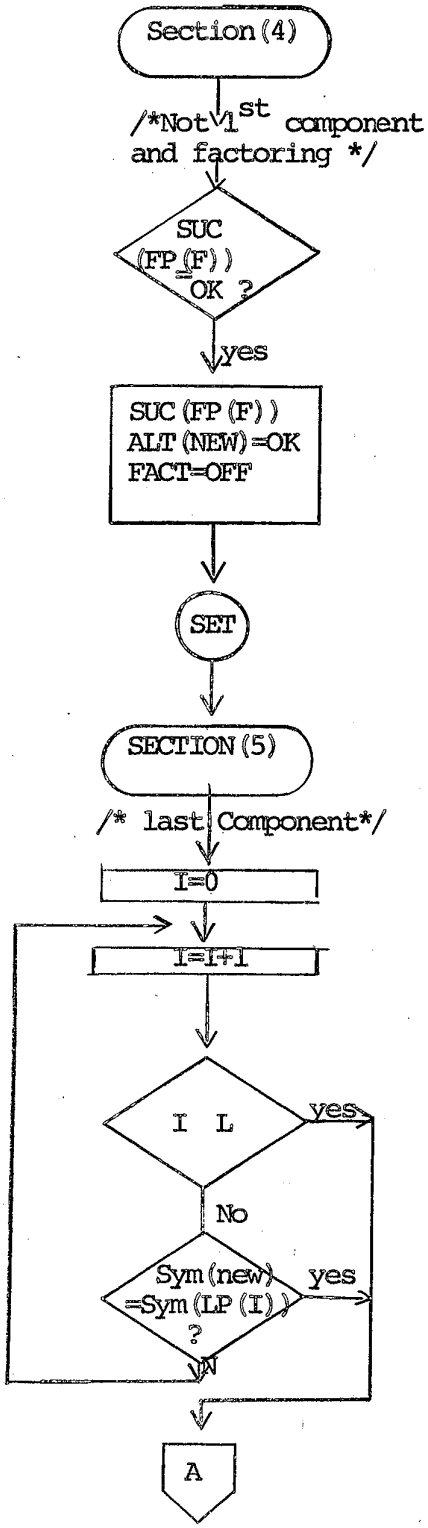
FLOW-CHART


NEXT
X1=
READ
X2=SKIP(X1)
CONSTRUCT
A NEW NODE
SW
PART
x2=2
x2= 1
SYMBOL
IS RECURSIVE
N
Y
EM
1
SWITCH
SECTION
(X1)
1
2
3
4
5
6
7

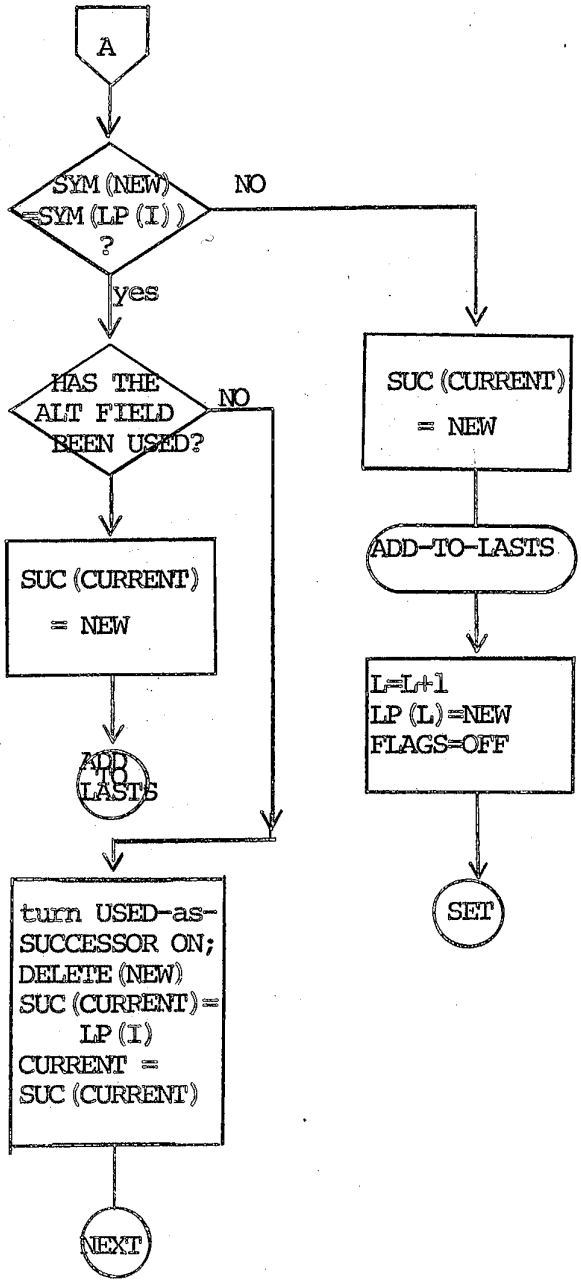

SECTION(1)
/* subject*/
SET"SKIP"VECTOR
ST=ST+1
ST>100
N
yes
"SYMBOL TABLE
OVERFLOW"
STOP
ADD TO SYMTABLE
DELETE
(NEW)
SET

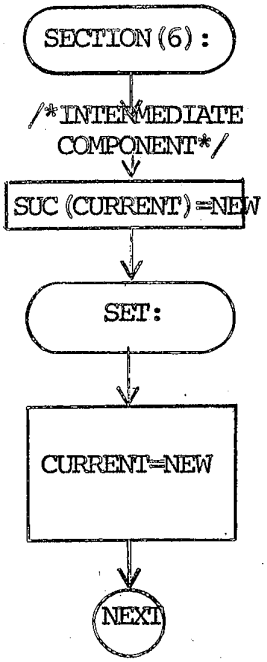
FLOW-CHART (cont'd)

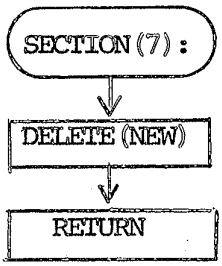


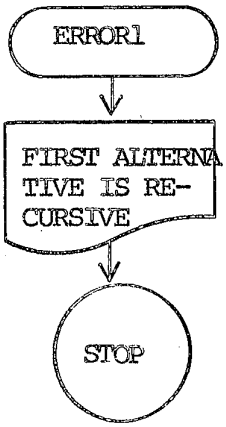


FLOW-CHART (cont'd)









(H) THE"PUT DEFINITION"PROCEDURE

1. The Method

After the Graph phase, we have a Symbol table filled with the defined types and with the pointers to the binary trees (if the DEF link is considered as another data field) corresponding to the definition of each of these defined types.
The main routine (Syntax2) will then iteratively call PUTDEFN, giving each time a pointer to one of the binary trees as a parameter.
PUTDEFN, a recursive procedure, will traverse these binary trees in postfix order, and, for each node "visited", will call the function SEARCH and will fill the DEF link with the value returned by SEARCH.
SEARCH - will return the pointer to the tree that corresponds to the definition of the defined type. If the type was not found in the symbol table, SEARCH will return the value zero. Thus, terminal types will have zero in their DEF links.

(I) RECRED - THE RECURSIVE REDUCTION

1. The Method

When RECRED is called (or invoked), three levels of subroutines will be working:

a) RECRED - the outmost procedure, will also use the forest of binary trees in the sense that it will call iteratively another routine to visit the nodes of each of these trees, by means of a do-loop.

For each line in the symbol-table (that is filled with useful information), RECRED will invoke TRAVRSE.

b) TRAVRSE - is the second level recursive procedure that will be in charge of visiting the nodes; however when the time comes to do something usefull with the node, TRAVRSE calls the REPLACE routine to do it.
It is very important to note that TRAVRSE has to traverse these trees in the inverse postfix order (right link, left link, root) because, during the traversal, the tree will (or may) become a graph of a list-type structure.
Any other type of traversal would allow an infinite loop to happen if a left recursive alternative were present in one of the alternatives.

c) REPLACE - this (non-recursive) routine contains the mechanism for replacing or, rather, eliminating a recursive component and changing the pointers accordingly. It identifies if the component is recursive or not, and, if recursive, wether it is a left, right, or middle recursive component.

(J) REDRED - THE REDUNDANCY REDUCTION

1. The Method

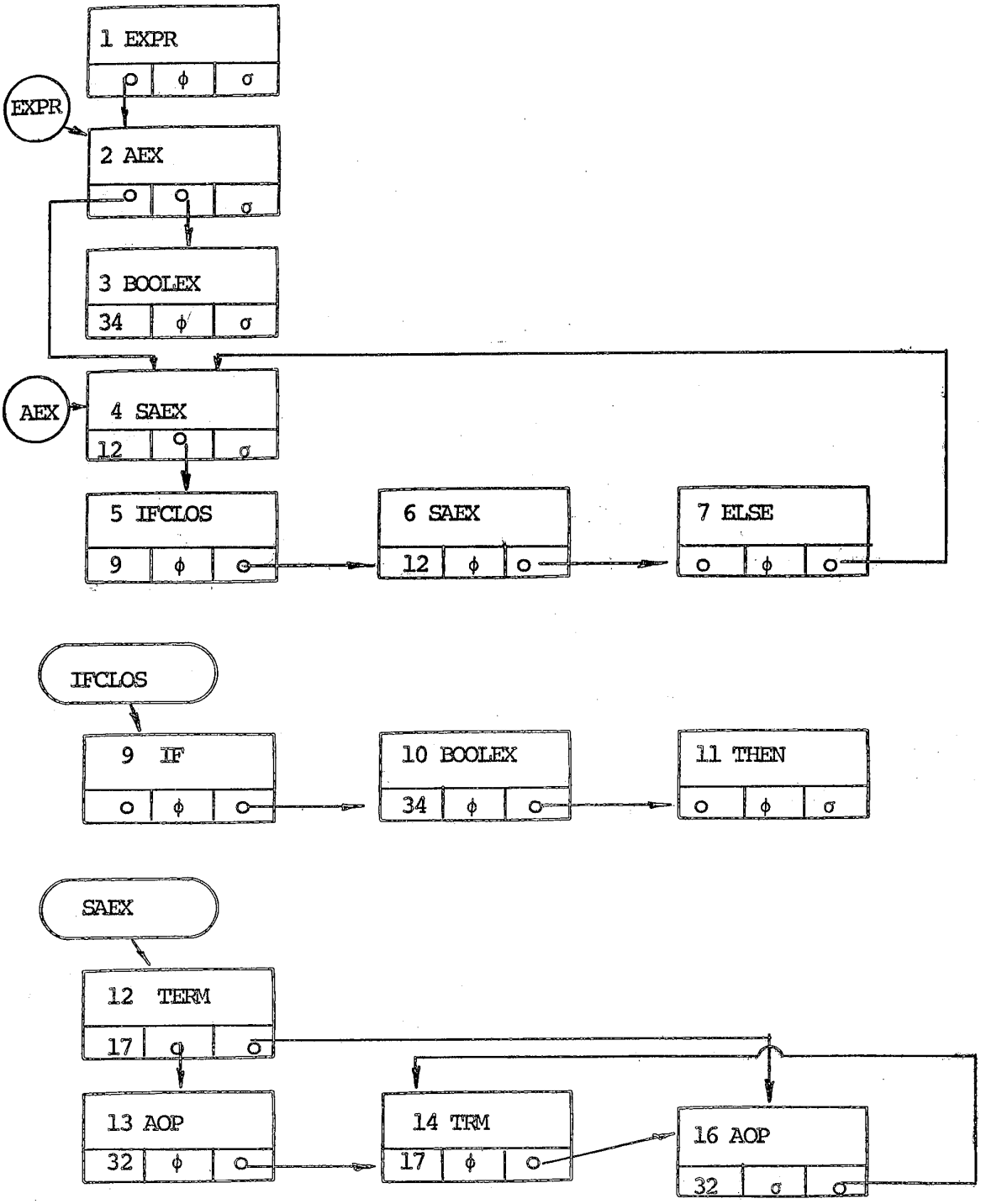
Using the fact that the nodes of the now built, factored and recursively reduced graph are in order in storage (in the initially AVAIL vector), this routine consists of three nested loops.
The outermost loop will control the processing of each and every node, and will skip blank-symbol nodes.
The second level loop will pick up the nodes to be compared with

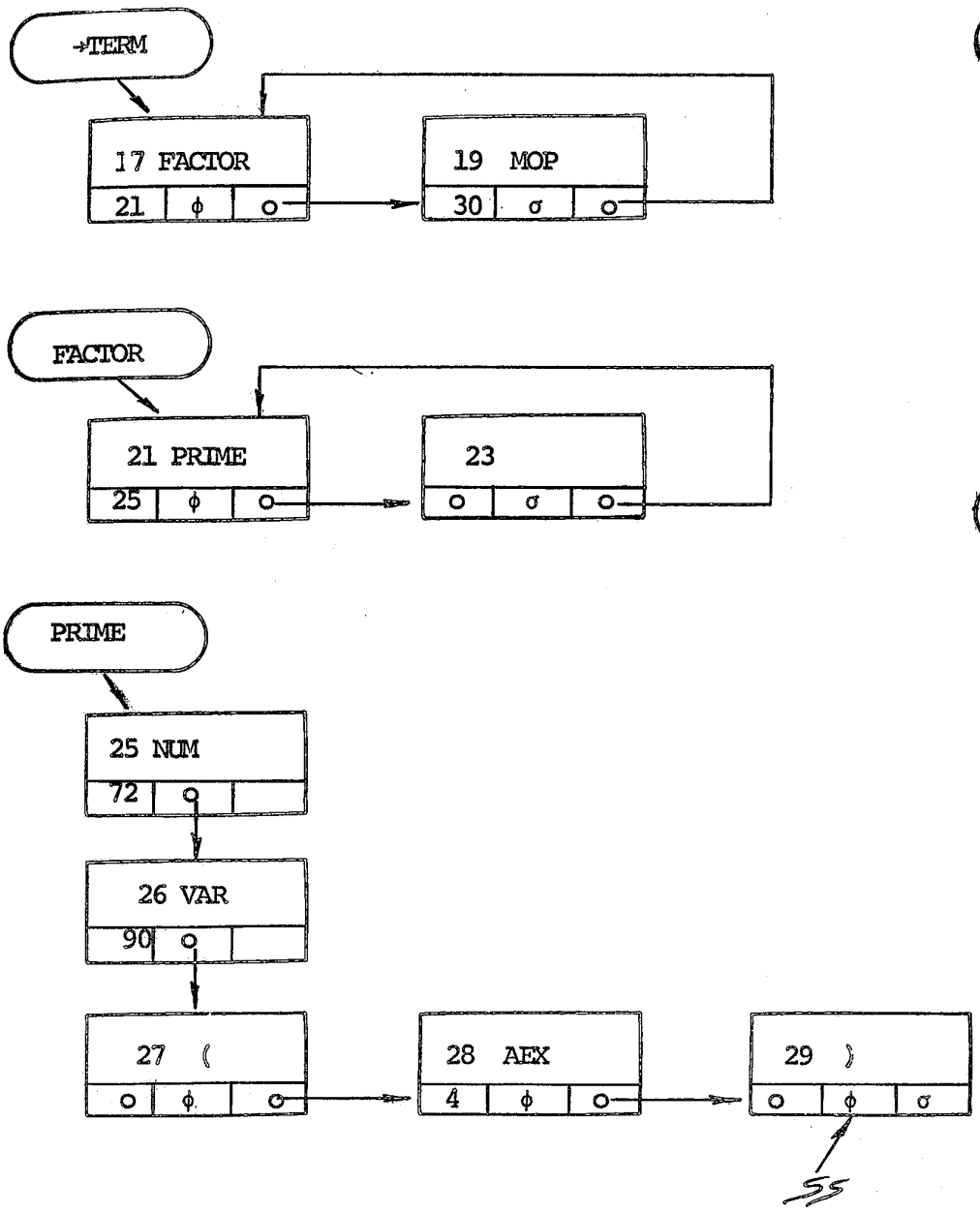
the node pointed at by the outermost loop's variable. If there is a match, the innermost loop will run over all nodes * looking for those who somehow point at the match node, and make them point at the "original" (pointed at by the outermost loop's variable).

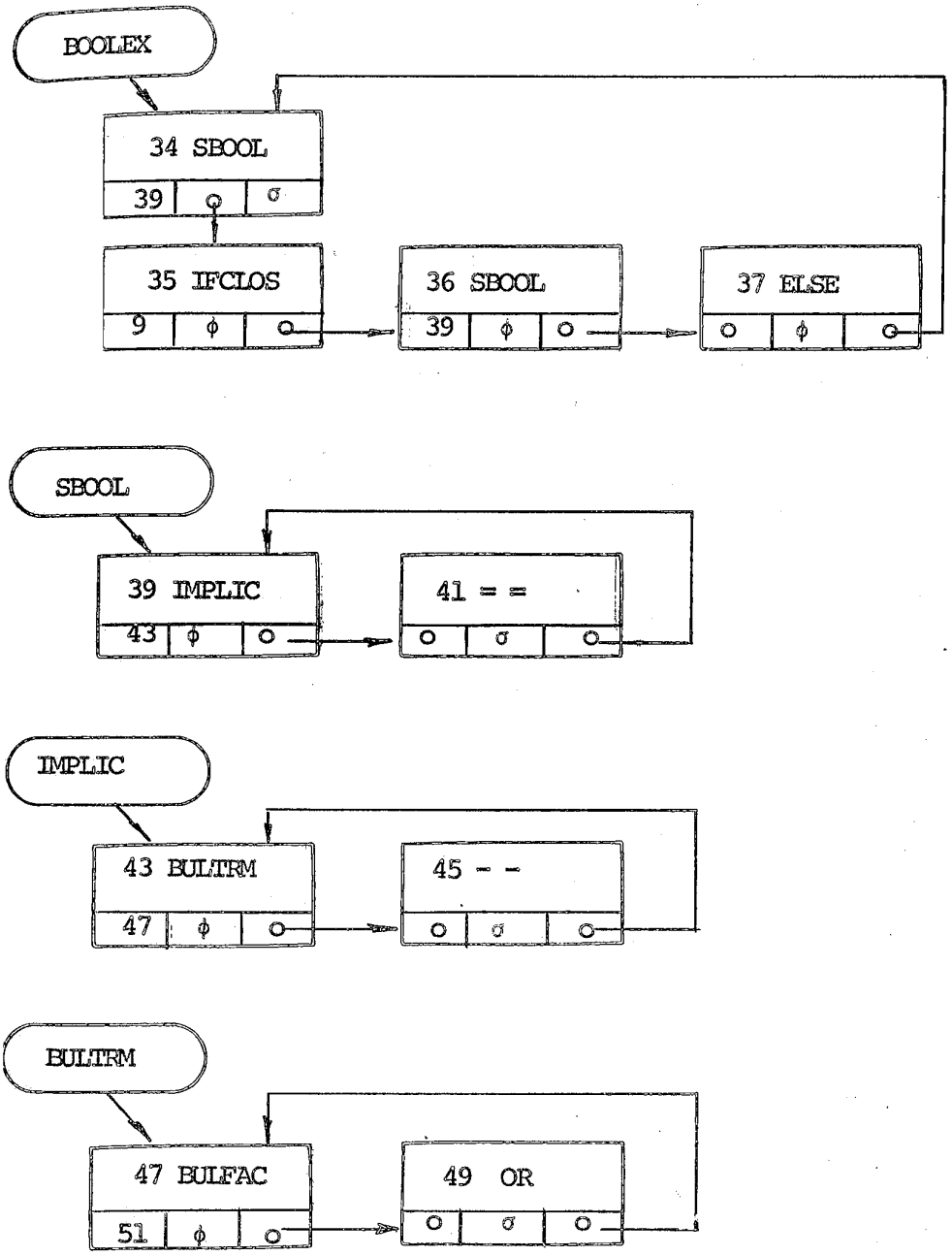
* Except the match node. The "copy" node is deleted, at the end of the whole search. This is accomplished by the use of the JFLAG which will indicate wether there was any node pointing at the current node or not.

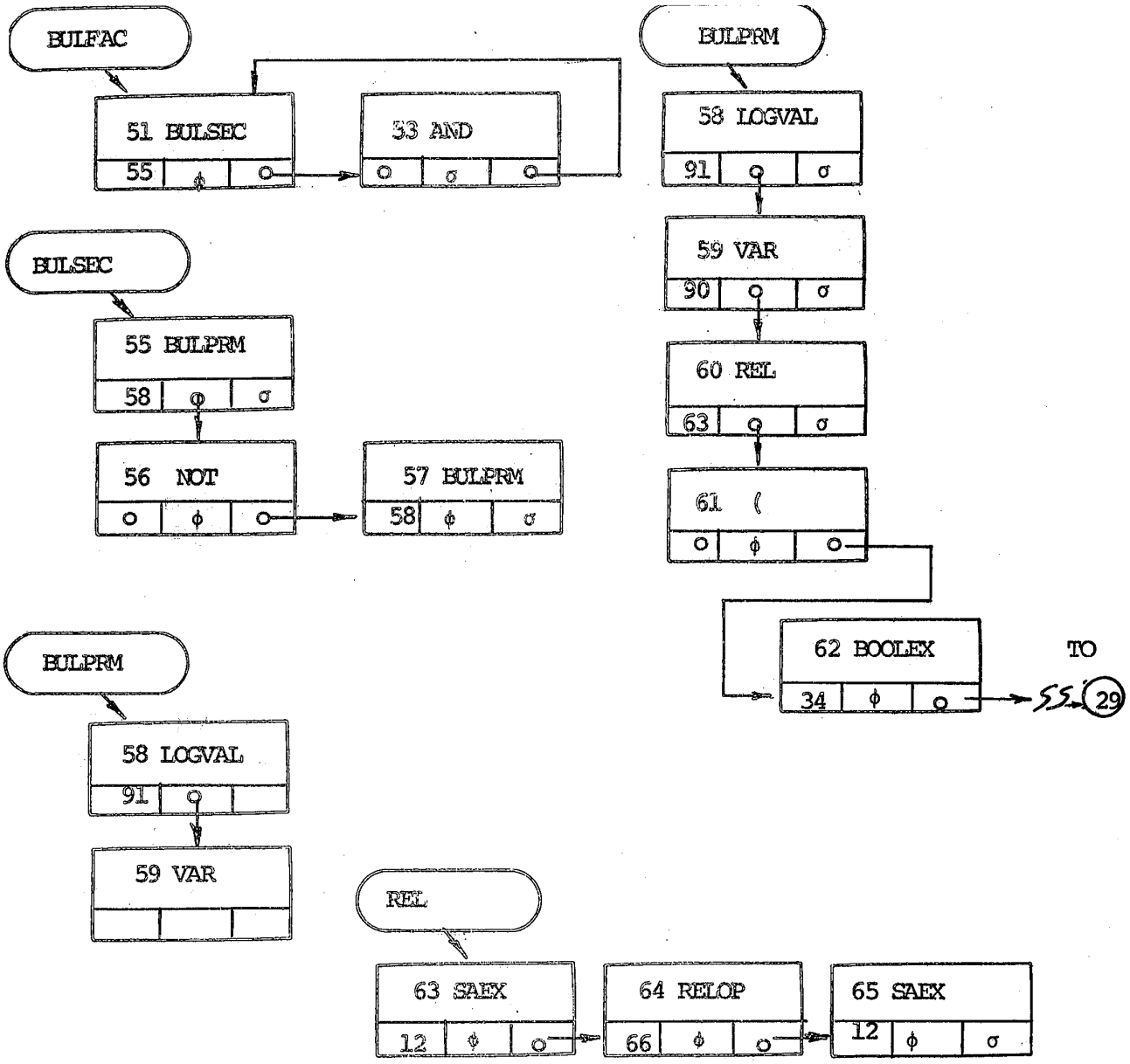
THE SYNTAX GRAPH IN GRAPHIC FORM

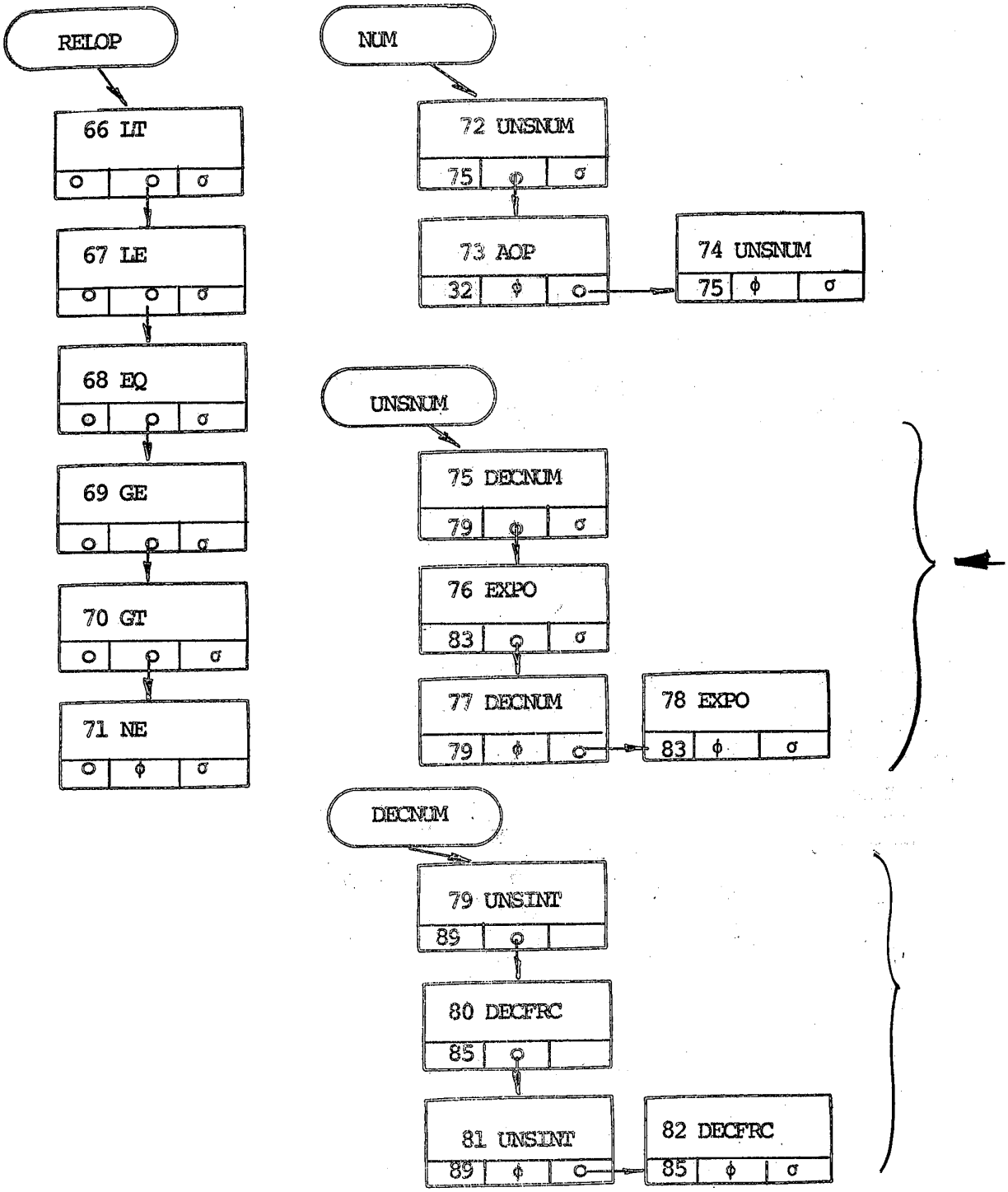
Note: This graphic has been drawn from the program output which comes next.
The table used was the last one, the GRAPH 3 table.

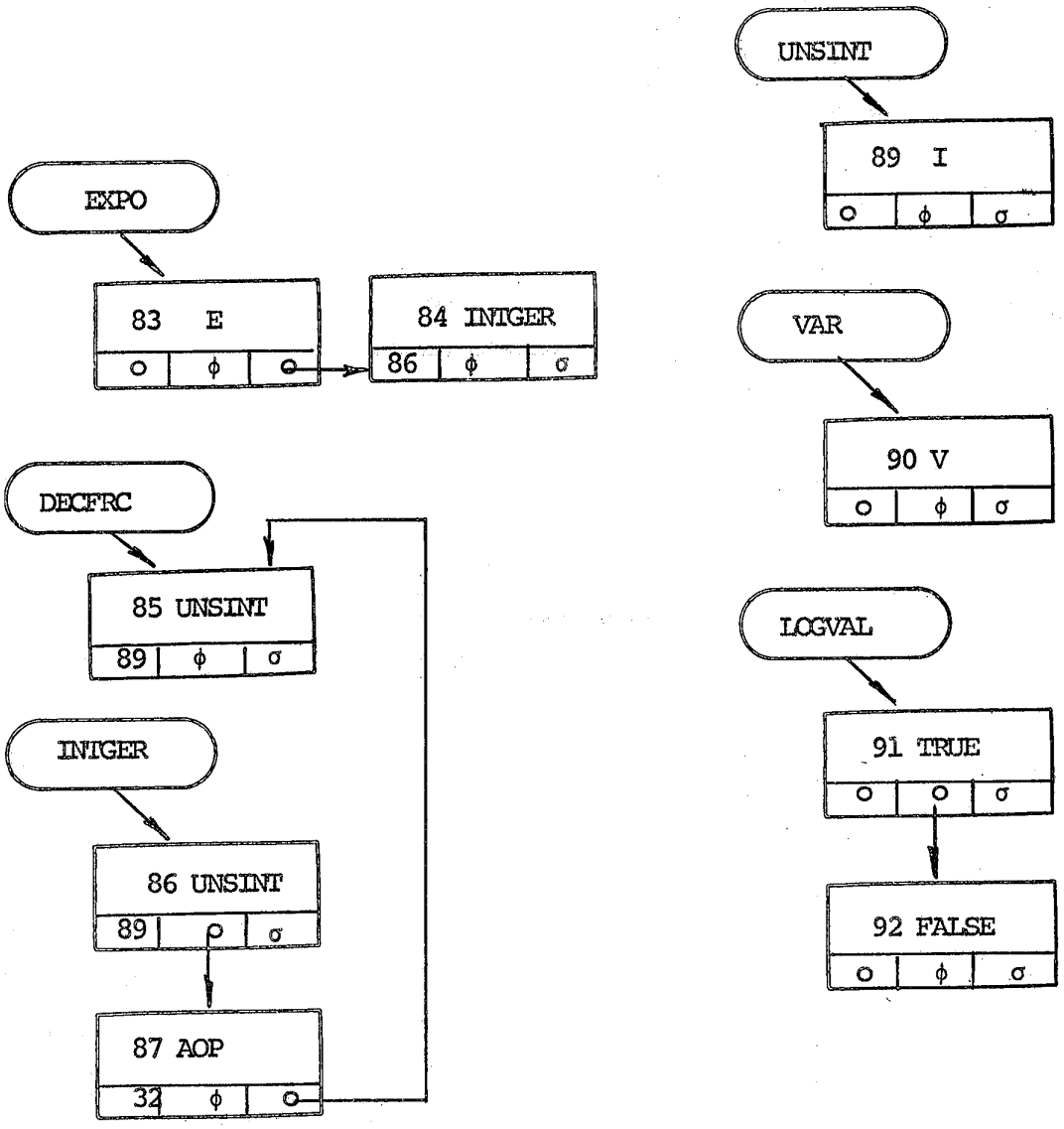

1 EXPR
ρ ϕ σ
EXPR
2 AEX
o o σ
3 BOOLEX
34 ϕ σ
AEX
4 SAEX
12 o σ
5 IFCLOS
9 ϕ o
6 SAEX
12 ϕ o
7 ELSE
o ϕ o
IFCLOS
9 IF
o ϕ o
10 BOOLEX
34 ϕ o
11 THEN
o ϕ σ
SAEX
12 TERM
17 o o
13 AOP
32 ϕ o
14 TRM
17 ϕ o
16 AOP
32 σ o


→TERM
17 FACTOR
21 ϕ o
19 MOP
30 σ o
FACTOR
21 PRIME
25 ϕ o
23
o σ o
PRIME
25 NUM
72 o
26 VAR
90 o
27 (
o ϕ o
28 AEX
4 ϕ o
29)
o ϕ σ
FROM 62
MOP
30 *
o o σ
31 /
o ϕ σ
AOP
32 +
o o σ
33 -
o ϕ σ


BOOLEX
34 SBOOL
39 o σ
35 IFCLOS
9 ϕ o
36 SBOOL
39 ϕ o
37 ELSE
o ϕ o
SBOOL
39 IMPLIC
43 ϕ o
41 = =
o σ o
IMPLIC
43 BULTRM
47 ϕ o
45 - -
o σ o
BULTRM
47 BULFAC
51 ϕ o
49 OR
o σ o


BULFAC
51 BULSEC
55 ϕ
53 AND
BULSEC
55 BULPRM
58 φ σ
56 NOT
57 BULPRM
BULPRM
58 LOGVAL
91 φ σ
59 VAR
90 φ σ
60 REL
63 φ σ
61 (
62 BOOLEX
34 ϕ
TO
55 29
BULPRM
58 LOGVAL
91 φ
59 VAR
REL
63 SAEX
12 ϕ
64 RELOP
66 ϕ
65 SAEX
12 ϕ σ


RELOP
66 LT
67 LE
68 EQ
69 GE
70 GT
71 NE
NUM
72 UNSNUM
75
73 AOP
32
74 UNSNUM
75
UNSNUM
75 DECNUM
79
76 EXPO
83
77 DECNUM
79
78 EXPO
83
DECNUM
79 UNSINT
89
80 DECFRC
85
81 UNSINT
89
82 DECFRC
85


UNSINT
89 I
o ϕ σ
EXPO
83 E
o ϕ o
84 INTGER
86 ϕ σ
VAR
90 V
o ϕ σ
DECFRC
85 UNSINT
89 ϕ σ
LOGVAL
91 TRUE
o o σ
INTGER
86 UNSINT
89 ρ σ
92 FALSE
o ϕ σ
87 AOP
32 ϕ o

BIBLIOGRAPHY

1. The Syntax Graph: A List Structure for Representing Grammars.
D.J. Cohen and C.C. Gotlieb
Univ. of Waterloo and Univ.of Toronto

2. Syntax - Directed Compiling
T.E. Cheatham, Jr. and Kirk Sattley
Proceedings of the Eastern Joint Computer Conferences
AFIPS, vol. 25 pp. 31 - 57, 1964

3. The Syntax of Programming Languages - A Survey
R.W. Floyd
IEEE Transactionson Eletronic Computers
vol. EC-13, pp. 346 - 353, Aug. 1964

4. Programming Systems and Languages
Part 3 - Compiling and Translating
Edited by Saul Rosen
Mc Graw Hill Computer Science Series, 1967

5. The Art of Computer Programming vol. 1
Donald E. Knuth - California Inst. of Technology
Addison - Wesley Publishing Company, 1968

6. PL1 Reference Manual
IBM Manual nº C28-8201-1.

```
        /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST
                /* CONVERSION FROM BNF TO SYNTAX GRAPH */
 1              (CHECK(REDRED)):
                SYNTAX2:
                  PROCEDURE OPTIONS(MAIN);
 2      1         DECLARE I BIN FIXED;
 3      1         DECLARE 1 NODE(100),
                            2 SYM CHAR(6) INIT((100)(6)' '),
                            2 (DEF,ALT,SUC) BIN FIXED
                              INITIAL((100)0,(100)0,(100)0);
 4      1         DECLARE (AVAIL,P,Q,ROOT) BIN FIXED;
 5      1         DECLARE STRING CHAR(82) INIT((82)' ');
 6      1         DECLARE CHAR BIN FIXED;
 7      1         DECLARE 1 SYMTAB(100),
                            2 SM CHAR(6) INITIAL((100)(6)' '),
                            2 PT BIN FIXED INITIAL((100)0),
                          ST BIN FIXED;
 8      1         DECLARE SYMBOL CHAR(6),
                          (OR,ON,OFF,EOF) BIT(1);
 9      1         DECLARE 1 FIRST(100),
                            2 FP BIN FIXED INITIAL((100)0),
                           F BIN FIXED;
10      1         DECLARE 1 LAST(100),
                            2 LP BIN FIXED INITIAL((100)0),
                            2 USEDALT BIT(1) INITIAL((100)'0'B),
                            2 USEDSUC BIT(1) INITIAL((100)'0'B),
                           L BIN FIXED;
11      1         DECLARE (SUBJECT,FACT) BIT(1) ;
12      1         DECLARE OK   BIN FIXED INIT(0);
13      1         DECLARE FAIL BIN FIXED INIT(-1);
14      1         DECLARE GRAPH ENTRY;
15      1         DECLARE READ RETURNS(BIN FIXED);
16      1         DECLARE CNS ENTRY(CHAR(6),BIN FIXED,BIN FIXED,BIN FIXED)
                           RETURNS(BIN FIXED);
17      1         DECLARE DELETE ENTRY(BIN FIXED);
18      1         DECLARE COPY ENTRY(BIN FIXED) RETURNS(BIN FIXED);
19      1         DECLARE RECRSIV ENTRY (CHAR(6)) RETURNS(BIT(1));
20      1         DECLARE GLOT ENTRY;
21      1          DECLARE PUTDEFN ENTRY(BIN FIXED);
22      1         DECLARE LETTER ENTRY(CHAR(1)) RETURNS(BIT(1));
23      1         DECLARE SPECIAL ENTRY(CHAR(1)) RETURNS(BIT(1));
24      1         DECLARE SEARCH ENTRY(CHAR(6)) RETURNS(BIN FIXED);

25      1        INITIAL_SETTING:
                  OR,OFF,EOF,FACT='0'B;
26      1         ON,SUBJECT='1'B;
27      1         AVAIL=2;
28      1         CHAR=82;
29      1         ST,F,L,P,Q,ROOT=0;
30      1         DO P=1 TO 99; DEF(P)=P+1; END; DEF(100)=0;
34      1         CALL GRAPH;
35      1         DO I=1 TO ST WHILE(SM(I)¬='      ');
36      1  1      CALL PUTDEFN(PT(I));
37      1  1      END;
38      1         PUT EDIT('N  SYMBOL    DEFNITION ALTERNATE SUCCESSOR ')
                        (PAGE,A);
39      1         DO P=1 TO 30;
```

```
        /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST
 40      1    1      PUT EDIT(P,SYM(P),DEF(P),ALT(P),SUC(P))
                       (SKIP(1),F(2),X(1),A(6),X(4),(3)(F(5),X(5)));
 41      1    1      END;
                     /* THE ROOT IS THE FIRST DEFINED TYPE */
                     ROOT=1; SYM(1)=SN(1); ALT(1)=FAIL; SUC(1)=OK;
 42      1           CALL RECRED;
 46      1           PUT EDIT('N  SYMBOL    DEFNITION ALTERNATE SUCCESSOR ')
 47      1                 (PAGE,A);
                     DO P=1 TO 30;
 48      1           PUT EDIT(P,SYM(P),DEF(P),ALT(P),SUC(P))
 49      1    1        (SKIP(1),F(2),X(1),A(6),X(4),(3)(F(5),X(5)));
 50      1    1      END;
 51      1           CALL REDRED;
 52      1           PUT EDIT('N  SYMBOL    DEFNITION ALTERNATE SUCCESSOR ')
                       (PAGE,A);
 53      1           DO P=1 TO 30;
 54      1    1      PUT EDIT(P,SYM(P),DEF(P),ALT(P),SUC(P))
                       (SKIP(1),F(2),X(1),A(6),X(4),(3)(F(5),X(5)));
 55      1    1      END;
```

```
        /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST
 56      1          GRAPH:
                      PROCEDURE;
 57      2           DECLARE (PART(2),SECTION(7)) LABEL;
 58      2           DECLARE (X1,X2,NEW,CURRENT) BIN FIXED ;
 59      2           DECLARE SKIP(7) BIN FIXED INIT(2,1,2,2,1,1,2);
 60      2           DECLARE I BIN FIXED;
 61      2           X1,NEW,CURRENT=1; X2=2;

 63      2          NEXT_COMPONENT:
                      X1 = READ;
 64      2           X2=SKIP(X1);
 65      2           NEW=CNS(SYMBOL,0,FAIL,OK);
 66      2           GO TO PART(X2);
 67      2          PART(1): /* WHEN AND WHILE 1ST ALTERNATIVE */
 68      2           IF RECRSIV(SYMBOL) THEN GO TO ERROR1;
 69      2          PART(2):
                      GO TO SECTION(X1);

 70      2          SECTION(1): /* SUBJECT (DEFINED TYPE) */
 71      2    1       DO I=2,5,6; SKIP(I)=1; END;
 73      2           ST=ST+1;
 74      2           IF ST>100 THEN DO;
 76      2           PUT LIST('SYMBOL TABLE OVFLO.');
 77      2           STOP;
```

```
 81      2            X2=1;
 82      2            GO TO SET;

 83      2          SECTION(2): /* 1ST COMPONENT OF 1ST ALTERNATIVE */
                      /* PUT IN TABLE OF FIRSTS */
                      F=1;
 84      2            FP(F)=NEW;
                       /* LINK SYMBOL TABLE */
 85      2            SYMTAB.PT(ST)=NEW;
 86      2            GO TO SET;

 87      2          SECTION(3):     /* FIRST COMPONENT AFTER OR*/
 88      2    1       DO I=2,5,6; SKIP(I)=2; END;
 90      2            F=1;
 91      2            IF SYM(NEW)= SYM(FP(1)) THEN DO;
 93      2            CALL DELETE(NEW);
 94      2            FACT=ON;
 95      2            GO TO NEXT_COMPONENT;
 96      2            END;
 97      2            ELSE DO;
 98      2            ALT(FP(1))=NEW;
 99      2            FP(1)=NEW;
100      2            GO TO SET;
101      2            END;

102      2          SECTION(4): /* NOT FIRST COMPONENT AND FACTORING */
103      2            IF SUC(FP(F))=OK THEN DO;
```

/* CONVERSION FROM BNF TO SYNTAX GRAPH */

```
STMT LEVEL NEST

104      2            SUC(FP(F))=NEW;
105      2            ALT(NEW)=OK;
106      2            FACT=OFF;
107      2            GO TO SET;
108      2            END;
109      2            P=SUC(FP(F));
110      2            DO WHILE(ALT(P)>0 ); P=ALT(P); END;
113      2            IF SYM(P)=SYM(NEW) THEN DO;
115      2            FACT=ON;
116      2            F=F+1;
117      2            FP(F)=P;
118      2            CALL DELETE(NEW);
119      2            GO TO NEXT_COMPONENT;
120      2            END;
121      2            DO I=1 TO L;
122      2    1       IF SYM(P)= SYM(LP(I)) THEN DO;
                      /* FACTORING ON A LAST COMPONENT */
124      2    1       IF USED SUC(I) THEN DO;
126      2    1       CURRENT.LP(I).SUC(FP(F))=COPY(LP(I));
127      2    1       ALT(SUC(FP(F)))=NEW;
128      2    1       GO TO SET;
129      2    1       END;
130      2    1       ELSE DO;
```

```
135     2   1       END;
136     2   1       END;

                SECTION(5): /* LAST COMPONENT */
137     2           DO I=1 TO L WHILE(SYM(LP(I))¬=SYM(NEW)); END;
138     2   1       DO I=1 TO L WHILE(SYM(LP(I))¬=SYM(NEW)); END;
139     2           IF SYM(NEW)= SYM(LP(I)) THEN DO;
141     2           IF USEDALT(I) THEN DO;
143     2           SUC(CURRENT)=NEW;
144     2           GO TO ADD_TO_LASTS;
145     2           END;
146     2           ELSE DO;
147     2           USEDSUC(I)='1'B;
148     2           CALL DELETE(NEW);
149     2           SUC(CURRENT)=LP(I);
150     2           CURRENT=SUC(CURRENT);
151     2           GO TO NEXT_COMPONENT;
152     2           END;
153     2           END;
154     2           ELSE SUC(CURRENT)=NEW;
                    /* ADD TO THE LIST OF LASTS */
155     2         ADD_TO_LASTS:
                    L=L+1;
156     2           LP(L)=NEW;
157     2           USEDALT(L),USEDSUC(L)='0'B;
158     2           GO TO SET;

159     2         SECTION(6): /*INTERMEDIATE COMPONENT */
```

```
        /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

                    SUC(CURRENT)=NEW;

160     2         SET:
                    CURRENT=NEW;
161     2           GO TO NEXT_COMPONENT;

162     2         SECTION(7):
                    CALL DELETE(NEW);
163     2           RETURN;

164     2         ERROR1:
                    PUT EDIT('FIRST ALTERNATIVE IS RECURSIVE.')(SKIP(1),A);
165     2           STOP;
166     2         END GRAPH;
```

```
        /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

167     1       READ:
                  PROCEDURE BIN FIXED;
                  DECLARE TERMINAL BIT(1) INITIAL('0'B);
168     2         DECLARE LABL(3) LABEL,JUMP BIN FIXED ;
169     2         DECLARE I BIN FIXED;
170     2         CALL GLOT;
171     2         IF SUBSTR(STRING,CHAR,1)='|' THEN DO;
172     2         OR=ON;
174     2         CHAR=CHAR+1;
175     2         CALL GLOT;
176     2         IF EOF THEN RETURN(7);
177     2         END;
179     2         IF SUBSTR(STRING,CHAR,1)¬= '<' THEN DO;
180     2         IF SUBJECT THEN GO TO ERROR1;
182     2          TERMINAL='1'B;
184     2         DO I=0 TO 6 WHILE(SPECIAL(SUBSTR(STRING,CHAR+I,1))); END;
185     2         IF I>6 THEN GO TO ERROR5;
187     2         IF I=0 THEN GO TO ERROR6;
189     2         END;
191     2         ELSE DO;
192     2         IF SUBJECT THEN JUMP=1;
193     2         CHAR=CHAR+1;
195     2         CALL GLOT;
196     2         DO I=0 TO 6 WHILE (LETTER(SUBSTR(STRING,CHAR+I,1))); END
197     2         IF I>6 THEN GO TO ERROR2;
199     2         END;
201     2
202     2         GET STRING(STRING) EDIT(SYMBOL)(X(CHAR-1),A(I));
203     2         CHAR=CHAR+I;
204     2         CALL GLOT;
205     2         IF SUBSTR(STRING,CHAR,1) ¬= '>' THEN DO;
207     2         IF -TERMINAL THEN GO TO ERROR3;
209     2         TERMINAL='0'B;
210     2         END;
211     2         ELSE DO;
212     2         CHAR=CHAR+1;
213     2         CALL GLOT;
214     2         END;
215     2         GO TO LABL(JUMP);
216     2       LABL(1): /* SUBJECT SYMBOL */
                  CHAR=INDEX(STRING,'::=') + 3;
217     2         IF CHAR=3 THEN GO TO ERROR4;
219     2         JUMP=2;
220     2         SUBJECT,OR=OFF;
221     2         RETURN(1);
222     2       LABL(2): /* FIRST AFTER SUBJECT */
                  JUMP=3;
223     2         RETURN(2);
224     2       LABL(3): /* ALL OTHERS */
[illegible]     1           IF OR THEN DO
```

```
/* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

      2           OR=OFF;
      2           RETURN(3);
      2           END;
      2           IF FACT THEN RETURN(4);
      2           IF SUBSTR(STRING,CHAR,1)='|' THEN DO;
      2           OR=ON;
      2           CHAR = CHAR + 1 ;
      2           RETURN(5);
      2           END;
      2           RETURN(6);
      2         ERROR1: PUT EDIT('BEGIN DELIMITER MISSING')(SKIP(1),A);STOP
      2         ERROR2: PUT EDIT('COMPONENT > 10 CHARS.')(SKIP(1),A);STOP;
      2         ERROR3: PUT EDIT('END DELIMITER MISSING.')(SKIP(1),A);STOP;
      2         ERROR4: PUT EDIT('SUBJECT SEPARATOR MISSING')(SKIP(1),A);
      2                 STOP;
      2         ERROR5: PUT EDIT('TERMINAL > 6 CHARS.')(SKIP(1),A);STOP;
      2         ERROR6: PUT EDIT('INVALID CHAR=',SUBSTR(STRING,CHAR,1))
      2                 (SKIP(1),A,A);STOP;
      2         END READ;
```

```
/* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

      1         GLOT:
                  PROCEDURE;
      2           DO  WHILE(SUBSTR(STRING,CHAR,1)=' ');
      2           CHAR=CHAR+1;
      2           IF CHAR >81 THEN DO;
      2           END_OF_FILE:
      2           ON ENDFILE(SYSIN) BEGIN;
      3           PUT EDIT('ABNORMAL EOF ENCOUNTERED')(SKIP(1),A);
      3           PUT EDIT('PROGRAM TERMINATED,')(SKIP(1),A);
      3           STOP;
      3           END;
      2         GET_ONE_MORE:
                  GET EDIT(STRING)(A(80));
      2           PUT EDIT(STRING)(SKIP(1),A);
      2           IF INDEX(STRING,'.SYNTAX.') ¬= 0 THEN GO TO GET_ONE_MORE;
      2           IF INDEX(STRING,'.ENDSYNTAX.') ¬= 0 THEN DO;
      2           EOF=ON;
      2           STRING=' ';
      2           RETURN;
      2           END;
      2           SUBSTR(STRING,81,1)='|';
      2           CHAR=1;
      2           SUBJECT='1';
      2           END;
      2           END;
```

```
        /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

279     1         CNS:
                    PROCEDURE($SYM,$DEF,$ALT,$SUC) BIN FIXED;
280     2           DECLARE $SYM CHAR(6),($DEF,$ALT,$SUC,$P) BIN FIXED;
281     2           IF AVAIL=0 THEN DO;
283     2           CALL COLLECT;
284     2           IF AVAIL =0 THEN DO;
286     2           PUT EDIT('AVAIL UNDERFLOW.')(SKIP,A);
287     2           STOP;
288     2           END;
289     2           END;
290     2           $P=AVAIL; AVAIL=DEF(AVAIL);
292     2           SYM($P)=$SYM;DEF($P)=$DEF;ALT($P)=$ALT;SUC($P)=$SUC;
296     2           RETURN($P);
297     2         END CNS;
```

```
        /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

298     1         DELETE:
                    PROCEDURE(P) ;
299     2           DECLARE P BIN FIXED;
300     2           IF P<1 THEN RETURN;
302     2           ALT(P)=FAIL;
303     2           SUC(P)=OK;
304     2         A: DEF(P)=AVAIL; AVAIL=P; SYM(P)='      ';
307     2           RETURN;
308     2         END DELETE;
```

```
        /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

309     1       COPY:
                  PROCEDURE(P) BIN FIXED;
310     2         DECLARE P BIN FIXED;
311     2         RETURN(CNS(SYM(P),DEF(P),FAIL,OK));
312     2       END COPY;
```

```
        /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

313     1       RECRSIV:
                  PROCEDURE($S) BIT(1);
314     2         DECLARE $S CHAR(6),I BIN FIXED;
315     2         DO I=1 TO ST; IF $S=SM(ST) THEN RETURN('1'B); END;
319     2         RETURN('0'B);
320     2       END RECRSIV;
```

```
        /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

321      1          LETTER:
                      PROCEDURE($) BIT(1);
                      DECLARE $ CHAR(1),(FROM,TO,I) BIN FIXED;
322      2
323      2            DECLARE TRMNLS CHAR(45)
                      INIT('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789;=+-*/'''()')||
                      TRMNL (45) CHAR(1) DEFINED TRMNLS;
                      FROM=1; TO=26; GO TO XEQ;
324      2

327      2          SPECIAL: ENTRY($) BIT(1);
328      2            FROM=1; TO = 45;
330      2          XEQ:
                      DO I=FROM TO TO ;
331      2    1       IF $=TRMNL(I) THEN RETURN('1'B);
333      2    1       END;
334      2            RETURN('0'B);
335      2          END LETTER;
```

```
        /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

336      1          PUTDEFN:
                      PROCEDURE(P) RECURSIVE;
337      2            DECLARE P BIN FIXED;
338      2            IF P<1 THEN RETURN;
340      2            IF SUC(P)<1 & ALT(P)<1 THEN GO TO A;
342      2            CALL PUTDEFN(ALT(P));
343      2            CALL PUTDEFN(SUC(P));
344      2          A: DEF(P)=SEARCH(SYM(P));
345      2            RETURN;
346      2          END PUTDEFN;
```

```
          /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

347     1          SEARCH:
                     PROCEDURE($S) BIN FIXED;
348     2            DECLARE $S CHAR(6), I BIN FIXED;
349     2            DO I=1 TO ST;
350     2    1       IF $S=$M(I) THEN RETURN(PT(I));
352     2    1       END;
353     2            RETURN(0);
354     2          END SEARCH;
                    (CHECK(TRAVRSE,I)):
355     1           (SUBRG):
```

```
          /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

                   RECRED:
                     PROCEDURE;
356     2            DECLARE I BIN FIXED;
357     2            DO I=1 TO ST;
358     2    1       CALL TRAVRSE(PT(I));
359     2    1       END;
360     2            RETURN;
                    (CHECK(REPLACE)):
```

```
/* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

361     2       TRAVRSE: /* IN INVERSE POSTFIX ORDER */
                  PROCEDURE(Q) RECURSIVE;
                  DECLARE Q BIN FIXED;
362     3         IF Q<1 THEN RETURN;
363     3         IF ALT(Q)<1 & SUC(Q)<1 THEN GO TO A;
365     3         CALL TRAVRSE(SUC(Q));
367     3         CALL TRAVRSE(ALT(Q));
368     3       A:
369     3         CALL REPLACE(Q);
370     3         RETURN;
                 (CHECK(R,J,K,LEFT)):
```

```
/* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

371     3       :PLACE:
                  PROCEDURE(R);
372     4         DECLARE (P,J,K,R) BIN FIXED;
373     4         P=R;
374     4         IF SYM(P)¬=SM(I) THEN /* NON-RECURSIVE*/ RETURN;

376     4         IF SUC(P-1)=P THEN DO;
378     4           IF SUC(P)=0K THEN /* RIGHT REDURSIVE*/ DO;
380     4           SUC(P-1)=DEF(P);
                    /* NODE P WILL BE LOST */
381     4           RETURN;
382     4           END;
383     4           ELSE /* MIDDLE RECURSIVE*/ RETURN;
384     4         END;

385     4         J=PT(I);
386     4         DO WHILE(J<P);
387     4         IF ALT(J)=P THEN /* LEFT RECURSIVE, $O: */ GO TO LEFT;
389     4         ELSE J=ALT(J);
390     4         END;
391     4         PUT EDIT('RECURSIVE BUT NOT L|R|M=',SYM(P))(SKIP,A,A);
392     4         STOP;

393     4       LEFT:
                  ALT(I)=ALT(P)
```

```
397     4     1       END;
398     4             K=SUC(P);
399     4             DO WHILE(ALT(K)>0 ); K=ALT(K); END;
402     4             ALT(K)=OK ; /* TO BE OK IF NOT FOUND*/
403     4             DEF(P)=AVAIL; AVAIL=P; /* DELETE P */
405     4             RETURN;
406     4           END REPLACE;
407     3           END TRAVRSE;
408     2           END RECRED;
```

```
          /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

409     1           COLLECT:
                      PROCEDURE;
410     2             DECLARE(I,J) BIN FIXED;
411     2             DO I=1 TO 100;
412     2     1       DO J=1 TO I-1,I+1 TO 100;
413     2     2       IF DEF(J)=I|ALT(J)=I|SUC(J)=I THEN GO TO OUT;
415     2     2       END;
416     2     1       DEF(I)=AVAIL;
417     2     1       AVAIL=I;
418     2     1     OUT:
                      END;
419     2             RETURN;
420     2           END COLLECT;
                    (SUBRG):
```

```
        /* CONVERSION FROM BNF TO SYNTAX GRAPH */

STMT LEVEL NEST

421     1          REDRED:  /* REDUNDANCY REDUCTION */
                     PROCEDURE;
422     2            DECLARE (I,J,K) BIN FIXED, JFLAG BIT(1) INIT('0'B);

423     2            DO I= 1 TO 100 ;
424     2    1        IF SYM(I)='       ' THEN GO TO NEXT;

426     2    1        DO J= I+1 TO 100;
427     2    2        IF SYM(I)=SYM(J) & DEF(I) = DEF(J) &
                      ALT(I) = ALT(J) & SUC(I)=SUC(J) THEN

428     2    2        DO K= I+1 TO J-1, J+1 TO 100 ;
429     2    3          IF SUC(K)=J THEN DO;
431     2    3            SUC(K)=I;
432     2    3            JFLAG=ON;
433     2    3            END;
434     2    3          END;

435     2    2        IF JFLAG THEN DO; CALL DELETE(J); JFLAG=OFF; END;
440     2    2       END;

441     2    1   NEXT:
                   END;
442     2          RETURN;
443     2        END REDRED;
444     1        END SYNTAX2;
```

```
     .SYNTAX.
 <PRG>::=<ASG>|<ASG>;<PRG>
  <ASG>::= <VAR> = <AEX>
<AEX>::=<TRM>|  <AEX><AOP> <TRM>
<TRM>::=<FCT> | <TRM>  <MOP> <FCT>
  <FCT>::= <VAR>|  <INT> | ( <AEX> )
 <VAR>::= V
<INT>::= I
<AOP>::= +  |  -
<MOP>::=  *  |  /
     .ENDSYNTAX.
```

N	SYMBOL	DEFNITION	ALTERNATE	SUCCESSOR
I		2	0	0
2	ASG	5	-1	3
3	;	0	0	4
4	PRG	2	-1	0
5	VAR	21	-1	6
6	=	0	-1	7
7	AEX	8	-1	0
8	TRM	12	9	0
9	AEX	8	-1	10
10	AOP	23	-1	11
11	TRM	12	-1	0
12	FCT	16	13	0
13	TRM	12	-1	14
14	MOP	25	-1	15
15	FCT	16	-1	0
16	VAR	21	17	0
17	INT	22	18	0
18	(	0	-1	19
19	AEX	8	-1	20
20	)	0	-1	0
21	V	0	-1	0
22	I	0	-1	0
23	+	0	24	0
24	-	0	-1	0
25	*	0	26	0
26	/	0	-1	0
27		28	-1	0
28		29	0	0
29		30	0	0

N	SYMBOL	DEFNITION	ALTERNATE	SUCCESSOR
1	PRG	2	-1	0
2	ASG	5	-1	3
3	;	0	0	2
4	PRG	2	-1	0
5	VAR	21	-1	6
6	=	0	-1	7
7	AEX	8	-1	0
8	TRM	12	-1	10
9	AEX	27	-1	10
10	AOP	23	0	8
11		13	-1	0
12	FCT	16	-1	14
13	TRM	9	-1	14
14	MOP	25	0	12
15		11	-1	0
16	VAR	21	17	0
17	INT	22	18	0
18	(	0	-1	19
19	AEX	8	-1	20
20	)	0	-1	0
21	V	0	-1	0
22	I	0	-1	0
23	+	0	24	0
24	-	0	-1	0
25	*	0	26	0
26	/	0	-1	0
27		28	-1	0
28		29	0	0
29		30	0	0
30		31	0	0

HASP JOB 128 STATISTICS -- 457 CARDS READ -- 1,358 LINES PRINTED -