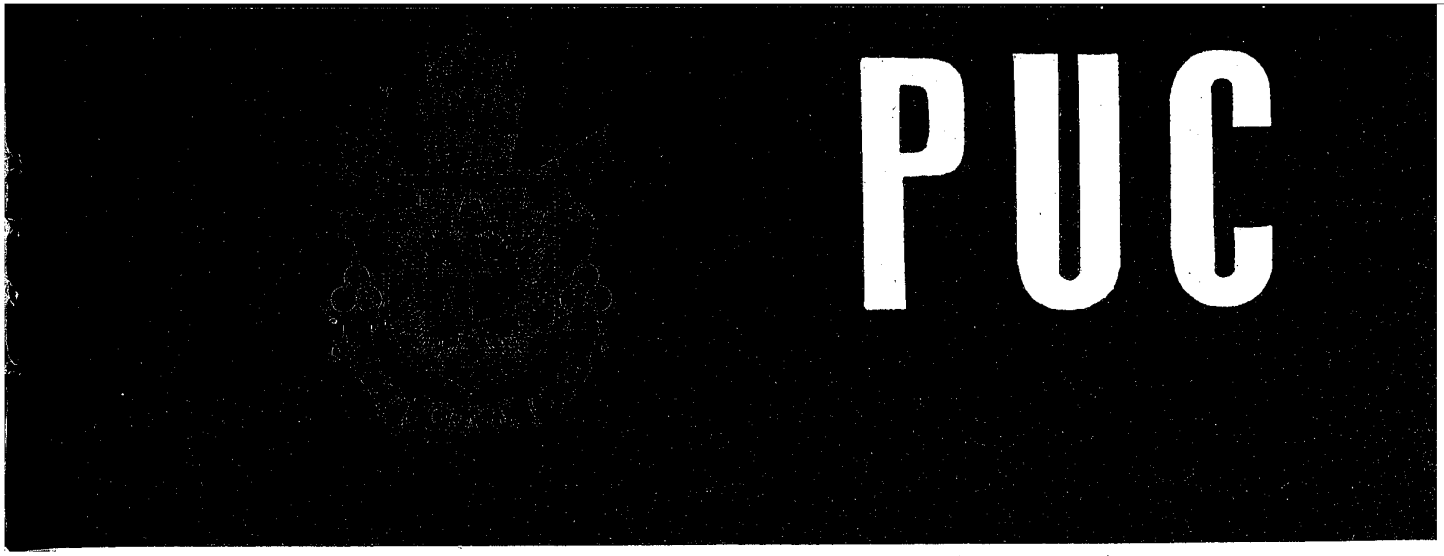

Series: Monographs in Computer Science
and Computer Applications

Nº 8/69

A RECOGNIZER FOR A PRECEDENCE GRAMMAR

by

Sergio Eduardo Rodrigues de Carvalho

Computer Science Department - Rio Datacentro

CENTRO TÉCNICO CIENTÍFICO
Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 209 — ZC-20
Rio de Janeiro — Brasil

A RECOGNIZER FOR A PRECEDENCE GRAMMAR

CONTENTS

Sergio Eduardo Rodrigues de Carvalho

RIO DATACENTRO

A RECOGNIZER FOR A PRECEDENCE GRAMMAR

1. Introduction and definitions

This paper describes an algorithm which analyses a given string of symbols, stating whether the string belongs to the language generated by a given grammar or not. The recognizer represented here may be classified as a left-right, bottom-up recognizer, in the sense that the string is repeatedly searched from left to right until a substring is found which may be replaced by a certain symbol.

In order to describe the recognizer, some basic definitions are introduced here.The vocabulary V is a given set, the elements of which are called symbols. The vocabulary V is the union of A, the set of non-terminal symbols (represented here by the capital letters at the end of the alphabet: S, T, U, and so on), and B, the set of terminal symbols (represented here by theletters i, j, k, and, in case of special characters, by the characters themselves). A string of symbols is a finite sequence of symbols (either of the terminal or of the non-terminal type); strings will be represented by lowercase letters u, v, x, y, z. As a particular case, a string may be empty; this is the null string.

A production is defined as a relation of the form $S \rightarrow x$. The left part of a production is defined as the non-terminal (S) being defined; the right part of a production is the string (x) which defines the non-terminal. A phrase structure grammar is a finite set of productions with the following properties:

a) x is a non empty string whose symbols belong to V;
b) S is a non-terminal symbol, belonging to A;
c) x can contain all the symbols in V, with one and only one exception, called the axiom.

A string z is a direct derivative of a string y if and only if there exists strings u, v (possibly empty) such that y=u U v and z=u x v, and also there exists a production of the form U → x in the grammar. A string z is a derivative of a string y if and only if there exists a sequence of strings $x_0,x_1,\ldots,x_n$ such that $y=x_0,x_0\rightarrow x_1,\ldots,x_{n-1}\rightarrow x_n,x_n=z$.

A parse of a string x with respect to the axiom S consists in a sequence of replacements made on the string x, each one substituting a substring by a certain non-terminal symbol. The parse is completed when a substring is replaced by the axiom S. A substring may be replaced when there exists a production U→y in the grammar, such that the string y is equal to the substring to be replaced. A canonical parse is defined as a parse in which at each time a replacement is to be made the substring to be replaced is the leftmost substring of the string being considered. The leftmost substring is called the handle. A grammar is said to be unambiguous if for each string x belonging to the language it generates there exists only one canonical parse.

The algorithm to be described here is based upon certain relations which may exist among the symbols belonging to V. These relations, called the precedence relations, are derived from the set of productions which ~~consists~~ exist in the grammar of the language. A grammar for which the relation between any two symbols is unique is called a precedence grammar.

Now that the necessary terminology is available, the recognizer can be described in more details. The recognizer will try to execute, by means of successive steps, the canonical parse of the given string; then, at each step, the first action to be taken is the search for the handle. When it is found, it is replaced by the corresponding non-terminal symbol and the process continues. The handle can be easily found if we use the precedence relations mentioned earlier. The relations are stored in a matrix, and are of the following types:

a) between all adjacent symbols in the handle, relation $\doteq$ holds;

b) between the leftmost symbol of the handle and the symbol immediately preceding the handle, relation $\lessdot$ holds;

c) between the rightmost symbol of the handle and the symbol immediately after the handle, relation $\gtrdot$ holds.

Once the precedence matrix is constructed, the process of finding the handle and reducing the string becomes very simple; it consists in a scan on the string searching a pair of sucessive symbols for which the relation $\gtrdot$ holds; when such a pair is found a scan is made leftwards looking for a pair of adjacent symbols with the relation $\lessdot$. All the symbols belonging to the substring so delimited are part of the handle and may be replaced. The process is then repeated until the last replacement is made yielding the axiom; in this case, the given string belongs to the language. The subsequent sections will provide a detailed description of the process, together with some restrictions made on the acceptable grammars and strings.

2. The precedence Matrix

The precedence relations can be defined in the following way (see Ref.1).

a) For any ordered pair of symbols (S,T) relation $S \doteq T$ holds if and only if there exists a production of the form $U \rightarrow x\ S\ T\ y$, for some non-terminal U and some (possibly empty) strings x, y.

b) For any ordered pair of symbols (S,T) relation $S \lessdot T$ holds if and only if there exists a production of the form $U \rightarrow x\ S\ V\ y$, for some U, x, V, and y and there exists a derivation of the form $V \rightarrow Tz$ for some z.

c) For any ordered pair of symbols (S,T), relation $S \gtrdot T$ holds if and only if (1) there exists a production of the form $U \rightarrow x\ V\ T\ y$, for some U, x, V, y and there exists a derivation of the form $V \rightarrow z\ S$ for some z; or (2) there exists a production of the form $U \rightarrow x\ V\ W\ y$, for some x, V, W, y and there exists derivations of the form $V \rightarrow u\ S$ and $W \rightarrow T\ v$ for some u, v.

In order to simplify the algorithm for constructing the precedence matrix, the definitions of left (or right) part of a non-terminal symbol S are now introduced.

The left (or right) part of a non-terminal symbols S is a set of symbols T_i (either terminal or non-terminal) with the following properties:

a) They are the leftmost (or rightmost) symbol of the right part of a production relative to S:

$$S \rightarrow T_i\ x \text{ (or } S \rightarrow x\ T_i)$$

b) They belong to the left (or right) part of a non-terminal S_1 which in turn satisfies condition a:

$$S \rightarrow S_1\ x \text{ (or } S \rightarrow x\ S_1) \text{ and } S_1 \rightarrow T_i\ y \text{ (or } S_1 \rightarrow y\ T_i)$$

The algorithm for constructing the precedence matrix consists of a search on the productions of the grammar looking for the left and right parts of non-terminal symbols. The algorithm can be outlined as follows: starting with the first production of the grammar and the first pair of adjacent symbols of the production, and scanning all the adjacent symbols and all productions:

a) assign relation $\doteq$ to the position in the matrix relative to the symbols in question;
b) if the second symbol of the pair is non-terminal, assign relation $<$ to the positions in the matrix relative to the line of the first symbol and to the columns of the symbols belonging to the left part of the second symbol of the pair;
c) if the first symbol of the pair is non-terminal, assign relation $>$ to the positions in the matrix relative to the column of the second symbol and the lines of the symbols belonging to the right of the first symbol of the pair;
d) if both symbols of the pair are non-terminal, assign relation $>$ to the positions in the matrix relative to the lines corresponding to the right part of the first symbol and the columns corresponding to the left part of the second symbol of the pair.

This part of the algorithm is performed by subroutines PREMAT and PART. Subroutine PREMAT assigns the adequate relations to the positions indicated by subroutine PART. The given grammar, upon which the scan is to be made, must conform to the following specifications:

a) no equal right parts are allowed in two or more different productions;
b) the right part of the production relative to the axiom must be delimited by terminal symbols which can not appear in any other producti n;

On the programs that follow, three other restrictions are made:

c) the delimiter mentioned in b must be the last symbol of the vocabulary when input is made;
d) the production relative to the axiom must be of the form S → 'T', where S can not appear in a right part of any other production.
e) All the symbols used are restricted to be one character in length.

Subroutine PART finds the left or right parts of a given non-terminal. The set of symbols found is stored in a vector.

This subroutine uses two stacks, one for the non-terminal symbols found during the search and the other for the number of the production being scanned. Each non-terminal found during the search is stacked and the subroutine tries to find its corresponding left (or right) part. When all the productions relative to the symbol in the top of the stack have been scanned out, the stacks are popped up and the search continues relative to the new top symbol. The search stops when the stack is empty.

Before each assignment is made subroutine PREMAT checks to see if the position in question is already occupied.

3. The Parser

Once the grammar is found to be of the precedence type, strings of symbols can be tested. This is made by subroutine PARSER, which executes the canonical parse. This subroutine scans the string looking for the end of a handle and then scans backwards looking for the beginning of the same handle. The last symbol of the handle is the first symbol found in the forward scan for which the relation with its successor is of the $\triangleright$ type. The first symbol of the handle is the first symbol found in the backward scan for which the relation with its predecessor is of the $\triangleleft$ type.

The productions are then searched for a right part which matches the handle so delimited. This subroutine works with a stack holding the scanned symbols and a vector where the relations of the type $\lessdot$ found during the forward scan are kept in. The use of this vector avoids the necessity of the backward scan upon the string. Once a handle is replaced, the scan starts at the position immediately after the rightmost element of the replaced handle.
This subroutine checks for input symbols not belonging to the vocabulary and for incorrect sequences of symbols on the given strings.

4. Results

This section presents a listing of the programs (written in FORTRAN IV) and results obtained in an IBM 7044 computer. The grammars used are described in Ref. 1 and Ref. 2, and, for illustrative purposes, are produced at the output of the program, together with their respective vocabularies.
Because of the limited character set of the 7044 system, relations $\doteq$, $\lessdot$ and $\gtrdot$ are represented by the symbols =, * and +. Positions of the precedence matrix filled with zeros represent impossible relations between the corresponding symbols.

References

1. N.Wirth and H. Weber: Euler, a generalization of ALGOL and its formal definition Part 1. Communications of the ACM, Jan. 66, pp 13-25.
2. J.Feldman and D. Gries: Translator writing systems, Communications of the ACM, Feb. 68, pp 77-113
3. F.R.A. Hopgood: Compiling Techniques
Mac Donald, ed. 1969.

```
      0 $IBFTC SC04
        C
        C       THIS IS A PARSER FOR A PRECEDENCE GRAMMAR . IT USES
        C       THREE SUBPROGRAMS. THE FIRST CONSTRUCTS THE PRECEDENCE
        C       MATRIX TO BE USED IN THE PARSING. THE SECOND FINDS THE
        C       LEFT OR RIGHT PARTS OF A PRODUCTION. THE THIRD PARSES
        C       THE INPUT SENTENCE CHECKING WHETHER THE SENTENCE
        C       BELONGS TO THE LANGUAGE OR NOT.
        C
        1       COMMON DEF(20),PROD(20,50),TERMV(50),PRMAT(50,50)
        2       COMMON PPART(50),VOC(100),NUM(20),STRING(30)
        3       COMMON NPROD,NTERM,NSYMB,KS
        4       INTEGER DEF,PROD,TERMV,PRMAT,PPART,VOC,STRING
        C
        C       -----INPUT AND OUTPUT OF SYNTAX AND VOCABULARIES-----
        C
        5     9 READ 1000,NPROD
        7  1000 FORMAT(I5)
       10       DO 100 I=1,NPROD
       11       READ 1001,K,DEF(I),(PROD(I,J),J=1,K)
       17  1001 FORMAT(I5,75A1)
       20   100 NUM(I) = K
       22       READ 1001,NTERM,(TERMV(I),I=1,NTERM)
       30       READ 1001,NSYMB,(VOC(I),I=1,NSYMB)
       36       PRINT 1020
       37  1020 FORMAT(1H=//////)
       40       PRINT 1002
       41  1002 FORMAT(//15X,24HTHIS IS THE GIVEN SYNTAX/)
       42       DO 110 I=1,NPROD
       43       K = NUM(I)
       44   110 PRINT 1003,DEF(I),(PROD(I,J),J=1,K)
       52  1003 FORMAT(/15X,A1,3H = ,74A1)
       53       PRINT 1004, (VOC(I),I=1,NSYMB)
       60  1004 FORMAT(//15X,22HTHIS IS THE VOCABULARY//15X,20(A1,1H,))
       61       PRINT 1005, (TERMV(I),I=1,NTERM)
       66  1005 FORMAT(//15X,31HTHIS IS THE TERMINAL VOCABULARY//
            115X,20(A1,1H,))
        C
        C       -----CONSTRUCTION OF THE PRECEDENCE MATRIX-----
        C       -----TEST IF GRAMMAR OK-----
        C
       67       CALL PREMAT(IND)
       70       GO TO (1,2,3) , IND
       71     1 PRINT 1006
       72  1006 FORMAT(///30X,
            145HTHE GIVEN GRAMMAR IS NOT A PRECEDENCE GRAMMAR)
       73       CALL EXIT
       74     3 PRINT 1007
       75  1007 FORMAT(///30X,
            140HA SYMBOL WAS NOT FOUND IN THE VOCABULARY)
       76       CALL EXIT
       77     2 PRINT 1008,(VOC(I),I=1,NSYMB)
      104  1008 FORMAT(//15X,29HTHIS IS THE PRECEDENCE MATRIX//
            121X,30(A1,2H  )//)
      105       DO 120 I=1,NSYMB
```

```
      106   120 PRINT 1009,  VOC(I), (PRMAT(I,J),J=1,NSYMB)
      114  1009 FORMAT(18X,A1,20(2H  ,A1)//)
C
C          -----INPUT AND PARSE OF STRING TO BE TESTED------
C
      115     7 READ 1010,LASTS,LASTG,KS,(STRING(I),I=1,KS)
      125  1010 FORMAT(2I3,I4,70A1)
      126       PRINT 1011, (STRING(I),I=1,KS)
      133  1011 FORMAT(///15X,29HINPUT STRING TO BE TESTED ---,70A1)
      134       CALL PARSER(KEY)
      135       GO TO (4,5),KEY
      136     4 PRINT 1012
      137  1012 FORMAT(//30X,36HGIVEN STRING BELONGS TO THE LANGUAGE)
      140       GO TO 6
      141     5 PRINT 1013
      142  1013 FORMAT(//30X,
     1    44HGIVEN STRING DOES NOT BELONG TO THE LANGUAGE)
      143     6 IF(LASTS.EQ.1) GO TO 8
      146       GO TO 7
      147     8 IF(LASTG.EQ.1) CALL EXIT
      152       DO 130 I=1,NSYMB
      153       DO 130 J=1,NSYMB
      154   130 PRMAT(I,J) = 0
      157       GO TO 9
      160       END
```

```
081207SRCASERGIO CARVALHO                    IBMAP ASSEMBLY SC04

   NO MESSAGES FOR ABOVE ASSEMBLY
```

```
      $IBFTC SC01
      SUBROUTINE PREMAT(IND)
C
C     THIS SUBROUTINE CONSTRUCTS THE PRECEDENCE MATRIX
C     RELATIVE TO THE GIVEN GRAMMAR. THE MATRIX(PRMAT) WILL
C     CONSIST OF THE SYMBOLS =,+,*. EACH TIME A NEW SYMBOL
C     IS TO BE PUT INTO THE MATRIX,A TEST IS MADE TO SEE
C     IF THE POSITION BEING CONSIDERED IS ALREADY OCCUPIED.
C     THIS BEING THE CASE, A MESSAGE IS PRINTED INDICATING
C     POSITION AND SYMBOLS IN QUESTION.ALSO, PARAMETER
C     IND IS SET TO 1 MEANING THAT THE GIVEN GRAMMAR IS NOT
C     A PRECEDENCE GRAMMAR. THIS SUBROUTINE USES SUBROUTINE
C     PART, WHICH DETERMINES THE SYMBOLS BELONGING TO THE
C     LEFT OR RIGHT PARTS OF A GIVEN SYMBOL.
C
      COMMON DEF(20),PROD(20,50),TERMV(50),PRMAT(50,50)
      COMMON PPART(50),VOC(100),NUM(20),STRING(30)
      COMMON NPROD,NTERM,NSYMB,KS
      INTEGER DEF,PROD,TERMV,PRMAT,PPART,VOC,AUX(50),STRING
      DATA IEQ,IPREC,ISUC/1H=,1H*,1H+/
C
C     -----SET INITIAL VALUES-----
C
      IND = 2
      KT = 0
      I = 1
C
C     -----TEST IF PRODUCTION HAS MORE THAN ONE SYMBOL-----
C     -----IF NOT , GET ANOTHER PRODUCTION-----
C
    2 IF(NUM(I).GT.1) GO TO 1
   22 IF(I.EQ.NPROD) RETURN
      I = I + 1
      GO TO 2
C
C     -----IDENTIFY SYMBOLS TO BE CONSIDERED-----
C
    1 J = 1
   23 M = 1
      DO 110 K1=1,NSYMB
      IF(PROD(I,J).EQ.VOC(K1)) GO TO 3
  110 CONTINUE
      NS = PROD(I,J)
      GO TO 5
    3 DO 120 K2=1,NSYMB
      IF(PROD(I,J+1).EQ.VOC(K2)) GO TO 4
  120 CONTINUE
      NS = PROD(I,J+1)
C
C     -----SYMBOL NOT FOUND IN THE VOCABULARY-----
C
    5 PRINT 50, NS
   50 FORMAT(//30X,6HSYMBOL,2X,A1,3X,
     123HNOT FOUND IN VOCABULARY)
      IND = 3
```

```
      46          RETURN
C
C         -----THIS IS THE APPLICATION OF RULE A (SEE TEXT).-----
C         -----TEST IF POSITION EMPTY . IF SO , PUT = SIGN-----
C         -----IF NOT , PRINT MESSAGE AND RETURN IND=1-----
C
      47        4 IF(PRMAT(K1,K2).EQ.0)GO TO 6
      52          PRINT 51, K1,K2,PRMAT(K1,K2)
      53       51 FORMAT(//30X,9HPOSITION(,I3,1H,,I3,15H)OCCUPIED WITH ,
                 1A1,5H SIGN)
      54          IND = 1
      55          GO TO 7
      56        6 PRMAT(K1,K2) = IEQ
C
C         -----THIS IS THE APPLICATION OF RULES B AND C-----
C         -----THE RULE TO BE APPLIED DEPENDS ON THE VALUE OF M---
C
      57        7 IF(M.EQ.1)GO TO 8
C
C         -----TEST IF SYMBOL IS TERMINAL . IF NOT , CALL PART---
C         -----TO FIND THE LEFT OR RIGHT PART OF SYMBOL-----
C
      62       17 IF(K1.GT.(NSYMB-NTERM)) GO TO 9
      65          ISYM = VOC(K1)
      66          GO TO 11
      67        8 IF(K2.GT.(NSYMB-NTERM)) GO TO 12
      72          ISYM = VOC(K2)
      73       11 CALL PART(M,ISYM,N)
C
C         -----IDENTIFY SYMBOLS TO BE CONSIDERED-----
C
      74          DO 130 K3=1,N
      75          DO 140 K4=1,NSYMB
      76          IF(PPART(K3).EQ.VOC(K4)) GO TO 10
     101      140 CONTINUE
     103          NS = PPART(K3)
     104          GO TO 5
C
C         -----FILL IN APPROPRIATE POSITIONS IF NOT OCCUPIED-----
C
     105       10 IF(M.EQ.1)GO TO 13
     110          IF(PRMAT(K4,K2).NE.0)GO TO 14
     113          PRMAT(K4,K2) = ISUC
     114          GO TO 130
     115       13 IF(PRMAT(K1,K4).NE.0)GO TO 15
     120          PRMAT(K1,K4) = IPREC
     121          GO TO 130
     122       14 PRINT 51, K4,K2,PRMAT(K4,K2)
     123          IND = 1
     124          GO TO 130
     125       15 PRINT 51, K1,K4,PRMAT(K1,K4)
     126          IND = 1
     127      130 CONTINUE
C
C         -----TEST IF RULE B (M=1) OR RULE C (M=2) APPLIED-----
```

```
      C
  131     IF(M.EQ.2) GO TO 16
  134  24 M = 2
  135     GO TO 17
      C
      C     -----THIS IS THE APPLICATION OF RULE D (SEE TEXT)-----
      C
  136  16 IF(KT.EQ.0)GO TO 18
  141     KT = 0
  142     GO TO 9
      C
      C     -----FIND LEFT PART OF APPROPRIATE SYMBOL AND STORE-----
      C     -----IT IN AUXILIARY VECTOR-----
      C
  143  18 CALL PART (1,VOC(K2),N1)
  144     DO 150 K5=1,N1
  145     DO 160 K6=1,NSYMB
  146     IF(PPART(K5).EQ.VOC(K6)) GO TO 19
  151  160 CONTINUE
  153     NS = PPART(K5)
  154     GO TO 5
  155  19 AUX(K5) = K6
  156  150 CONTINUE
      C
      C     -----FIND RIGHT PART OF SYMBOL IN QUESTION AND PUT + ---
      C     -----SIGN INTO MATRIX, IF POSITION NOT OCCUPIED-----
      C
  160     CALL PART(2,VOC(K1),N2)
  161     DO 170 K7=1,N2
  162     DO 180 K8=1,NSYMB
  163     IF(PPART(K7).EQ.VOC(K8)) GO TO 20
  166  180 CONTINUE
  170     NS = PPART(K7)
  171     GO TO 5
  172  20 DO 190 K9=1,N1
  173     L = AUX(K9)
  174     IF(PRMAT(K8,L).EQ.0)GO TO 21
  177     PRINT 51,K8,L,PRMAT(K8,L)
  200     IND = 1
  201     GO TO 190
  202  21 PRMAT(K8,L) = ISUC
  203  190 CONTINUE
  205  170 CONTINUE
      C
      C     -----TEST IF END OF PRODUCTION-----
      C
  207   9 IF(J.EQ.(NUM(I)-1)) GO TO 22
  212     J = J + 1
  213     GO TO 23
  214  12 KT = 1
  215     GO TO 24
  216     END
```

```
      $IBFTC SC02
      SUBROUTINE PART(M,ISYM,N)
C
C     THIS SUBROUTINE FINDS THE LEFT (M=1) OR RIGHT (M=2)
C     PARTS OF A NON-TERMINAL SYMBOL. VECTOR PPART HOLDS
C     THE N SYMBOLS FOUND.  THIS SUBROUTINE USES TWO STACKS
C     ONE FOR THE SYMBOL BEING CONSIDERED AND OTHER FOR THE
C     INDEX OF THE PRODUCTION IN QUESTION. IF A NON-TERMINAL
C     IS FOUND TO BE IN PPART, IT IS STACKED AND THE PROGRAM
C     TRIES TO FIND ITS LEFT(OR RIGHT) PART. WHEN ALL THE
C     PRODUCTIONS HAVE BEEN SCANNED, A TEST IS MADE TO SEE
C     IF THE STACK IS EMPTY, IF NOT, THE SCAN CONTINUES,
C     IF EMPTY, RETURN.
C
      COMMON DEF(20),PROD(20,50),TERMV(50),PRMAT(50,50)
      COMMON PPART(50),VOC(100),NUM(20),STRING(30)
      COMMON NPROD,NTERM,NSYMB,KS
      INTEGER DEF,PROD,TERMV,PRMAT,PPART,VOC,STS(20),STP(20)
      INTEGER STRING
C
C     -----SET INITIAL VALUES-----
C
      I = 1
      N = 0
      STS(I) = ISYM
    8 STP(I) = 2
      L = STP(I)
C
C     -----FIND A PRODUCTION RELATIVE TO SYMBOL IN QUESTION---
C
    3 IF(DEF(L).EQ.STS(I)) GO TO 1
C
C     -----TEST IF END OF PRODUCTIONS-----
C
    7 IF(L.EQ.NPROD) GO TO 2
      L = L +1
      GO TO 3
C
C     -----FIND APPROPRIATE PART OF PRODUCTION-----
C
    1 STP(I) = L
      IF(M.EQ.1)GO TO 4
      LL = NUM(L)
      IT = PROD(L,LL)
      GO TO 5
    4 IT = PROD(L,1)
C
C     -----TEST IF SYMBOL ALREADY IN PPART.-----
C
    5 IF(N.EQ.0)GO TO 6
      DO 120 K2=1,N
      IF(IT.EQ.PPART(K2)) GO TO 7
  120 CONTINUE
    6 N = N + 1
      PPART(N) = IT
```

```
C
C       -----TEST IF SYMBOL IN QUESTION EQUALS TOP OF STACK-----
C
      47      IF(IT.EQ.STS(I)) GO TO 7
C
C       -----TEST IF SYMBOL IN QUESTION IS TERMINAL-----
C
      52      DO 110 K1=1,NTERM
      53      IF(IT.EQ.TERMV(K1)) GO TO 7
      56  110 CONTINUE
C
C       -----STACK NON-TERMINAL SYMBOL-----
C
      60      I = I + 1
      61      STS(I) = IT
      62      GO TO 8
C
C       -----TEST IS STACK IS EMPTY-----
C
      63    2 I = I - 1
      64      IF(I.EQ.0) RETURN
      67      L = STP(I)
      70      GO TO 7
      71      END
```

```
081207SRCASERGIO CARVALHO                    IBMAP ASSEMBLY SC02

   NO MESSAGES FOR ABOVE ASSEMBLY
```

```
      $IBFTC SCO3
      SUBROUTINE PARSER(KEY)
C
C     THIS SUBROUTINE PARSES AN INPUT STRING , RETURNING
C     KEY=1 IF THE STRING BELONGS TO THE LANGUAGE. THE
C     STRING SYMBOLS ARE STACKED UNTIL A RELATION OF THE TYPE
C     ISUC IS FOUND BETWEEN TWO ADJACENT SYMBOLS. DURING THE
C     STACKING,RELATIONS OF THE TYPE IPREC ARE KEPT IN AN
C     AUXILIARY VECTOR. WHEN AN ISUC IS FOUND, A SEARCH IS
C     MADE BACKWARDS IN THE AUXILIARY VECTOR UNTIL AN IPREC
C     IS FOUND. THE SUBSTRING SO DELIMITED MUST CONSIST OF
C     A RIGHT PART OF A PRODUCTION AND CAN BE REPLACED BY
C     THE NON-TERMINAL BEING DEFINED IN THAT PRODUCTION. AT
C     THE END OF THE PROCESS, THE STACK MUST CONTAIN THE
C     DEFINITION OF THE AXIOM.
C
      COMMON DEF(20),PROD(20,50),TERMV(50),PRMAT(50,50)
      COMMON PPART(50),VOC(100),NUM(20),STRING(30)
      COMMON NPROD,NTERM,NSYMB,KS
      INTEGER DEF,PROD,TERMV,PRMAT,PPART,VOC,STRING,
     1STACKS(30),STACKP(30),DEL
      DATA DEL/1H'/,ISUC,IPREC,IEQ/1H<,1H*,1H=/
C
C     -----TEST IF STRING STARTS WITH DELIMITER-----
C
      IF(STRING(1).EQ.DEL) GO TO 1
    3 PRINT 1000
 1000 FORMAT(//30X,
     141HINPUT STRING CONTAINS UNIDENTIFIED SYMBOL)
    6 KEY = 2
      RETURN
C
C     -----STACK DELIMITER AND START PARSING-----
C
    1 STACKS(1) = DEL
      K1 = NSYMB
      K = 2
      J = 2
C
C     -----LOOK FOR NEXT SYMBOL-----
C
   10 DO 120 K2=1,NSYMB
      IF(STRING(K).EQ.VOC(K2)) GO TO 2
  120 CONTINUE
      GO TO 3
C
C     -----SEARCH RELATION IN PRECEDENCE MATRIX AND-----
C     -----STACK INFORMATION-----
C
    2 IF(PRMAT(K1,K2).EQ.ISUC) GO TO 4
      IF(PRMAT(K1,K2).EQ.IPREC) GO TO 7
      IF(PRMAT(K1,K2).EQ.IEQ) GO TO 5
      PRINT 1001
 1001 FORMAT(//30X,28HINCORRECT SEQUENCE ON STRING)
      GO TO 6
```

```
   45      7 STACKP(J) = IPREC
   46      5 STACKS(J) = STRING(K)
           C
           C       -----TEST IF END OF STRING.IF SO,TEST CONTENTS OF-----
           C       -----STACK. IF NOT, PREPARE NEXT SEARCH-----
           C
   47            IF(K.LT.KS) GO TO 8
   52            IF(STACKS(2).EQ.VOC(1).AND.STACKS(3).EQ.DEL) GO TO 9
   55            PRINT 1002
   56      1002  FORMAT(//30X,20HINCOMPLETE REDUCTION)
   57            GO TO 6
   60      9     KEY = 1
   61            RETURN
   62      8     J = J + 1
   63            K = K + 1
   64            K1 = K2
   65            GO TO 10
           C
           C       -----SEARCH RELATION IPREC IN AUXILIARY VECTOR-----
           C
   66      4     M= J - 1
   67     12     IF(STACKP(M).EQ.IPREC) GO TO 11
   72            M = M - 1
   73            GO TO 12
   74     11     N = M
   75            L = 2
   76            I = 1
           C       -----SEARCH ADEQUATE PRODUCTION-----
           C
   77     15     IF(PROD(L,I).EQ.STACKS(N)) GO TO 13
  102            IF(L.EQ.NPROD) GO TO 14
  105            L = L + 1
  106            I = 1
  107            N = M
  110            GO TO 15
  111     14     PRINT 1003
  112     1003   FORMAT(//30X,31HNO PRODUCTIONS MATCH THE HANDLE)
  113            GO TO 6
  114     13     IF(N.EQ.(J-1)) GO TO 16
  117            N = N + 1
  120            I = I + 1
  121            GO TO 15
           C
           C       -----  REPLACE  HANDLE AND PREPARE NEXT SEARCH-----
           C
  122     16     STACKS(M) = DEF(L)
  123            J = M + 1
  124            DO 130 K3=1,NSYMB
  125            IF(STACKS(M-1).EQ.VOC(K3)) GO TO 17
  130     130    CONTINUE
  132            GO TO 3
  133     17     DO 140 K4=1,NSYMB
  134            IF(STACKS(M).EQ.VOC(K4)) GO TO 18
  137     140    CONTINUE
  141            GO TO 3
```

```
      142      18 IF(PRMAT(K3,K4).NE.IPREC) STACKP(M) = 0
      145         K1 = K4
      146         GO TO 2
      147         END
```

```
081207SRCASERGIO CARVALHO                    IBMAP ASSEMBLY SC03

   NO MESSAGES FOR ABOVE ASSEMBLY
```

```
THIS IS THE GIVEN SYNTAX

S = 'T'
T = U)
U = (
U = UI
U = UT

THIS IS THE VOCABULARY

T,U,),(,I,',

THIS IS THE TERMINAL VOCABULARY

),(,I,',

THIS IS THE PRECEDENCE MATRIX

     T  U  )  (  I  '
  T  +  +  +  +  +  =
  U  =  *  =  *  =  0
  )  +  +  +  +  +  +
  (  +  +  +  +  +  0
  I  +  +  +  +  +  0
  '  =  *  0  *  0  0

INPUT STRING TO BE TESTED ---'(I)'

                GIVEN STRING BELONGS TO THE LANGUAGE

INPUT STRING TO BE TESTED ---'(I(I))'

                GIVEN STRING BELONGS TO THE LANGUAGE

INPUT STRING TO BE TESTED ---'(III(I))'

                GIVEN STRING BELONGS TO THE LANGUAGE

INPUT STRING TO BE TESTED ---'()'
```

```
GIVEN STRING BELONGS TO THE LANGUAGE

INPUT STRING TO BE TESTED ---'(II())'

GIVEN STRING BELONGS TO THE LANGUAGE

INPUT STRING TO BE TESTED ---'(I)

INCOMPLETE REDUCTION

GIVEN STRING DOES NOT BELONG TO THE LANGUAGE

INPUT STRING TO BE TESTED ---'(I'

INCORRECT SEQUENCE ON STRING

GIVEN STRING DOES NOT BELONG TO THE LANGUAGE.
```

```
THIS IS THE GIVEN SYNTAX

S = 'T'
T = U
U = V
U = U+V
V = W
W = X
W = W*X
X = (T)
X = I

THIS IS THE VOCABULARY

T,U,V,W,X,I,+,*,(,),',

THIS IS THE TERMINAL VOCABULARY

I,+,*,(,),',

THIS IS THE PRECEDENCE MATRIX

     T  U  V  W  X  I  +  *  (  )  '
T    0  0  0  0  0  0  0  0  0  =  =
U    0  0  0  0  0  0  =  0  0  +  +
V    0  0  0  0  0  0  +  0  0  +  +
W    0  0  0  0  0  0  +  =  0  +  +
X    0  0  0  0  0  0  +  +  0  +  +
I    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

INPUT STRING TO BE TESTED ---'I'

                GIVEN STRING BELONGS TO THE LANGUAGE

INPUT STRING TO BE TESTED ---'I+I*I'
```

```
                    GIVEN STRING BELONGS TO THE LANGUAGE

     INPUT STRING TO BE TESTED ---'I**I'

                    INCORRECT SEQUENCE ON STRING

                    GIVEN STRING DOES NOT BELONG TO THE LANGUAGE

     INPUT STRING TO BE TESTED ---'(I+I*I)+I*I'

                    GIVEN STRING BELONGS TO THE LANGUAGE

     INPUT STRING TO BE TESTED ---'(I+I)'

                    INCORRECT SEQUENCE ON STRING

                    GIVEN STRING DOES NOT BELONG TO THE LANGUAGE

     INPUT STRING TO BE TESTED ---'((I+I)*I)*I'

                    GIVEN STRING BELONGS TO THE LANGUAGE

081207SRCASERGIO CARVALHO

081207SRC TEMP TOTAL 000178 SEG - TEMP EXEC. 000021 SEG - CART LIDOS 00
```