CS 57

THE USE OF TRANSITION MATRICES IN COMPILING

BY

D. GR IES

TECHNICAL REPORT CS 57
MARCH 17, 1967

Supported in part by the
Atomic Energy Commission.

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

THE USE OF TRANSITION MATRICES IN COMPILING

by

D. Gries

Supported in part by the Atomic Energy Commission.

The Use of Transition Matrices in Compiling

Contents | Page

The Use of a Transition Matrix in Compiling

1. Introduction,

The construction of efficient parsing algorithms for programming languages has been the subject of many papers in the last few years. Techniques for efficient parsing and algorithms which generate the parser from a grammar or phrase structure system have been derived. Some of the well-known methods are the precedence techniques of Floyd [4] and Wirth and Weber [10], and the production language of Feldman [3]. Perhaps the first such discussion was by Samelson and Bauer [9]. There the concept of the push-down stack was introduced, along with the idea of a transition matrix. A transition matrix is just a switching table which lets one determine from the top element of the stack (denoting a row of the table) and the next symbol of the program to be processed (represented by a column of the table) exactly what should be done. Either a reduction is made in the stack, or the incoming symbol is pushed onto the stack.

Considering its efficiency, the transition matrix technique does not seem to have achieved much attention, probably because it was not sufficiently well-defined. The purpose of this paper is to define the concept more formally, to illustrate that the technique is very efficient, and to describe an algorithm which generates a transition matrix from a suitable grammar. We will also describe other uses of transition matrices besides the usual ones of syntax checking and compiling.

We will require that the set of productions $\{U_i ::= x_i\}$ form an operator grammar (Floyd [4]), which means that no production has the form $U ::= xV_1V_2y$ for strings x,y and nonterminal symbols V_1 and V_2. This restriction is not necessary in order to use a transition matrix. One may also describe suitable conditions for the general phrase structure grammar $\{U_i ::= x_i\}$ which allow the use of a transition matrix. The restriction to operator grammars is a rather natural way to reduce the size of storage necessary to implement the technique. The syntax of the usual ALGOL-like languages can easily be represented by such a grammar.

We emphasize that the use of a transition matrix is just another technique, though a very efficient one, for parsing sentences of a suitable (programming) language.

Section 2 introduces the notation and terminology. Sections 3 through 5 are devoted to discussing sufficient conditions for a unique canonical parse which enable us to use a transition matrix. These conditions are of course closely related to those derived by Floyd [4], Wirth and Weber [10], and Eickel et al [2]. Sections 6 and 7 explain the technique and go through an example in detail. In Sections 8 and 9 practical examples and applications are discussed. Appendix A gives Floyd's ALGOL-like grammar ([4]) and the associated matrix and subroutines. All examples were produced by an algorithm, written in Extended ALGOL [11], on the B5500 at Stanford.

The author is indebted to Jerome Feldman and Niklaus Wirth for their critical comments on this manuscript. This work was partially supported by the U.S. Atomic Energy Commission.

2. Notation, terminology, basic definitions.

Let $\mathcal{V}$ be a given set: the vocabulary. Elements of $\mathcal{V}$ are called symbols and are denoted here by capital Latin letters, S , T , U , etc. Finite sequences of symbols - including the empty sequence (Λ) - are called strings and are denoted by small Latin letters u , v , y , z , etc. The set of all strings over $\mathcal{V}$ is denoted by $\mathcal{V}^*$.

If $z = xy$ is a string, x is a head and y a tail of z .

A production or syntactic rule $\varphi : U ::= x$, is an ordered pair consisting of a symbol U and a nonempty string x . U is called the left part and x the right part of φ . We assume that $U \neq x$.

Let $\mathcal{P}$ be a finite set of productions $\varphi_1, \ldots, \varphi_n$. y directly-produces z $(y \rightarrow z)$ and conversely z directly reduces into y , if and only if there exist strings u,v such that $y = uUv$, $z = uxv$, and the production $U ::= x$ is an element of $\mathcal{P}$.

y produces z (y 3 z) and conversely z reduces into y , if and only if there exist strings $x_0, \ldots, x_n$ such that $y = x_0$, $x_n = z$ and

$$x_{i-1} \rightarrow x_i \qquad (i = 1, \ldots, n; \quad n \geq 1)$$

z is also said to be a derivation of y .

Let $\mathcal{P}$ be a set of productions $\varphi_1, \ldots, \varphi_n$. If $\mathcal{V}$, the vocabulary, contains exactly one symbol A which occurs in no right part of a production, and a non empty set $\mathcal{B}$ of symbols which appear only in the right part of productions, then $\mathcal{P}$ is a phrase structure grammar. The symbols of $\mathcal{B}$ are called terminal or basic symbols and are denoted by capital letters T , T_1 , T_2 . The letters U , V always denote symbols in $\mathcal{V}-\mathcal{B}$ and are called <u>nonterminal</u> symbols.

$x \in \mathcal{V}^*$ is called a <u>sentential form</u> of $\mathcal{G}$ if either $A = x$ or $A \Rightarrow x$. The set of <u>sentences</u> x - i.e., the set of sentential forms consisting only of <u>terminal symbols</u> - constitutes the <u>phrase structure language</u> $L_{\mathcal{G}}$, that is:

(2.1) $$L_{\mathcal{G}} := \{x \mid A \Rightarrow x \wedge x \in \mathcal{B}^*\}$$

Without restricting the set of phrase structure languages we shall assume that

(2.2) $$U \not\Rightarrow U \quad \text{for any} \quad U \in \mathcal{V}-\mathcal{B} \; ;$$

(2.3) if $U_1 \Rightarrow U_k$, then the sequence

$$U_1 ::= U_2 \,,\; U_2 ::= U_3 \,,\; \ldots \; U_n ::= U_k$$

is unique; and

(2.4) every symbol may be used in deriving some sentence: for each symbol $X \in \mathcal{V}$ there exists strings x , z , t such that $A \Rightarrow xXz$. and either $X \in \mathcal{B}$ or $X \Rightarrow t$ where $t \in \mathcal{B}^*$.

A <u>parse</u> of the string z_0 into the symbol U is a sequence of productions $\varphi_1, \varphi_2, \ldots, \varphi_n$ such that $\varphi_j = (U_j ::= x_j)$ directly reduces $z_{j-1} = u_j x_j v_j$ into $z_j = u_j U_j v_j$ $(j = 1, \ldots, n)$ and $z_n = U$. The <u>canonical parse</u> is the parse which proceeds strictly from left to <u>right</u> in a sentence, and reduces a leftmost part of a sentence as far as possible before proceeding further to the right. That is, the parse $\varphi_1,\varphi_2,\ldots,\varphi_n$ of z_o into U is canonical if and only if for $j = 1,\ldots,n$ x_k is not contained in u_j for all $k > j$.

Every parse has a unique canonical form (simply rearrange the productions to form a canonical parse), but in an ambiguous grammar there exists more than one canonical parse for some sentence. An unambiguous grammar is a phrase structure grammar such that for every string $x \in L_{\mathcal{G}}$ there exists exactly-one canonical parse of x into the symbol A .

It has been shown that there exists no algorithm which decides whether an arbitrary grammar is unambiguous. However, a sufficient condition for a grammar to be unambiguous is subsequently derived, and a method is explained which determines whether a given grammar satisfies this condition.

3. Operator and augmented operator grammars and languages.

If no production φ_i of the phrase structure grammar $\mathcal{G}$ takes the form U ::= xV_1V_2y for some (possibly empty) strings x,y and nonterminal symbols V_1 'and V_2 , then (Floyd [4]) $\mathcal{G}$ is called an operator grammar (OG). The phrase structure language $L_{\mathcal{G}}$ generated by an OG is then called an operator language.

Floyd proved that in an operator grammar no sentential form contains two adjacent non-terminal symbols - i.e., if $A \Rightarrow x$ then there exists no strings x_1 and x_2 and no nonterminals V_1 and V_2 such that $x = x_1V_1V_2x_2$. The grammar in Figure 1 is not an operator grammar, since <IF CLAUSE> and <STATEMENT> are both nonterminal.

<PROG> ::= <STATEMENT>

<PROG> ::= <IF CLAUSE> <STATEMENT>

<IF CLAUSE> ::= IF <EXPRESSION> THEN

<STATEMENT> ::= <IF CLAUSE> <STATEMENT> ELSE <STATEMENT>

<STATEMENT> ::= VARIABLE := <EXPRESSION>

<EXPRESSION> ::= <EXPRESSION> OR VARIABLE

<EXPRESSION> ::= VARIABLE

Figure-1

The grammar in Figure 2, which is equivalent to (generates the same language as) the grammar in Figure 1, <u>is</u> an operator grammar.

<PROG> ::= <STATEMENT>

<PROG> ::= IF <EXPRESSION> THEN <STATEMENT>

<STATEMENT> ::= IF <EXPRESSION> THEN <STATEMENT> ELSE <STATEMENT>

<STATEMENT> ::= VARIABLE := <EXPRESSION>

<EXPRESSION> ::= <EXPRESSION> OR VARIABLE

<EXPRESSION> ::= VARIABLE

NONTERMINAL SYMBOLS: <PROG> , <STATEMENT> , <EXPRESSION> .

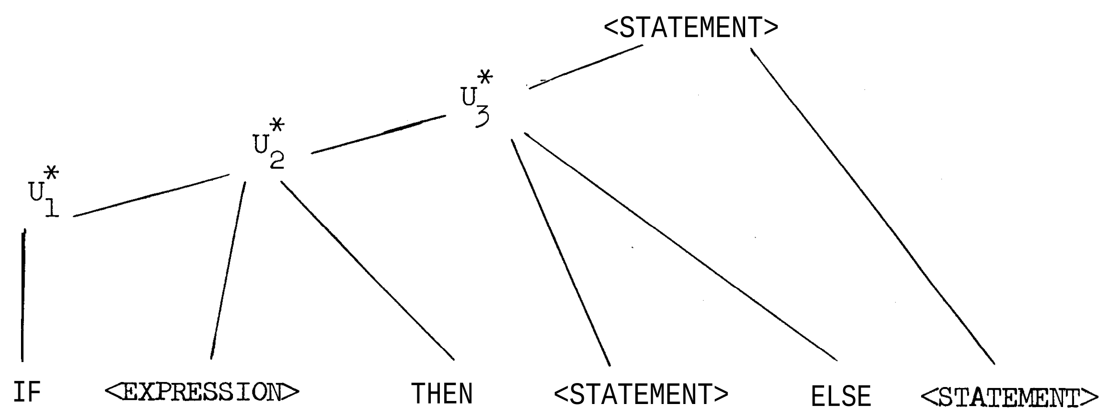
TERMINAL SYMBOLS: IF , THEN , ELSE , VARIABLE , := , OR

Figure 2

When parsing a sentence, at each step the leftmost right part x of a production U ::= x must be detected. Then x is replaced by U and the process is repeated. In order to reduce the number of symbols to be checked at each step, we introduce <u>intermediate reductions</u>. For instance, although the string

IF <EXPRESSION> THEN <STATEMENT> ELSE <STATEMENT>

can be reduced directly to <STATEMENT> by the grammar of Figure 2, we want to parse it as is shown below:



This can be achieved by constructing an augmented operator grammar (AOG) $\mathcal{G}_A$ corresponding to $\mathcal{G}$. The augmented operator grammar is useful for describing theoretically the mechanism of the matrix technique to be introduced later. However, so as not to complicate the process too much, one can give mnemonic names-to the introduced symbols needed for the intermediate reductions. For instance, in the above diagram U_1^* may be named "<IF*>" , U_2^* "<IF expr THEN*>" , and U_3^* "<IF expr THEN state ELSE*>" . That is, each new symbol is just a representation of the head of the right part of some production. $\mathcal{G}_A$ is constructed from $\mathcal{G}$ by repeating the following step 1 until no longer applicable, then step 2 until no longer applicable, and finally steps 3a and 3b alternately until no longer applicable. New nonterminal symbols will be introduced into V and V-B (but not B). Note that all newly introduced symbols are distinguished from the original nonterminals by an asterisk "*" .

step 1: If there is a production $U_1 ::= T_2 y_1$ (y_1 may be empty), and if k new symbols $U_1^*,\ldots,U_k^*$ have been created so far, create a new symbol U_{k+1}^*, replace each production $U_i ::= T_2 y_i$ (each production whose right part begins with T_2) by the production $U_i ::= U_{k+1}^* y_i$, and insert the production $U_{k+1}^* ::= T_2$ into the grammar,

After step 1 all productions have one of the forms

$$U_1 ::= U_2 \ , \ U_1 ::= U_2 T y \ , \ U_1 ::= \overset{*}{U} y \ , \ \overset{*}{U} ::= T .$$

where y contains no introduced symbol U^* .

step 2: If there is a production $U_1 ::= U_2 T_2 y_1$ (note that U_2 must be one of the original nonterminals of the OG), and if k new symbols have been created so far, create a new symbol U_{k+1}^* , replace each production. $U_i ::= U_2 T_2 y_i$ (each production whose right part begins with $U_2 T_2$) by $U_i ::= U_{k+1}^* y_i$, and insert the production $U_{k+1}^* ::= U_2 T_2$.

After step 2 all productions have one of the forms

$$U_1 ::= U_2 \, , U_1 ::= U^* y \ , \ U^* ::= UT \ , \ U^* ::= T$$

where y contains no introduced symbol U^* .

step 3a: If there is a production $U_1 ::= U^* T_2 y$, and if k new symbols have been created so far, create a new symbol U^*_{k+1} , replace each production $U_1 ::= U^*_2 T_2 y_i$ by $U_i ::= U^*_{k+1} y_i$, and insert the new production $U^*_{k+1} ::= U^*_2 T_2$.

step 3b: If there is a production $U_1 ::= U^*_2 U_2 T_2 y$, and if k new symbols have been created so far, create a new symbol U^*_{k+1} , replace each production $U_1 ::= U^*_2 U_2 T_2 y_i$ by $U_i ::= U^*_{k+1} y_i$, and insert the new production $U^*_{k+1} ::= U^*_2 U_2 T_2$.

An AOG has, therefore, only productions of one of the following forms:

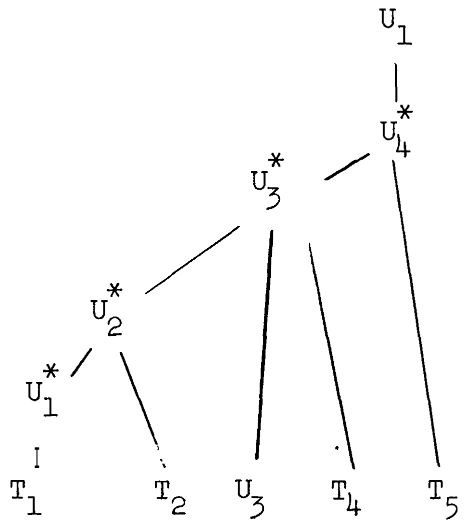
$$U_1 ::= U_2 \ , \ U_1 ::= U^* \ , \ U_1 ::= U^* U_2$$
$$U^* ::= T \ , \ U^* ::= UT \ , \ U^*_2 ::= U^*_1 T \ , \ U^*_2 ::= U^*_1 UT$$

Figure 3

Note that we differentiate between the original Unstarred NonTerminal Symbols (called UNTS) and the newly created Starred NonTerminal Symbols' (SNTS), which for reasons explained later are also called stack nonterminals.

Again, we have introduced augmented operator grammars and stack nonterminals in order to be able to make intermediate reductions. A stack non-terminal can also be thought of as a representation for the head xT (ending in a terminal symbol T) of the right part of a production U ::= xTy of the original OG. As another example, if $U_1 ::= T_1 T_2 U_3 T_4 T_5$ is a production of the OG, then the string $T_1 T_2 U_3 T_4 T_5$

will be parsed as a sentence of the AOG as follows:



<u>Productions in AOG</u>

$U_1^* ::= T_1$

$U_2^* ::= U_1^* T_1$

$U_3^* ::= U_2^* U_3 T_4$

$U_4^* ::= U_3^* T_5$

$U_1 ::= U_4^*$

Consider the OG in Figure 2. After step 1 it will have been changed to

<PROG> ::= <STATEMENT>

<IF*> ::= IF

<VARIABLE*> ::= VARIABLE

<PROG> ::= <IF*> <EXPRESSION> THEN <STATEMENT>

<STATEMENT> ::= <IF*> <EXPRESSION> THEN <STATEMENT> ELSE <STATEMENT>

<STATEMENT> ::= <VARIABLE*> := <EXPRESSION>

<EXPRESSION> ::= <EXPRESSION> OR VARIABLE

<EXPRESSION> ::= <VARIABLE*>

Step 2 changes it to

```
<PROG>        ::= <STATEMENT>
<IF*>         ::= IF
<VARIABLE*>   ::= VARIABLE
<EXPR-OR*>    ::= <EXPRESSION> OR
<PROG>        ::= <IF*> <EXPRESSION> THEN <STATEMENT>
<STATEMENT>   ::= <IF*> <EXPRESSION> THEN <STATEMENT> ELSE <STATEMENT>
<STATEMENT>   ::= <VARIABLE*> := <EXPRESSION>
<EXPRESSION>  ::= <EXPR-OR*> VARIABLE
<EXPRESSION>  ::= <VARIABLE*>
```

Finally, after step 3 we have the AOG of Figure 4:

```
<PROG>          ::= <STATEMENT>
<IF*>           ::= IF
<VARIABLE*>     ::= VARIABLE
<EXPR-OR*>      ::= <EXPRESSION> OR
<IF-THEN*>      ::= <IF*> <EXPRESSION> THEN
<IF-ELSE*>      ::= <IF-THEN*> <STATEMENT> ELSE
<VAR-:=*>       ::= <VARIABLE*> :=
<EXPR-OR-VAR*>  ::= <EXPR-OR*> VARIABLE
<PROG>          ::= <IF-THEN*> <STATEMENT>
<STATEMENT>     ::= <IF-ELSE*> <STATEMENT>
<STATEMENT>     ::= <VAR-:=*> <EXPRESSION>
<EXPRESSION>    ::= <EXPR-OR-VAR*>
<EXPRESSION>    ::= <VARIABLE*>
```

Figure 4

We will need the following definition: A string y is a phrase if

(3.1) y contains at least one terminal or SNTS; and

(3.2) there exists a production U ::= y_1 or U^* ::= y_1 of the ACG where $y_1 = y$ or $y_1 = uU_1v$, $y = uU_2v$ and $U_1 \Rightarrow U_2$, for some u , v .

Thus, y is a phrase if it is the right part of some production of the AOG (except a production of the form U_1 ::= U_2), or if it can be reduced to the right part of some production by a sequence of reductions U_i ::= U_j . Given a sentential form $x = x_1yx_2$, y is called a reducible phrase (of x), if

(3.3) y is a phrase; and

(3.4) for some U (or U^*) as defined in (3.2), the string resulting from replacing the string y by U (or U*) is a sentential form.

The problem for the compiler, then, is to find the leftmost reducible phrase and to make the correct replacement (reduction).

The following statements, which help to explain the relationship between an OG and the corresponding AOG, follow directly from the construction of the AOG. For lack of a better name, we call them lemmas.

Lemma 1. Each SNTS U^* appears as the left part of only one production U^* ::= x . The corresponding right part x appears as the right part of no other production.

Lemma 2. If the SNTS's U^* are numbered in the order in which they were introduced, $U_1^*, U_2^*, \ldots, U_n^*$, and if a production $U_i^* ::= U_j^* y$ exists in the AOG, then $i > j$.

Lemma 3. For each production $U ::= y$ with $y \notin V\text{-}B$ of the AOG there exists a unique set of productions $U_1^* ::= y_1$, $U_2^* ::= U_1^* y_2, \ldots,$ $U_n^* ::= U_{n-1}^* y_{n-1}$, $u ::= U_n^* y_n$ of the AOG such that $y = y_1 y_2 \ldots y_n$.

Lemma 3 follows directly from the construction and Lemmas 1 and 2. Lemma 4 follows directly from Lemma 3.

Lemma 4. If $\varphi_1, \varphi_2, \ldots$ [illegible] is a canonical parse of a string x relative to the OG, then we get a parse of x relative to the AOG by substituting for each φ_i (which is not of the form $U_1 ::= U_2$) the unique set of productions defined in Lemma 3.

Since two different canonical parses $\{\varphi_n\}$ and $\{\lambda_m\}$ of a string must for some i have $\varphi_i \neq \lambda_i$, we have also

Lemma 5. Different canonical parses of a string x with respect to an OG, yield different canonical parses of x with respect to the AOG.

Therefore we have finally

Lemma 6. If an AOG is unambiguous, the corresponding OG must also be unambiguous.

A sufficient condition for an OG to be unambiguous is therefore the unambiguousness of the corresponding AOG.

4. Parsing a string using an AOG.

In order to parse a sentence x we first enclose x in symbols Φ^* and Φ (where Φ^* is assumed to be a new SNTS and Φ a new terminal symbol), yielding $\Phi^* x \Phi$. Formally, we add to the AOG the productions <Program> ::= $\Phi^* A \Phi$ and Φ^* ::= Φ , where A is the symbol which appeared only in a left part. We show that after each reduction the string has one of the following two forms

(4.1) $\quad U_1^* U_2^* \dots U_{\ell\,1}^* U_\ell^* T_1 T_2 \dots T_m$

or

(4.2) $\quad U_1^* U_2^* \dots U_{\ell\,1}^* U_\ell^* U_1 T_1 \dots T_m$,

where the U_i^* are SNTS's, the T_i are terminals and U_1 is an UNTS. Note that the original string $\Phi^* x \Phi$ has form (4.1). We assume also that no reduction U ::= U_i^* , $i < \ell$, can be made such that the resulting string is still a sentential form, since such reductions will have already been made. It will be seen later that the sufficient conditions for a unique canonical parse fulfill this requirement.

From the form of productions in an AOG (Figure 3), any reducible phrase containing U_i , $i < \ell$, must be U_i itself, and this by assumption is not the case. Now at some point of the parse a reducible phrase must contain T_1 , and from the form of productions in an AOG, T_1 must be the last character of any reducible phrase containing it. Therefore, at this step, the leftmost reducible phrase may not contain $T_2, T_3, \dots, T_m$. We have therefore, using again the possible forms of productions in an AOG, Lemma 7. A leftmost reducible phrase at step k, assuming the string is of form (4.1), must be either

$$U_\ell^* \text{ , } T_1 \text{ , or } U_\ell^* T_1 \text{ .}$$

Assuming a string of form (4.2) the leftmost reducible phrase is

$$U_\ell^* U_1 \text{ , } U_\ell^* U_1 T_1 \text{ , or } U_1 T_1$$ [1]

In the case (4.1), if we know which of the strings U_ℓ^* , T_1 or $U_\ell^* T_1$ is the leftmost reducible phrase, we make a reduction $U ::= U_a^*$, $U_\ell^* ::= T_1$ resp. $U^* ::= U_\ell^* T_1$, yielding again a string of the form (4.1) or (4.2).

In case (4.2) we first make a sequence of reductions $U_2 ::= U_1 , \ldots , U_j ::= U_{j-1}$ for some $j \geq 1$, and then execute a final reduction $U ::= U_\ell^* U_j$, $U^* ::= {}^*U_\ell U_j T_1$, or $\hat{U} ::= U_j T_1$, depending on which of the three possibilities is the leftmost reducible phrase.

Note that at each step not only the leftmost reducible phrase, but also the sequence of reductions to be made, should be unique. If this is the case, then of course there exists a unique canonical parse. The next section-gives sufficient conditions for the uniqueness of the canonical parse. The reason for calling the U^* "stack nonterminal symbols" is now clear. They are the only symbols which get pushed into the stack.

---

[1] Note that a reduction $U ::= U_\ell^*$ or $U^* ::= T_1$ at this time would result in a string which is not a sentential form, since the grammar is an AOG (two original nonterminals of the OG would eventually appear adjacent).

5. Sufficient conditions for a unique canonical parse.

We will use the following set L(S) where S is a symbol:

$$\mathcal{L}(S) = \{S_1 | xS_1Sy \text{ is a sentential form for some } x,y\} .$$

$\mathcal{L}(S)$ is just the set of symbols which are adjacent and to the left of S_1 in some sentential form. The construction of $\mathcal{L}$ or related sets has been discussed elsewhere (see for instance Wirth and Weber [10]). We therefore do not wish to discuss at length the construction of L(S). We just state that

$$S_1 \in \mathcal{L}(S) \text{ if and only if } (S_1 \doteq S \vee S_1 \lessdot S \vee S_1 \gtrdot S) ,$$

where $\doteq$, $\lessdot$ and $\gtrdot$ are the precedence relations defined by Wirth and Weber (page 18, [10]).

Now consider case (4.1). We have a sentential form $U_1^* U_2^* \ldots U_\ell^* \; T_1 T_2 \ldots T_m$. If U_ℓ^* is a leftmost reducible phrase, then obviously

(4.3) $\exists$ a production $U ::= U_\ell^*$ such that $U \in \mathcal{L}(T_1)$.

Similarly, if $U_\ell^* T_1$ or T_1 is a leftmost reducible phrase, we have respectively

(4.4) $\exists$ a production $U^* ::= U_\ell^* T_1$;

(4.5) $\exists$ a production $U^* ::= T_1$ such that $U_\ell^* \in \mathcal{L}(U^*)$.

Consider case (4.2). We have a sentential form $U_1^* \ldots U_\ell^* U_1 T_1 \ldots T_m$. Depending on whether $U_\ell^* U_1$, $U_\ell^* U_1 T$, or $U_1 T_1$ is a leftmost reducible phrase, we have respectively

(4.6) 3 a production $U ::= U_a^* U_2$ where $U \in \mathcal{L}(T_1)$, and either
$U_2 = U_1$ or $U_2 \Rightarrow U_1$;

(4.7) 3 a production $U^* ::= U_\ell^* U_2 T_1$ where $U_2 = U_1$ or $U_2 \Rightarrow U_1$;

(4.8) 3 a production $U^* ::= U_2 T_1$ where $U_\ell^* \in \mathcal{L}(U^*)$, and either
$U_2 = U_1$ or $U_2 \Rightarrow U_1$.

We may now state the main

Result. Let U_ℓ^* be a SNTS, U_1 a UNTS and T_1 a terminal symbol. Assume that for any such $^*U_\ell$, U_1 and T_1

(a) At most one of the conditions (4.3), (4.4), (4.5) holds;

(b) At most one of the conditions (4.6),(4.7),(4.8) holds;

(c) If one of the conditions (4.3) - (4.8) holds, the production described therein is unique.

Then there exists a unique canonical parse for each sentence x of the language.

The result follows from the fact that at each step the leftmost reducible phrase is unique (from (a) and (b)), and the corresponding reduction or set of reductions is unique (remember the restriction on an OG that if $U_j \Rightarrow U_i$ then the reductions $U_j ::= U_{j+1}, \ldots, U_{i-1} ::= U_i$ are unique).

The algorithm which generates the "compiler is then straightforward. We first check that if $U_1 \Rightarrow U_j$, the sequence of productions $U_1 ::= U_2, \ldots, U_{j-1} ::= U_j$ is unique. Next the AOG is constructed.

$\mathfrak{L}(S)$ is then determined for all terminal and SNTS S . Then for each U^*_ℓ and T1 the productions are searched to see whether conditions (4.3), (4.4), or (4.5) hold, and if so, the production number (or just the left part) together with the reducible phrase is recorded. If for some U^*_ℓ and T_1 two different reductions are found to be possible, then some sentence may not be parsed unambiguously with the technique given in the next section. Note that this does not mean that the grammar is unambiguous ; it just has not satisfied our sufficiency conditions. Triples U^*_ℓ , Ul and T_1 are handled similarly.

6. <u>The transition matrix and stack</u>.

We have seen that, with sufficient restrictions on the grammar, at each step in the parsing of a sentence according to the augmented operator grammar AOG, the partially reduced string has the form

(6.1) $\quad U^*_1 U_2 \ . \ . \ . \ U^*_\ell T_1 \cdots T_m$

or

(6.2) $\quad U^*_1 U^*_2 \ . \ . \ . \ U^*_\ell U_1 T_1 \cdots T_m$

where U_1 is a non-terminal symbol of the original OG, T_i are terminal symbols, and U^*_j are SNTs of the AOG, or can be thought of as

(6.3) a representation for the head xT (ending in a terminal symbol T) of the right part of a production U ::= xTy of the original OG.

Furthermore, in the case (6.1) the leftmost reducible phrase, either

U_ℓ^*, T_1 or $U_\ell^* T_1$ is uniquely determined by U_ℓ^* and T_1. In the case (6.2), the leftmost reducible phrase is uniquely determined by U_ℓ^*, T_1 and U_1 and is either $U_\ell^* U_1$, $U_1 T_1$ or $U_\ell^* U_1^* T_1$.

In order to parse a sentence as quickly as possible, we construct a transition matrix B — a rectangular matrix B whose elements b_{ij} are numbers of subroutines. Each column j represents a terminal symbol T (or a class of terminal symbols — for instance, one column could represent the class <type> consisting of real, Boolean, and integer). For each stack symbol U_ℓ^* we designate two rows of the matrix — a basic row and a secondary row. Their uses are as follows:

Suppose at a step of the parse, the string has the form (6.1). Then the basic row ${}_bU_\ell^* = i$ corresponding to U_ℓ^* together with the column j representing T_1 determine an element b_{ij} of the matrix — the number of a subroutine which, when executed, will effect the unique reduction to be made.

If the string has form (6.2), the secondary row corresponding to U_ℓ^* together with the column representing T_1 determine an element of the matrix. When the corresponding subroutine is executed, U_1 will be checked and the appropriate unique reductions will be made.

For practical purposes we assume that the secondary row always follows the basic row. That is, if the basic row for U_ℓ^* is row number ${}_bU_\ell^*$, then the secondary row is number ${}_bU_\ell^* + 1$.

The pushdown (last-in-first-out) stack ST consists of elements, each consisting of two parts. If p is the pointer to the current top stack element, the two parts of the top stack element are labeled $ST1_p$ and $ST2_p$. The contents of each stack element are best illustrated by a diagram. If the string has the form (6.1) or (6.2), the stack configuration is as illustrated in Figure 5a or Figure 5b respectively, where again ${}_bU_i^*$ is the index or row number of the basic row corresponding to U_i^*.

(a) (ST1 , ST2)		(b) (ST1 , ST2)
$({}_bU_\ell^* , 0)$	top of stack	$({}_bU_\ell^* + 1 , U_1)$
$({}_bU_{\ell-1}^* + 1 , 0)$		$({}_bU_{\ell-1}^* + 1 , 0)$
$({}_bU_1^* + 1 , 0)$	bottom of stack	$({}_bU_1^* + 1 , 0)$

(a) (b)

Figure 5

Note that the first part ST1 of a stack element which is not the top stack element is always the index of a secondary row, since when it later becomes the top stack element, $ST2_p$ must contain a nonterminal symbol U . Later it will be shown how the second part of each element may be used systematically to hold semantic information.

To illustrate how efficient the technique is, we give an example of a typical implementation on the IBM 7090. Suppose that the stack element $ST1_p$ actually contains the instruction

TRA*{ADDRESS OF $B[{}_bU_\ell^*, 0]$},4 or TRA*(ADDRESS OF $B[{}_bU_\ell^* + 1, 0]$},4

that the matrix B is stored rowwise in memory, that each matrix element consists of a single location containing the address of the corresponding subroutine, that there exists a vector COLUMN to map a terminal symbol into the corresponding column number, and that the stack pointer p is in index register 1. The following sequence then determines the subroutine to be executed:

```
LXA  T1,2        PUT THE INCOMING SYMBOL IN INDEX REGISTER 2.
CLA  COLUMN,2    COLUMN NUMBER FOR T1 IN ACCUMULATOR.
PAC  0,4         COMPLEMENT OF COLUMN NUMBER IN XR4.
TRA  ST1,1       JUMP TO TOP STACK ELEMENT, WHICH IN TURN
                 WILL JUMP TO THE SUBROUTINE
```

As an example of a matrix and subroutines, consider first the grammar in Figure 6, which is the same as the grammar in Figure 2 (Section 3) except for the introduction of the production

<PROGRAM> ::= Φ <PROG> Φ.

Both the matrix in Figure 7 and the associated subroutines of Figure 8 were produced exactly as they appear from the grammar of Figure 6 by the algorithm programmed in Extended ALGOL [11] on the B5500 (except for the numbering of the rows of the matrix).

The SNTSs of the AOG do not appear in the matrix or subroutines; we have labeled the basicrows of the matrix with the head of the right part of the production which they represent. Correspondingly, for example, subroutine 400 of Figure 8 contains the instruction $ST1_p$ ← ROW(<EXPRESSION> OR), which means that $ST1_p$ is to have as its value the index of the basic row corresponding to the SNTS which represents <EXPRESSION> OR . This is row 13. As can be seen, the AOG is actually not necessary practically, but is a convenient theoretical tool.

The zero elements of the matrix represent incorrect pairs, while the other matrix elements are numbers of the subroutines listed in Figure 8.

The individual statements of the subroutines are separated by a slash "/" , The statement "NEW T1" means "SCAN", or use the symbol T_2 as the next incoming symbol T_1 . If the statement "NEW T1" is not executed, the old T_1 will be used again on the next cycle. After a'subroutine has been executed, the next cycle is performed.

Some of the subroutines test $ST2_P$ for the presence of a nonterminal symbol. If $ST2_P$ is one of the nonterminals listed, a corresponding subroutine is executed. If not, a syntactic error has occurred — the original string is not a sentence of the grammar.

Note also that if a reduction to some non-terminal U is made ($ST2_P \leftarrow U$), the original production of the operator grammar corresponding to this reduction is also listed for reference.

The following simplification has been made. Suppose that xT is the right part of only one production U ::= xT of the OG, and that there is no production U_1 ::= xTy for some nonempty y . The AOG will contain among others two productions- U^* ::= x_1T and U ::= U^* . Obviously this is not necessary. In order to save the intermediate step, the two productions are replaced by the single production U ::= x_1T .

PRODUCTIONS

1	<PROGRAM>	::=	PHI	<PROG>	PHI		
2	<PROG>	::=	<STATEMENT>				
3		::=	IF	<EXPRESSION>THEN		<STATEMENT>	
4	<STATEMENT>	::=	IF	<EXPRESSION>THEN		<STATEMENT> ELSE	<STATEMENT>
5		::=	VARIABLE	:=	<EXPRESSION>		
6	<EXPRESSION>	::=	<EXPRESSION>OR		VARIABLE		
7		::=	VARIABLE				

NONTERMINAL SYMBOLS

1	<PROGRAM>	2	<PROG>	3	<STATEMENT>	4	<EXPRESSION>

TERMINAL SYMBOLS

5	PHI	6	IF	7	THEN	8	ELSE
9	VARIABLE	10	:=	11	OR.		

MATRIX IS 14 ×

Figure 6

		PHI	IF	THEN	ELSE	VARIABLE	:=	OR
	PHI	0	1	0	0	1	0	0
2	SECONDARY ROW	3	0	0	0	0	0	0
3	IF	0	0	0	0	2	0	0
4	SECONDARY ROW	0	0	4	0	0	0	5
5	IF <EXPRESSION>							
	THEN	0	1	0	0	1	0	0
6	SECONDARY ROW	6	0	0	7	0	0	0
7	IF <EXPRESSION>							
	THEN <STATEMENT>		1					
	ELSE	0	0	0	0	1	0	0
8	SECONDARY ROW	8		0	8	0	0	0
9	VARIABLE	10	0	10	10	0	9	10
10	SECONDARY ROW	0	0	0	0	0	0	0
11	VARIABLE :=	0	0	0	0	2	0	0
12	SECONDARY ROW	11	0	0	10	0	0	5
13	<EXPRESSION>OR	0	0	0	0	12	0	0
14	SECONDARY ROW	0		0		0	0	0'

Figure 7

```
MATRIX SUBROUTINES, THOSE SUBS WHICH ARE ACTUALLY MATRIX ENTRIES HAVE NUMBERS LESS THAN 400

1    ST1P←ST1P+1  /  P←P+1  /  ST1P←ROW(T1 )  /  NEW T1
2    ST1P←ST1P+1  /  ST2P←<EXPRESSION>   FOR PROCUCTION *EXPRESSION> ::=  VARIABLE   /  NEW T1
3      USE SUB 401 IF ST2P IS <PROG>   <STATEMENT>
4      USE SUB 402 IF ST2P IS <EXPRESSION>
5      USE SUB 400 IF ST2P IS <EXPRESSION>
6      USE SUB 403 IF ST2P IS <STATEMENT>
7      USE SUE 434 IF ST2P IS <STATEMENT>
8      USE SUB 405 IF ST2P IS <STATEMENT>
9    ST1P←ROW(  VARIABLE   :=      )   /  NEW  T1
10   P←P-1  /  ST2P←<EXPRESSION>   FOR PRODUCTION <EXPRESSION> ::=  VARIABLE
11     USE SUE?406 IF ST2P IS <EXPRESSION>
12     P←P-1 /  ST2P←<EXPRESSION>   FORPRODUCTION <EXPRESSION> ::=  <EXPRESSION>OR    VARIABLE .   /  NEW T1

400  P←P+1  /  ST1P←ROW(  <EXPRESSION>OR     )   /  NEW  T1
401  P←P-1  /  ST2P←<PROGRAM>    FOR PRODUCTION <PROGRAM>   ::=  PHI    <PROG>    PHI    /  NEW T1

402  ST1P←ROW( I  F     <EXPRESSION>THEN     )   / NEW   Tl
403  P←P-1  /  ST2P←<PROG>     FOR PRODUCTION <PROG>   ::=  IF    <EXPRESSION>THEN    <STATEMENT>

404  ST1P←ROW(  IF     <EXPRESSION>THEN     <STATEMENT> ELSE     )   / NEW   Tl
405    P←P-1 /  ST2P←<STATEMENT>   FOR PRODUCTION <STATEMENT>  ::=  If    <EXPRESSION>THEN    <STATEMENT>
     ELSE     <STATEMENT>
406  P←P-1     ST2P←<STATEMENT>   FOR PRODUCT ION<STATEMENT>  ::=  VARIABLE     <EXPRESSION>
```

Figure 8

7. <u>An example of a parse</u>.

Let us parse the sentence

$$\Phi \text{ IF VARIABLE THEN VARIABLE := VARIABLE } \Phi$$

of the OG in Figure 6. We start with the following configuration:

Cycle	P	STACK	T_1	Rest of string
1	1	([row] 1 [PHI], 0)		IF VARIABLE THEN VARIABLE := VARIABLE PHI .

The row labeled PHI in Figure 7 and T_1 = "IF" determine subroutine 1 of Figure 8. Execution of the first statement "$ST1_p \leftarrow ST1_p + 1$" of subroutine 1 changes $ST1_1$ to [row]2. The stack pointer is then increased by 1 and the index of the row corresponding to "IF" (since T_1 = "IF"), which is 3, is put in $ST1_2$. "VARIABLE" is then scanned, yielding

Cycle	P	STACK	T_1	Rest of string
2	2	([row]3, 0)	VARIABLE	THEN VARIABLE := VARIABLE PHI
		([row]2, 0)		

Row 3 and "VARIABLE" now determine subroutine number 2. Here we change $ST1_2$ to [row]4, put "<EXPRESSION>" in $ST2_2$, and indicate that the next symbol, "THEN", is to be scanned. This yields

Cycle	P	STACK	T_1	Rest of string
3	2	(4, <EXPRESSION>)	THEN	VARIABLE := VARIABLE PHI
		(2, 0)		

Row 4 and "THEN" lead to subroutine number 4. There, $ST2_2$ is checked for "<EXPRESSION>" . Since it is correct, subroutine 402 is executed yielding

Cycle	P	STACK	T_1	Rest of string	subroutine to execute
4	2	(5, 0)	VARIABLE	:= VARIABLE PHI	1
		(2, 0)			

Continuing in this manner gives the following configurations at the beginning of each cycle.

Cycle	P	STACK	T_1	Rest of string	subroutine to execute
5	3	(9, 0)	:=	VARIABLE PHI	9
		(6, 0)			
		(2, 0)			
6	3	(11, 0)	VARIABLE	PHI	2
		(6, 0)			
		(2, 0)			
7	3	(12, <EXPRESSION>)	PHI		11
		(6, 0)			
		(2, 0)			
8	2	(6, <STATEMENT>)	PHI		6
		(2, 0)			
9	1	(2, <PROG>)	PHI		3(STOP)

8. Representation of non-terminals in the stack.

Strictly speaking, one should insert the nonterminal symbol U itself into $ST2_p$. This is however neither practical nor necessary. In practice, nonterminals fall into classes whose elements are the same semantically. For instance, in ALGOL the nonterminals <primary> , <factor> , <term> , <simple arith expr> are introduced only to help define the precedence of operations. In a compiler, they would all be represented by an address specifying a location which gives the type, location of the value during execution time (accumulate, register, storage location), etc. The determination of which U is actually in $ST2_p$ turns out to be almost always a semantic evaluation, which would have to be done anyway. There is therefore very rarely any list searching to determine which U is in $ST2_p$, but just a semantic evaluation of $ST2_p$. Accordingly, a reduction U ::= x is accomplished by inserting into $ST2_p$ the semantic meaning of the symbol U and not U itself. Notice that we assume in the discussion of the method that productions U_i ::= U_j have no "interpretation rule" associated with them, which is usually the case.

Note also that the part $ST2_p$ of the elements p = 1, ... n-l may also be used systematically to store semantic information. If we formally parse the ALGOL statement BEGIN A := (B+E)+C*D END there will be in the stack at some time the elements

$$({}_b\langle\text{term}*\rangle + 1 \text{ , identifier})$$

$$({}_b\langle\text{expr} +\rangle +1 \text{ , } 0)$$

$$({}_b\langle\text{var} :=\rangle + 1 \text{ , } 0)$$

$$({}_b\langle \mathrm{BEGIN}\rangle + 1\ ,\ 0)$$
$$({}_b\langle\ \Phi\ \rangle + 1\ ,\ 0)$$

We can, though, use the second part of each stack element to contain semantic information:

$$({}_b\langle \mathrm{term}*\rangle + 1\ ,\ \text{(semantics of D)})$$
$$({}_b\langle \mathrm{expr}\ +\rangle + 1\ ,\ \text{(semantics of C)})$$
$$({}_b\langle \mathrm{VAR} := \rangle + 1\ ,\ \text{(semantics of B+E)})$$
$$({}_b\langle \mathrm{BEGIN}\rangle + 1\ ,\ \text{(semantics of A)})$$
$$({}_b\langle\ \Phi\ \rangle + 1\ ,\ \text{(any necessary information)})$$

9. Other uses of transition matrices.

Two other uses will be introduced here, both concerned with optimizing the calculation of addresses of subscripted variables within FOR-loops ([6],[9],[5]).

Provided that a FOR-loop meets certain conditions, calculation of the address of a subscripted variable $A[E_1, \ldots, E_n]$ occurring in the statement of the FOR-loop may be optimized if the E_i satisfy certain restrictions, some of which we list here:

1. E_i is <u>linear</u> in the loop variable of the FOR-loop, $i = 1, \ldots, n$. That is, E_i may be put in the form $C_1 * I + C_2$, where C_1 and C_2 are expressions not containing the loop variable I.

2. E_i contains only simple integer variables, integer constants, parentheses (and), and the operators + , - , and *.

3. The variables appearing in the E_i do not change within the <u>FOR</u>-loop statement.

Restrictions 1 and 2 may be checked systematically using the (operator) grammar in Figure 9. If $A[E_1, \ldots, En]$ is the subscripted variable and <CONST EL> $\Rightarrow$ $[E_1, \ldots, E_n]$, then $A[E_1, \ldots, E_n]$ satisfies restrictions 1 and 2 and moreover no E_i contains the loop variable. If <LIN EL> $\Rightarrow$ $[E_1, \ldots, En]$, then similarly $A[E_1, \ldots, E_n]$ satisfies restrictions 1 and 2 but at least one E_i contains the loop variable.

From the grammar we generate the <u>optimizable subscript checker</u> — the transition matrix and subroutines in Figure 10. Note that column 1 of the matrix contains only zeroes. We may map all terminal symbols except for the ones listed in restriction 2 into column 1. If, when parsing a subscripted variable according to the grammar in Figure 9, an error occurs, then this subscripted variable is handled in the usual way. Otherwise, it may be possible to optimize here and the variables occurring in the E_i should be stored in some list for further checking.

As a second example we look at the <u>FOR</u>-loop itself. We want it to have the form

$$\underline{\text{FOR}}\ I \leftarrow E_1\ \underline{\text{STEP}}\ E_2\ \underline{\text{UNTIL}}\ E_3\ \underline{\text{DO}}\ S\ ;$$

where the variables in E_2 do not change within the statement S, E_2 does not contain the loop variable I, and the E_i are integer expressions. The further restriction is again made, that the E_i consist only of integer simple variables, integer constants, (,), +, -, and *. Note that E_1 and E_3 may contain the loop variable I, but E_2 may not. The variables in E_2 should be listed for further checking.

```
PRODUCTIONS

 1   <CONST ELEM> ::=   [                <CONST SUBS> ]
 2   <LIN ELEM>   ::=   [                <LIN SUBS>   ]
 3   <CONST SUBS> ::=   <CONST EXPR>
 4                ::=   <CONST SUBS> ,                 <CONST EXPR>
 5   <LIN SUBS>   ::=   <LIN EXPR>
 6                ::=   <LIN SUBS> ,                   <LIN EXPR>
 7                ::=   <LIN SUBS> ,                   <CONST EXPR>
 8                ::=   <CONST SUBS> ,                 <LIN EXPR>
 9   <CONST EXPR> ::=   <CONST EXPR> <+ OR -> <CONST TERM>
10                ::=   <+ OR ->         <CONST TERM>
11                ::=   <CONST TERM>
12   <CONST TERM> ::=   <CONST TERM> *                 <CONST FACT>
13                ::=   <CONST FACT,>
14   <CONST FACT> ::=   (                <CONST EXPR> )
15                ::=   INTEGER
16                ::=   INTEGER VAR
17   <LIN EXPR>   ::=   <LIN EXPR>       <+ OR ->      <LIN TERM>
18                ::=   <CONST EXPR> <+ OR ->          <LIN TERM>
19                ::=   <LIN EXPR>       <+ OR ->      <CONST TERM>
20                ::=   <+ OR ->         <LIN TERM>
21                ::=   <LIN TERM>
22   <LIN TERM>   ::=   <CONST TERM> *                 <LIN FACT>
23                ::=   <LIN TERM> *                   <CONST FACT,
24                ::=   <LIN FACT>
25   <LIN FACT>   ::=   (                <LIN EXPR> )
26                ::=   LOOP VAR

NONTERMINAL SYMBOLS
 1   <CONST ELEM>      2   <LIN ELEM>        3   <CONST SUBS>      4   <LIN SUBS>
 5   <CONST EXPR>      6   <CONST TERM>      7   <CONST FACT>      8   <LIN EXPR>
 9   <LIN TERM>       10   <LIN FACT>

TERMINAL SYMBOLS
11   [                12   ]                13   ,                14   <+ OR ->
15   *                16   (                17   )                18   INTEGER
19   INTEGER VAR      20   LOOP VAR

MATRIX IS 18 x 10
```

Figure 9

	[	]	,	<+ OR ->	*	(	)	INTEGER	INTEGER VAR	LOOP VAR
[	0	0	0	1	0	1	0	2	3	4
SECONDARY ROW	0	5	6	7	8	0	0	0	0	0
<CONST SUBS>,	0	0	0	1	0	1	0	2	3	4
SECONDARY ROW	0	9	9	7	8	0	0	0	0	0
<LIN SUBS> ,	0	0	0	1	0	1	0	2	3	4
SECONDARY ROW	0	10	10	7	8	0	0	0	0	0
<CONST EXPR><+ OR ->	0	0	0	0	0	1	0	2	3	4
SECONDARY ROW	0	11	11	11	8	0	11	0	0	0
<+ OR ->	0	0	0	0	0	1	0	2	[illegible]	4
SECONDARY ROW	0	12	12	12	8	0	12	0	0	0
<CONST TERM>*	0	0	0	0	0	1	0	2	3	4
SECONDARY ROW	0	13	13	13	13	0	13	0	0	0
(	0	0	0	1	0	1	0	2	3	4
SECONDARY ROW	0	0	0	7	8	0	14	0	0	0
<LIN EXPR> <+ OR ->	0	0	0	0	0	1	0	2	3	4
SECONDARY ROW	0	15	15	15	8	0	15	0	0	0
<LIN TERM> *	0	0	0	0	0	1	0	2	3	0
SECONDARY ROW	0	16	16	16	16	0	16	0	0	0

Figure 10

MATRIX SUBROUTINES, THOSE SUBS WHICH ARE ACTUALLY MATRIX ENTRIES HAVE NUMBERS LESS THAN 000

```
 1  ST1P←ST1P+1  /   P←P+1  /   ST1P←ROW(T1)  /  NEW  T1
 2  ST1P←ST1P+1  /   ST2P←<CONST FACT>   FOR PRODUCTION <CONST FACT>::=  INTEGER       /  NEW  T1
 3  ST1P←ST1P+1  /   ST2P←<CONST FACT>   FOR PRODUCTION <CONST FACT>::=  INTEGER VAR   /  NEW  T1
 4  ST1P←ST1P+1  /   ST2P←<LIN FACT>     FOR PRODUCTION <LIN FACT>  ::=  LOOP VAR      /  NEW  T1
 5 EITHER
      USE SUB 406 IF ST2P IS <CONST SUBS><CONST EXPR><CONST TERM><CONST FACT>
      USE SUB 407 IF STOP IS <LIN SUBS><LIN EXPR><LIN TERM><LIN FACT>
 6 EITHER
      USE SUB 400 IF ST2P IS <CONST SUBS><CONST EXPR><CONST TERM><CONST FACT>
      USE SUB 403 IF ST2P IS <LIN SUBS><LIN EXPR>  <LIN TERM>  <LIN FACT>
 7 EITHER
      USE SUB 401 IF ST2P IS <CONST EXPR><CONST TERM><CONST FACT>
      USE SUB 434 IF ST2P IS <LIN EXPR>  <LIN TERM>  <LIN FACT>
 8 EITHER
      USE SUB 402 IF ST2P IS <CONST TERM><CONST FACT>
      USE SUB 405 IF ST2P IS <LIN TERM>  <LIN FACT>
 9 EITHER
      USE SUB 408 IF ST2P IS <CONST EXPR><CONST TERM><CONST FACT>
      USE SUB 439 IF ST2P IS <LIN EXPR>  <LIN TERM>  <LIN FACT>
10 EITHER
      USE SUB 411 IF ST2P IS <CONST EXPR><CONST TERM><CONST FACT>
      USE SUB 410 IF STOP IS <LIN EXPR>  <LIN TERM> <LIN FACT>
11 EITHER
      USE SUB 412 If ST2P IS <CONST TERM><CONST FACT>
      USE SUB 413 IF ST2P IS <LIN TERM>  <LIN FACT>
12 EITHER
      USE SUB 414 IF ST2P IS <CONST TERM><CONST FACT>
      USE SUB 415 IF ST2P IS <LIN TERM>  <LIN FACT>
13 EITHER
      USE SUB 416 IF ST2P IS <CONST FACT>
      USE SUB 417 IF STOP IS <LIN FACT>
14 EITHER
      USE SUB 418 IF ST2P IS <CONST EXPR><CONST TERM><CONST FACT>
      USE SUB 419 IF ST2P IS <LIN EXPR>  <LIN TERM*  <LIN FACT>
15 EITHER
      USE SUB 421 IF ST2P IS <CONST TERM><CONST FACT,
      USE SUB 420 IF ST2P IS <LIN TERM>  <LIN FACT>
```

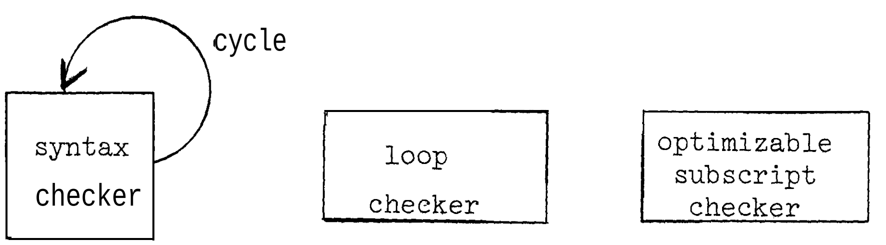
Figure 10 (continued)

```
16   USE SUB 422 IF ST2P IS <CONST FACT>
400  P←P+1  /  ST1P←ROW(  <CONST SUBS>,             )   / NEW T1
401  P←P+1  /  ST1P←ROW(  <CONST EXPR><+ OR ->      )   / NEW T1
402  P←P+1  /  ST1P←ROW(  <CONST TERM>*             )   / NEW T1
403  P←P+1  /  ST1P←ROW(  <LIN SUBS> ,              )   / NEW T1
404  P←P+1  /  ST1P←ROW(  <LIN EXPR>  <+ OR ->      )   / NEW T1
405  P←P+1  /  ST1P←ROW(  <LIN TERM>  *             )   / NEW T1
406  P←P-1  /  ST2P←<CONST ELEM>   FOR PRODUCTION <CONST ELEM> ::= [      <CONST SUBS>]          / NEW T1
407  P←P-1  /  ST2P←<LIN ELEM>     FOR PRODUCTION <LIN ELEM>   ::= [      <LIN SUBS> ]           / NEW T1
408  P←P-1  /  ST2P←<CONST SUBS,   FOR PRODUCTION <CONST SUBS> ::= <CONST SUBS>,      <CONST EXPR>
409  P←P-1  /  ST2P←<LIN SUBS>     FOR PRODUCTION <LIN SUBS>   ::= <CONST SUBS>,      <LIN EXPR>
410  P←P-1  /  ST2P←<LIN SUBS>     FOR PRODUCTION <LIN SUBS>   ::= <LIN SUBS> ,       <LIN EXPR>
411  P←P-1  /  ST2P←<LIN SUBS>     FOR PRODUCTION <LIN SUBS>   ::= <LIN SUBS> ,       <CONST EXPR>
412  P←P-1  /  ST2P←<CONST EXPR>   FOR PRODUCTION <CONST EXPR> ::= <CONST EXPR><+ OR ->  <CONST TERM>
413  P←P-1  /  ST2P←<LIN EXPR>     FOR PRODUCTION <LIN EXPR>   ::= <CONST EXPR><+ OR ->  <LIN TERM>
414  P←P-1  /  ST2P←<CONST EXPR>   FOR PRODUCTION <CONST EXPR> ::= <+ OR ->   <CONST TERM>
415  P←P-1  /  ST2P←<LIN EXPR>     FOR PRODUCTION <LIN EXPR>   ::= <+ OR ->   <LIN TERM>
416  P←P-1  /  ST2P←<CONST TERM>   FOR PRODUCTION <CONST TERM> ::= <CONST TERM>*      <CONST FACT>
417  P←P-1  /  ST2P←<LIN TERM>     FOR PRODUCTION <LIN TERM>   ::= <CONST TERM>*      <LIN FACT>
418  P←P-1  /  ST2P←<CONST FACT>   FOR PRODUCTION <CONST FACT> ::= (      <CONST EXPR>)          / NEW T1
419  P←P-1  /  ST2P←<LIN FACT>     FOR PRODUCTION <LIN FACT>   ::= (      <LIN EXPR> )           / NEW T1
420  P←P-1  /  ST2P←<LIN EXPR>     FOR PRODUCTION <LIN EXPR>   ::= <LIN EXPR>  <+ OR ->   <LIN TERM>
421  P←P-1  /  ST2P←<LIN EXPR>     FOR PRODUCTION <LIN EXPR>   ::= <LIN EXPR>  <+ OR ->   <CONST TERM>
422  P←P-1  /  ST2P←<LIN TERM>     FOR PRODUCTION <LIN TERM>   ::= <LIN TERM>  *          <CONST FACT>
```

Figure 10 (continued)

The grammar of Figure 11 is then constructed. The terminal "INT CONS, VAR" represents the class of integer constants and simple integer variables (except the loop variable of this loop). The corresponding transition matrix and subroutines are in Figure 12. We call this the loop checker. Since we are not interested in the precedence of the operators + , - , * , we have simplified the productions for arithmetic expressions. Note also in Figure 11 that " * " is used as a unary operator. Since we assume that the subscripts have been or are being checked for syntactical errors by another part of the compiler, this will never happen. Grammars should always be constructed according to what they will be used for, and should be as simple as possible.

One can incorporate the loop checker and optimizable subscript checker into an existing syntax checker as follows. All three are put in memory together. The main syntax checker executes as it normally does, performing the usual "cycles" described already.



When a FOR is scanned, the syntax checker activates the loop checker. Thereafter both process in parallel. The syntax checker processes one symbol and then passes it on to the loop checker, which when finished returns to the syntax checker to process the next symbol:

```
PROCUCTIONS

  1   <FOR LOOP>     ::=   FOR         LOOP   VAR   ←         <EXP2>     STEP     <EXP1>     UNTIL     <EXP2>
                           DO
  2   <EXP1>         ::=   <EXP1>      <+,-,x>      <EXP1 FACT>
  3                  ::=   <+,-,x>     <EXP1 FACT>
  4                  ::=   <EXP1 FACT>
  5   <EXP1 FACT>    ::=   (           <EXP1>       )
  6                  ::=   INT CONS,VAR
  7   <EXP2>         ::=   <EXP1>      <+,-,x>      <EXP2 FACT>
  8                  ::=   <EXP2>      <+,-,x>      <EXP1 FACT>
  9                  ::=   <EXP2>      <+,-,x>      <EXP2 FACT>
 10                  ::=   <+,-,x>     <EXP2 FACT>
 11                  ::=   <EXP2 FACT>
 12   <EXP2 FACT>    ::=   (           <EXP2>       )
 13                  ::=   LOOP VAR

NONTERMINAL SYMBOLS
  1   <FOR LOOP>       2   <EXP1>          3   <EXP1 FACT>      4   <EXP2>
  5   <EXP2 FACT>

TERMINAL SYMBOLS
  6   FOR              7   LOOP  VAR       8   ←                9   STEP
 10   UNTIL           11   DO             12   <+,-,x>         13   (
 14   )               15   INT  CONS,VAR

MATRIX IS 18 x 10
```

Figure 11

		FOR	LOOP VAR	←	STEP	UNTIL	DO	<+,-,x>	(	)	INT CONS, VAR
FOR		0	4	0	0	0	0	0	0	0	0
SECONDARY ROW		0	0	0	0	0	0	0	0	0	0
FOR	LOOP VAR	0	0	5	0	0	0	0	0	0	0
SECONDARY ROW		0	0	0	0	0	0	0	0	0	0
FOR	LOOP VAR										
←		0	3	0	0	0	0	1	1	0	2
SECONDARY ROW		0	0	0	6	0	0	7	0	0	0
FOR	LOOP VAR										
←	<EXP2>										
STEF		0	0	0	0	0	0	1	1	0	2
SECONDARY ROW		0	0	0	0	8	0	9	0	0	0
FOR	LOOP VAR										
←	<EXP2>										
STEF	<EXP1>										
UNTIL		0	3	0	0	0	0	1	1	0	2
SECONDARY ROW		0	0	0	0	0	10	7	0	0	0
<EXP1>	<+,-,x>	0	3	0	0	0	0	0	1	0	2
SECONDARY ROW		0	0	0	11	12	11	13	0	13	0
<+,-,x>		0	3	0	0	0	0	0	1	0	2
SECONDARY ROW		0	0	0	14	15	14	16	0	16	0
(		0	3	0	0	0	0	1	1	0	2
SECONDARY ROW		0	0	0	0	0	0	7	0	17	0
<EXP2>	<+,-,x>	0	3	0	0	0	0	0	1	0	2
SECONDARY ROW		0	0	0	18	0	18	18	0	18	0

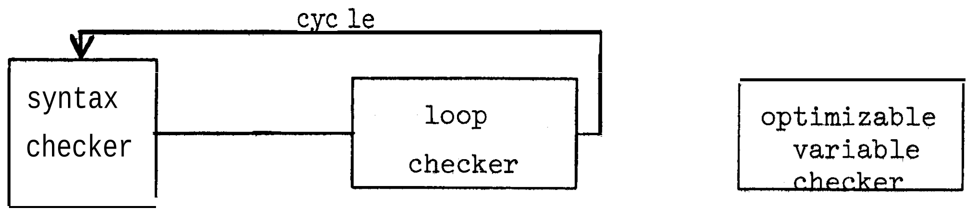
Figure 12

```
MATRIX SUBROUTINES, THOSE SUBS WHICH ARE ACTUALLY MATRIX ENTRIES HAVE NUMBERS LESS THAN 400

1     ST1P←ST1P+1 /    P←P+1   /    ST1P←ROW(T1)    /  NEW T1
2     ST1P←ST1P+1 /    ST2P←<EXP1 FACT>    FOR PRODUCTION <EXP1 FACT> ::=  INT  CONS,VAR  /  NEW T1
3     ST1P←ST1P+1 /    ST2P←<EXP2 FACT>    FOR PRODUCTION <EXP2 FACT> ::=  LOOP VAR        /  NEW T1
4   ST1P←ROW(  FOR          LOOP VAR     )       /  NEW  T1
5   ST1P←ROW(  FOR          LOOP   VAR   ←              )       /  NEW  TI
6     USE SUB 402 IF ST2P IS <EXP2>       <EXP2 FACT>
   EITHER
      USE SUB 400 IF STEP IS <EXP1>       <EXP1 FACT>
      USE SUB 401 IF ST2P IS <EXP2>       <EXP2 FACT>
0     USE SUB 403 IF ST2P IS <EXP1>       <EXP1 FACT>
9     USE SUB 400 IF ST2P IS <EXP1>       <EXP1 FACT>
10    USE SUB 434 IF ST2P IS <EXP2>       <EXP2 FACT>
11    USE SUB 436 IF ST2P IS <EXP2 FACT>
12    USE SUB 435 IF ST2P IS <EXP1 FACT>
13 EITHER
      USE SUB 435 IF ST2P IS <EXP1 FACT>
      USE SUB 406 IF STEP IS <EXP2 FACT>
14    USE SUB 408 IF ST2P IS <EXP2 FACT>
15    USE SUB 407 IF ST2P IS <EXP1 FACT>
16 EITHER
      USE SUB 407 IF ST2P IS <EXP1 FACT>
      USE SUB 408 IF ST2P IS <EXP2 FACT>
17 EITHER
      USE SUB 409 IF ST2P IS <EXP1>       <EXP1 FACT>
      USE SUB 410 IF ST2P IS <EXP2>       <EXP2 FACT>
18 EITHER
      USE SUB 411 IF ST2P IS <EXP1 FACT>
      USE SUB 412 IF ST2P IS <EXP2 FACT>
400 P←P+1   /   ST1P←ROW(  <EXP1>      <+,-,x>      )       /  NEW T1
401 P←P+1   /   ST1P←ROW(  <EXP2>      <+,-,x>      )       /  NEW T1
402 ST1P←ROW(  FOR          LOOP   VAR   ←         <EXP2>      STEP          )      /  NEW T1
```

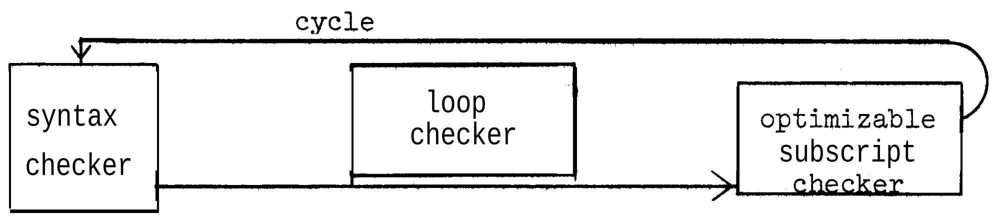
Figure 12 (continued)

```
403  ST1P←ROW(   FOR              LOOP VAR    ←          <EXP2>      STEP        <EXP1>      UNTIL                  /  NEW TI

404    P←P-1 /   ST2P←<FOR LOOP>     F O R PRODUCTION <FOR LOOP>   ::=  FOR         LOOP VAR    ←           <EXP2>
     STEP        <EXP1>      UNTIL       <EXP2>       DO                /  NEW T1

405  P←P-1   /   ST2P←<EXP1>         F O R PRODUCTION <EXP1>       ::=  <EXP1>      <+,-,x>     <EXP1 FACT>

406  P←P-1   /   ST2P←<EXP2>         F  O  R PRODUCTION <EXP2>     ::=  <EXP1>      <+,-,x>     <EXP2 FACT>

407    P←P-1 /   ST2P←<EXP1>         F O R PRODUCTION <EXP1>       ::=  <+,-,x>     <EXP1 FACT>

408   P←P-1  /   ST2P←<EXP2>         F O R PRODUCTION <EXP2>       ::=  <+,-,x>     <EXP2 FACT>

409  P←P-1   /   ST2P←<EXP1 FACT>    FOR PRODUCTIOh <EXP1FACT>     ::=  (           <EXP1>      )                      /  NEW  TI

410  P←P-1       ST2P←<EXP2 FACT>    F O R PRODUCTION <EXP2 F A C T*  ::=           <EXP2>                             /  NEW T1

411    P←P-1 /   ST2P←<EXP2>         FOR PRODUCTIOh <EXP2>         ::=  <EXP2>      <+,-,x>     <EXP1 FACT>

412  P←P-1   /   ST2P←<EXP2>         F O R PRODUCTION <EXP2>       ::=  <EXP2>      <+,-,x>     <EXP2 FACT>
```

Figure 12 (continued)



The loop checker disconnects itself as soon as it determines that the loop is not of the right form, or when it is finished. Similarly, the optimizable subscript checker is connected when the [of the subscript variable $A[E_1, \ldots . E_n]$ is first scanned, The optimizable variable checker disconnects itself when finished or when it is determined that this subscripted variable does not satisfy one of the restrictions.



The loop checker and optimizable subscript checker are not concerned with errors and error recovery, If any error occurs, they simply disconnect themselves, or will be disconnected by the syntax checker itself.

10. Summary.

The ideas in this paper have been used intuitively in the ALCOR-ILLINOIS 7090 Compiler ([5], [1]). The second pass actually contains the three matrices illustrated in Section 9. The matrix technique has its most important use, in my opinion, in a student system, where a very fast compiler resides in core and must also produce excellent error messages. Because the syntax checker matrix for ALGOL is so large (on the 7090 (100 x 45) and because over sixty percent of the array elements represent illegal symbol pairs, a much wider variety of error messages is efficiently possible. An algorithm is being developed for producing, from the grammar, an error recovery subroutine for each "error" element of the matrix. Another advantage of the matrix technique is the simplicity of the overall design.

The only disadvantage is the space used. A partial solution to this problem might be to parse those constructions of the grammar which are most used (for instance, expressions) using the matrix technique and to use some other slower but less space-consuming technique for the rest of the grammar. Note also that the size of the matrix may be cut in half by allowing only one row for each stack nonterminal symbol. Each subroutine must then check whether a nonterminal exists in $ST2_P$ or not ($ST2_P$ nonzero or not).

REFERENCES

[1] Bayer, R., Murphree Jr., E., Gries, D. User's Manual for the ALCOR-ILLINOIS 7090 ALGOL-60 Translator, 2nd ed., U. of Illinois, Sept. 1964.

[2] Eickel, J., Paul, M., Bauer, F. L. and Samelson, K. "A Syntax Controlled Generator of Formal Language Processors," Comm. ACM 6 (Aug. 1963), 451 - 455.

[3] Feldman, J. A., 'A Formal Semantics for Computer Languages and its Application in a Compiler-compiler," Comm. ACM 9 (Jan. 1966), 3-9.

[4] Floyd, R. W., "Syntactic Analysis and Operator Precedence," J. ACM 10 (July 1963) 316 - 333.

[5] Gries, D., Paul, M. and Wiehle, H. R., 'Some Techniques Used in the ALCOR-ILLINOIS 7090," Comm. ACM 8 (Aug. 1965), 496 - 500.

[6] Hill, V., Langmaack, H., Schwarz, H. R., and Seegmüller, G. "Efficient handling of subscripted variables in ALGOL 60 compilers,' Proc. 1962 Rome Symposium on Symbolic Languages in Data Processing, Gordon and Breach, New York, 1962, 311 - 340.

[7] Nauer, P. (Ed.), "Report on the Algorithmic Language ALGOL 60," Num. Math. 2 (1960), 106 - 136; Comm. ACM 3 (May 1960), 299 - 314.

[8] ________ 'Revised Report on the Algorithmic Language ALGOL 60," Comm. ACM 6 (Jan. 1963), 1 - 17.

[9] Samelson, K. and Bauer, F. L., "Sequential Formula Translation," Comm. ACM 3 (Feb. 1960), 76 - 83.

[10] Wirth, N. and Weber, H., "EULER: A Generalization of ALGOL, and its Formal Definition: Part I," Comm. ACM 9 (Jan. 1966), 13 - 25.

[11] Burroughs B5500 Information Processing Systems Extended ALGOL Language Manual.

Appendix A.

Below is the grammar of an ALGOL-like language defined by Floyd ([4]) in his article on operator precedence. Note that in the subroutines the phrase "ambiguity in sub" appears 6 times. This means that in this subroutine the reduction to make is not uniquely determined from U_ℓ^*, U_1 and T_1, and not that the grammar is ambiguous. The difficulties can be circumvented by either changing the grammar or using semantic information to determine which reduction to be made.

The matrix and subroutines were constructed using the grammar as input.

PRODUCTIONS

```
 1  ID            ::=  LETTER
 2  <ID LIST>     ::=  ID
 3                ::=  <ID LIST>     ,              ID
 4  <LITCON>      ::=  DIGITSTR
 5                ::=  DIGITSTR      .
 6                ::=  .             DIGITSTR
 7                ::=  DIGITSTR      .              DIGITSTR
 8  <SUB VAR>     ::=  ID            [              <AEXP LIST>  ]
 9  <VARIABLE>    ::=  ID
10                ::=  <SUB VAR>
11  <FUNC DES>    ::=  ID            (              <EXP LIST>  )
12  <PRIMARY>     ::=  <FUNC DES>
14                ::=  <VARIABLE>
15                ::=  <LITCON>
                  ::=  (             <ARITH EXP> )
16  <FACTOR>      ::=  <PRIMARY>
17                ::=  <PRIMARY> * * <FACTOR>
18                ::=  @             <FACTOR>
19                ::=  <+ OR ->      <FACTOR>
20  <TERM>        ::=  <FACTOR>
21                ::=  <TERM>        <* OR />       <FACTOR>
22  <SIMP AEXP>   ::=  <TERM>
23                ::=  <SIMP AEXP>   <+ OR ->       <TERM>
24  <ARITH EXP>   ::=  <SIMP AEXP>
25                ::=  IF            <BOOL EXP>     THEN         <ARITH EXP> ELSE      <ARITH EXP>
26  <AEXP LIST>   ::=  <ARITH EXP>
27                ::=  <AEXP LIST>   ,              <ARITH EXP>
28  <RELATION>    ::=  <ARITH EXP>   <REL OP>       <ARITH EXP>
29                ::=  <RELATION>    <REL OP>       <ARITH EXP>
30  <BOOL PRIM>   ::=  <T OR F>
31                ::=  <VARIABLE>
32                ::=  <FUNC DES>
34                ::=  <RELATION>
35                ::=  (             <BOOL EXP>     )
    <BOOL SEC>    ::=  <BOOL PRIM>
36                ::=  NOT           <BOOL PRIM>
37  <CONJ>        ::=  <BOOL SEC>
38                ::=  <CONJ>        AND            <BOOL SEC>
39  <DISJ>        ::=  <CONJ>
40                ::=  <DISJ>        OR             <CONJ>
41  <IMPL>        ::=  <DISJ>
42                ::=  <IMPL>        IMPLIES        <DISJ>
43  <BOOL EXP>    ::=  <IMPL>
44                ::=  <BOOL EXP>    EQUIV          <IMPL>
45  <EXPR>        ::=  <ARITH EXP>
46                ::=  <BOOL EXP>
47  <EXP LIST>    ::=  <EXPR>
48                ::=  <EXP LIST>    ,              <EXPR>
49  <LIM PAIR>    ::=  <ARITH EXP>   :              <ARITH EXP>
50  <LIM P LIST>  ::=  <LIM PAIR>
51                ::=  <LIM P LIST>  ,              <LIM PAIR>
52  <NAME PART>   ::=  ID            (              <ID LIST>  )
53  <SPECIFIER>   ::=  <TYPE>        <ID LIST>
```

```
54                ::=  <TYPE>        VALUE          <ID LIST>
55                ::=  <TYPE>        ARRAY          <ID LIST>
56                ::=  ARRAY         <ID LIST>
57  <SPEC LIST>   ::=  <SPECIFIER>
58                ::=  <SPEC LIST>   ;              <SPECIFIER>
59  <DECL>        ::=  <TYPE>        <ID LIST>
60                ::=  CONSTANT      ID             :=             <EXPR>
61                ::=  <TYPE>        ARRAY          <ID LIST>      [              <LIM P LIST>]
62                ::=  ARRAY         <ID LIST>      [              <LIM P LIST>]
63                ::=  SWITCH        <NAME PART>
64                ::=  PROCEDURE     <NAME PART>    ;              <SPEC LIST> ;  <ST LIST>    END
65                ::=  FUNCT IDN     <NAME PART>    ;              <STATEMENT>
66  <DEC LIST>    ::=  <DECL>
67                ::=  <DEC LIST>    ;              <DECL>
68  <FOR L EL>    ::=  <ARITH EXP>
69                ::=  <ARITH EXP>   STEP           <ARITH EXP>    UNTIL          <ARITH EXP>
70                ::=  <ARITH EXP>   WHILE          <BOOL EXP>
71  <FOR LIST>    ::=  <FOR L EL>
72                ::=  <FOR LIST>    ,              <FOR L EL>
73  <GO STATE>    ::=  GO TO         ID
74                ::=  GO TO         ID             [              <ARITH EXP> ]
75  <ASS STATE>   ::=  <VARIABLE>    :=             <EXPR>
76                ::=  <VARIABLE>    :=             <ASS STATE>
77  <PROC CALL>   ::=  ID            (              <EXP LIST>     )
78  <COMP ST>     ::=  BEGIN         <ST LIST>      END
79                ::=  BEGIN         <DEC LIST>     ;              <ST LIST>      END
80  <CL STATE>    ::=  <GO STATE>
81                ::=  <ASS STATE>
82                ::=  <PROC CALL>
83                ::=  ID
84                ::=  ID            :              <CL STATE>
85                ::=  COMMENT
86                ::=  <COMP ST>
87                ::=  FOR           ID             :=             <FOR LIST> DO  <CL STATE>
88                ::=  IF            <BOOL EXP>     THEN           <CL STATE> ELSE <CL STATE>
89  <OP STATE>    ::=  ID            :              <OP STATE>
90                ::=  FOR           ID             :=             <FOR LIST> DO  <OP STATE>
91                ::=  IF            <BOOL EXP>     THEN           <CL STATE> ELSE <OP STATE>
92                ::=  IF            <BOOL EXP>     THEN           <STATEMENT>
93  <STATEMENT>   ::=  <CL STATE>
94                ::=  <OP STATE>
95  <ST LIST>     ::=  <STATEMENT>
96                ::=  <ST LIST>     ;              <STATEMENT>
97  <PROGRAM>     ::=  PHI           <STATEMENT>    PHI

NONTERMINAL SYMBOLS
 1  ID                 2  <ID LIST>        3  <LITCON>        4  <SUB VAR>
 5  <VARIABLE>         6  <FUNC DES>       7  <PRIMARY>       8  <FACTOR>
 9  <TERM>            10  <SIMP AEXP>     11  <ARITH EXP>    12  <AEXP LIST>
17  <RELATION>        14  <BOOL PRIM>     15  <BOOL SEC>     16  <CONJ>
21  <EXP <DISJ>LIST>  2218 <IMPL>         19  <BOOL EXP>     20  <EXPR>
                          <LIM PAIR>      23  <LIM P LIST>   24  <NAME PART>
25  <SPECIFIER>       26  <SPEC LIST>     27  <DECL>         28  <DEC LIST>
29  <FOR L EL>        30  <FOR LIST>      31  <GO STATE>     32  <ASS STATE>
33  <PROC CALL>       34  <COMP ST>       35  <CL STATE>     36  <OP STATE>
```

37	<STATEMENT>	38	<ST LIST>	39	<PROGRAM>

TERMINAL SYMBOLS

40	LETTER	41	,	42	DIGITSTR	43	.
44	[	45	]	46	(	47	)
48	**	49	@	50	<+ O R ->	51	<* O R />
52	IF	53	THEN	54	ELSE	55	<REL OP>
56	<T O R F>	57	NOT	58	AND	59	OR
60	IMPLIES	61	EQUIV	62	:	63	<TYPE>
64	VALUE	65	ARRAY	66	;	67	CONSTANT
68	:=	69	SWITCH	70	PROCEDURE	71	END
31	FUNCTION	73	STEP	74	UNTIL	75	WHILE
	GO TO	77	BEGIN	78	COMMENT	79	FOR
80	00	81	PHI				

MATRIX IS 116 × 42

col	1				5					10					15					20			23
	LETTER	,	DIGITSTR	.	[	]	(	)	**	@	<+ OR ->	<* OR />	IF	THEN	ELSE	<REL OP>	<T OR F>	NOT	AND	OR	IMPLIES	EQUIV	;
<ID LIST> ,	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0
SECONDARY ROW	0	5	0	0	5	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DIGITSTR	0	6	0	7	0	6	0	6	6	0	6	6	0	6	6	6	0	0	6	6	6	6	6
SECONDARY ROW	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0
.	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DIGITSTR .	0	9	10	0	0	9	0	9	9	0	9	9	0	9	9	9	0	0	9	9	0	9	9
SECONDARY ROW	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
ID [	2	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	3	0	0	0	0	0
SECONDARY ROW	0	11	1	1	12	13	14	0	15		lb	17	1	0	0	0	0	1	0	0	0	0	0
ID (	2	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	3	0	0	0	0	0	0
SECONDARY ROW	0	18			12	0	14	19	15	0	16	17		0	0	20	0		21	22	23	24	0
	2	0	1	1	0	0	1	0	0	1	1	0	1	0	0	0	3	1	0	0	0	0	0
SECONDARY ROW	2	0	0	0	12	0	14	25	15	1	16	17	0	0	0	20	0	0	21	22	23	24	0
<PRIMARY> **	0	26 0	0 1	0 1	12 0	26 0	14 1	26 0	15 0	0	26 1	26 0	0 0	26 0	26 0	26 .0	0	0	28	26 0	26 0	0	0
SECONDARY ROW																	0					26	26
@										1	1	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	27 0	10	0 1	12	29	14	27	15	0	27	27	0	27	27	27	0	0	27	27	27	27	27
<+ OR ->	2	0	1	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	28	0	0	12	28	14	28	15	0	28	28	0	28	28	28	0	0	28	28	28	28	28
<TERM> <* OR />	2	0	1	1	0	0		0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	2	29 0	10		12	29	1:	29	15	0	29	29	0	0	29	29	0	0	29	29	29	29	29
<SIMP AEXP> <+ OR ->	0	30 0	0	1	0	0	1	0	0	1	1	0	0 0	30	30 0	30 0	0	0 0	30 0	30 0	30 0	0	0
SECONDARY ROW				0	12	30	14	30	15	0	30	17										30	30
IF	2	0	1	1	0	0	1	0	0	1	1	0	1	0	0	0	3	1	0	0	0	0	0
SECONDARY ROW	0	0	0	0	12	0	14	0	15	0	16	17	0	31	0	20	0	0	21	22	23	24	0
IF <BOOL EXP>																							
THEA	2	0	1	1	0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	12	0	14	0	15	0	16	17	0	0	32	0	0	0	0	0	0	0	33
IF <BOOL EXP>																							
THEN <ARITH EXP>																							
ELSE	2	0	1	1	0	0		10	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	1	0	12	36	14	36	15	0 1	1 1	0	0	3 6	36	36	0	0	36	36	36	36	36
<AEXP LIST> ,	2	37	0	1	0	0	1	0	0	0	16	17	1	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0			0	12	37	14	0	15				0	0	0	0	0	0	0	0	0	0	0
<ARITH EXP> <REL OP>	2	0	1	1	0	0		10	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	38	0	0	12	0	14	38	15	0	16	17	0	38	38	38	0	0	38	38	38	38	0
<RELATION> <REL OP>	2	0	1	1	0	0					1 1	0	1	0	0	0	0	0		0	0	0	0
SECONDARY ROW	0	39	0	0	12	0 14	1	39 0	15	0	0 16	17	0	39	39	39	0	0	39 0	39	39	39	0
NOT	2	. 0	1	1	0	0	1	0	0	1	1	0	1	0	0	0	3	0	0	0	0	0	0
SECONDARY ROW	0	40	0	0	12	0	14	40	15	0	16	17	0	40	40	20	0	0	40	40	40	40	0
<CONJ> AND	2	0	1	1	0	0	1	0	0			0	1	0	0	0	3	1	0	0	0	0	0
SECONDARY ROW	0	41	0	0	12	0	14	41	15	0 1	16 1	17	0	41	41	20	0	0	41	41	41	41	0

	1				5					10					15					20			23
<DISJ> OR	2	0	1	1	0	0	1	0	0	1	1	0	1	0	0	0	3	1	0	0	0	0	0
SECONDARY ROW	0	42	0	0	12	0	14	42	15	0	16	17	0	42	42	20	0	0	21	42	42	42	0
<IMPL> IMPLIES	2	0	1	1	0	0	1	0	0	1	1	0	1	0	0	0	3	1	0	0	0	0	0
SECONDARY Row	0	0	0	0	12	0	14	43	15	0	lb	17	0	43	43	20	0	0	21	22	43	43	0
<BOOL EXP> EQUIV	2	44	1	1	0	0	1	0	0	1	1	0	1	0	0	0	3	1	0	0	0	0	0
SECONDARY ROW	2	0	1	0	12	0	14	44	15	0	16	17	0	44	44	20	0	0	21	22	23	44	0
<EXP LIST> ,	0	45	0	01	0	0	1	0	0	1	1	0	1	0	0	0	3	1	0	0	0	0	0
SECONDARY ROW					12	0	14	45	15	0	16	17	0	0	0	20	0	0	21	22	23	24	0
<ARITH EXP> :	2	0	1	1	0	0																0	0
SECONDARY ROW	0	46	1	0	12	46	12 0	0	150	01	1lb	17 0	0	10 0	00	00	00	0 0	0 0	00	00	0	0
<LIM P LIST>,	2	0	0	1	0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	47		0	12	47	14	0	15	0	lb	17	0	0	0	0	0	0	0	0	0	0	48
<TYPE>	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	51	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<TYPE> VALUE	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SEcondaRY ROW	0	51	0	0	0	0												0	0		0	0	0
<TYPE' ARRAY	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	51	0	0	54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ARRAY	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	51	0	0	56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<SPEC LIST> ;	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CONSTANT	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CONSTANT ID																							
:=	2	0'1		1	0	0	1	0	0	1	1	0	1	0	0	0	3	1	0	0	0	0	0
SECONDARY ROW	0	0	0	0	12	0	14	0	15	0	lb	17	0	0	0	20	0	0	21	22	23	24	0
<TYPE> ARRAY																							
<ID LIST> [	2	0	1	1	0	0	10		0	110		10		0		0	0	0	0	0	0	0	0
SECONDARY ROw	0	61	0	0	12	62	14	0	15	0	lb	17	0	0	0	0	00	0	0	0	0	0	48
ARRAY <ID LIST>																							
[	2	0	1	1	0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
SecondarY Row	0	61	0	0	12	63	14	0	15	0	lb	17	0	0	0	0	0	0	0	0	0	0	48
SWITCH	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PROCEDURE	2	0	0	0	0	0	0															0	0
SECONDARY ROW	0	0	0	0	0	0	14 0	0	00	0	0 0	0 0	0 0	0 0	00	00	00	00	00	00	00	0	0
PROCEDURE <NAME PART>																							
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PROCEDURE <NAME PART>																							
; <SPEC LIST>																							
	2	0	0	0	0	0	0	0	0	0	0	0		10	0	0	0	0	0	0	0	0	0
SECoNDaRy ROW	0	0	0	0	12	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33
FUNCTION	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6 9
FUNCTION <NAME PART>																							
I				0	0	0	0	0			0	0	1	0			0		0			0	0
SECONDARY ROW	0	0	0	0	12	0	14	0	00	00	0	0	0	0	00	00	0	00	0	00	00	0	33
<DEC L I S T * ;	0	00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	2	00	01	01	00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<ARITHEXP> STEP						0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	12	0	14	0	15	0	16	17	0	0	0	0	0	0	0	0	0	0	0
<ARITHEXP> STEP																							
<ARITHEXP> UNTIL	2	0	1	1	0	0	10		0	110		10	0		0	0	0	0		0	0	0	0

	1				5					10					15					20			23
SECONDARY Row	0	73	0	0	12	0	14	0	15	0	16	17	0	0	0	0	0	0	0	0	0	0	0
<ARITHEXP> WHILE	2	0	1	1	0	0	1	0	0	1	1	0	1	0	0	0	3	1	0	0	0	0	0
SECONDARY ROW	2	74	0	0	12	0	14	0	15	0	lb	17	0	0	0	20	0	0	21	22	23	24	0
<FOR LIST>	0	750	1	1	0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
SECONOARY'ROW			0	0	0	0	10	0	10	0	llo	10	0	0	0	0	0	0	0	0	0	0	0
GO TO	2	0	0	0	78	0	0	0	0	0	0	0	0 0 0 0		79	00	0	00	00	0	00	0	0
SECONDARY row	0	0																				0	0
GO TO 10																							
[	2	0		1	0	0	1	0	0	1		0	1	0	0	0	0	0	0	0	0	0	0
SECONDARY row	0	0	0	0	12	80	14	0	15	0	16	17	0	0	0	0	0	0	0	0	0	0	0
<VARIABLE> :=	2	0	1	1	-0	0	-1	0	-0	1	1	0	1	0	0	0	3	1	0	0	0	0	0
SECONDARY RoW	0	0	0	0	12	0	14	0	15	0	16	17	0	0	81	20	0	Cl	21	22	23	24	0
BEGIN	2	0	0	0	0	0	0	0	0	0	0	0	1	0		0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	12	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33
BEGIN <DEC LIST>				0																			
	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0		12	0	14	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	33
10 :	2	0	0	0	0	0	0	0	0	0	0	0	0	0	85	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	12	0	14	0						0	86	0	0	0	0	0	0	0	33
FOR	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY Row	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FOR 10																							
:=	2	0	1	1	0	0		0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	89	0	0	12	0	1:	0	15	0	1b	17	0	0	0	0	0	0	0	0	0	0	0
FOR 10																							
:= <FOR LIST>																							
DO	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
SECONDARY Row	0	0	0	0	12	0	14 .	0	0	0	0	0	0	0	91	0	0	0	0	0	0	0	33
IF <BOO, EXP>																							
THEN <CL STATE>																							
ELSE	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
SECONDARY row	0	0	0	0	12	0	14	0	0	0	0	0	0	0	93	0	0	0	0	0	0	0	33
<ST LIST> ;	2	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	12	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33
PHI	2	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0
SECONDARY Row	0	0	0	0	12	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33

	<TYPE> (24)	VALUE	ARRAY	;	CONSTANT	:=	SWITCH (30)	PROCEDURE	END	FUNCTION	STEP	UNTIL (35)	WHILE	GO TO	BEGIN	COMMENT	FOR (40)	DO	PHI (42)
<ID LIST*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONOARY'ROW	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DIGITSTR	0	0	0	6	0	0	0	0	6	0	b	6	6	0	0	0	0	6	6
SECONDARY ROW	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
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DIGITSTR .	0	0	0	9	0	0	0	0	9	0	9	9	9	0	0	0	0	9	9
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10 [	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10 C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	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
SECONDARY Row	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0
<PRIMARY> **	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0 0	26	0	0	0	0	26	0	26	26	26	0	0	0	0	26	26
@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0		27	0	0	0		0	0	27	27	27	0	0	0	0	27	27
<+ C R =>	0	0	0	0	0	0	0	0	28	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0		0	28				0		0	28	28	28				0	28	28
																			0
<TERM> SECONDARY ROW />	0	0	0	29	0	0	0	0	29	0	29	29	29	0	0	0	0	29	29
				0						0	0	0	0				0	0	0
<SIMPLE AEXP> SECONDARY ROW R =>	0	0	0 0	30	0	0	0	0	30	0	30	30	30	0	0	0	0	30	30
IF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IF <BOOL EXP>		0	0				0		0	0		0	0	1	1	4	1	0	
THEN	0	0	0	0	0	0	0	0	34	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0			34	0	35		0			0								34
IF <BOOL EXP>																			
THEN <ARITH EXP>																			
ELSE		0			0	0	0								0				0
SECONDARY ROW	0	0	0	36	0	0	0	0	36	0	36	36	36	0	0	0	0	36	36
<AEXP LIST> ,	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<ARITH EXP> <REL OP>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	38	0	0	0	0	38	0	0	0	0	0	0	0	0	38	38
<RELATION* <REL OP>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	39	0	0	0	0	39	0	0	0	0	0	0	0	0	39	39
NOT	o -	o	0	400	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY Row	0	0			0	0	0	0	40	0	0	0	0	0	0	0	0	40	40
<CONJ> AN0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY Row	0	0	0	41	0	0	0	0	41	0	0	0	0	0	0	0	0	41	41

	24 30 35 40 42
<DISJ> OR	0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 042 0 0 0 0 0 0 0 0 42 0 0 0 0 0 0 0 0 0 0 0 0 0 4 2 0 42
<IMPL> IMPLIES	0 0 0 0 0 00 0 0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 0 . 4 3 0 0 0 043 0 0 0 0 0 0 0 0 43 43
<BOOL EXP> EQUIV	0 0 0 0 44 0 0 0 0 0 0 0 0 0 44 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 440440
SECONDARY ROW	0 0 0
<EXP LIST> ,	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
<ARITH EXP> :	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
<LIM P LIST>,	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
<TYPE>	0 49 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 0 5 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
<TYPE> VALUE	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 0 5 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
<TYPE> ARRAY	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 5 5 0 0 0 0 0 0 0 0 0 0 0 0 0" 0 0
ARRAY	0 0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 0 57 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0
<SPEC LIST> ;	1 0 1 0 0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 0 5 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CONSTANT	0 0 0 0 0 0 0 0 0 0 00 00 00 00 00 0 0 0 0
SECONDARY ROW	0 0 0 0 0 5 9 0 0 0 0 0 0 0 0 0 0 0 0 0
CONSTANT I O	
:=	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
<TYPE> ARRAY	
<ID LIST> [	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0'0 0 0
SECONDARY Row	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ARRAY <ID LIST>	
[	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SWITCH	0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 00 0 0 0 0 00 0 0 0 00 00 0 00 0 00 00 00
PROCEDURE	0 0 00 65 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PROCEDURE <NAME PART*	
;	1 0 0 0 0 0
SECONDARY ROW	0 0 0 66 0 00 0 0 0 0 00 00 00 00 00 00 0 00 00 00
PROCEDURE <NAME PART>	0
I <SPEC LIST>	
;	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 4 1 0 0
SECONDARY ROW	0 0 0 67 0 35 0 0 68 0 0 0 0 0 0 0 0 0 0
FUNCTION	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SECONOARY ROW	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FUNCTION <NAME PART>	
:	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 4 1 0 0
SECONDARY ROW	0 0 0 70 0 3 5 0 0 0 0 0 0 0 0 0 0 0 0 0
<DEC LIST> ;	1 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0
SECONDARY ROW	0 00 0 70 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
<ARITHEXP> STEP	0 · 00 0 0 0 0 0 0 0 0 0 0 0 7 x 0 0 0 0 0 0 0 0 0 00
SECONDARY ROW	0
<ARITHEXP> STEP	
<ARITHEXP> UNTIL	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

	24						30					35					40		42
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	73	0
<ARITHEXP> WHILE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	74	0
<FOR LIST> ,	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	76	0	77	0	0	0	0	75	0
GO TO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	79	0	0	0	0	79	0	0	0	0	0	0	0	0	0	79
GO TO ID																			
[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<VARIABLE> :=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	81	0	35	0	0	81	0	0	0	0	0	0	0	0	0	81
BEGIN	1	0	1	0	1	0	1	1	0	1	0	0	0	1	1	4	1	0	0
SECONDARY ROW	0	0	0	82	0	35	0	0	83	0	0	0	0	0	0	0	0	0	0
BEGIN <DEC LIST>																			
;	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	4	1	0	0
SECONDARY ROW	0	0	0	67	0	35	0	0	84	0	0	0	0	0	0	0	0	0	0
ID :	0	0	0	85	0	0	0	0	85	0	0	0	0	1	1	4	1	0	85
SECONDARY ROW	0	0	0	87	0	35	0	0	87	0	0	0	0	0	0	0	0	0	87
FOR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	88	0	0	0	0	0	0	0	0	0	0	0	0	0
FOR ID																			
:=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SECONDARY ROW	0	0	0	0	0	0	0	0	0	0	76	0	77	0	0	0	0	90	0
FOR ID																			
:= <FOR LIST>																			
DO	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	4	1	0	0
SECONDARY ROW	0	0	0	92	0	35	0	0	92	0	0	0	0	0	0	0	0	0	92
IF <BOOL EXP>																			
THEN <CL STATE>																			
ELSE	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	4	1	0	0
SECONDARY ROW	0	0	0	94	0	35	0	0	94	0	0	0	0	0	0	0	0	0	94
<ST LIST> ;	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	4	1	0	0
SECONDARY ROW	0	0	0	95	0	35	0	0	95	0	0	0	0	0	0	0	0	0	0
PHI	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	4	1	0	0
SECONDARY ROW	0	0	0	0	0	35	0	0	0	0	0	0	0	0	0	0	0	0	96

```
MATRIX  SUBROUTINES,  THOSE SUBS WHICH ARE ACTUALLY MATRIX ENTRIES HAVE NUMBERS LESS THAN 400

 1  ST1P←ST1P+1    /    P←P+1    /    ST1P←ROW(T1)    /  NEW  T1
 2   ST1P←ST1P+1  /    ST2P←ID              FOR PRODUCTION 10           ::=  LETTER          /  NEW T1
 3   ST1P←ST1P+1  /    ST2P←<BOOL PRIM>    FOR PROCUCTION <BOOL PRIM> ::=  <T OR F>        /  NEW TI
 4  ST1P←ST1P+1    /    ST2P←<CL STATE>     FOR PRODUCTION <CL STATE> ::=  COMMENT         /  NEW T1
 5   USE SUB 424 IF ST2P IS ID
 6  P←P-1        ST2P←<LITCON>        FOR PRODUCTION <LITCON>     ::=  DIGITSTR
    ST1P←ROW(  DIGITSTR                          /  NEW  TI
 8  P←P-1        ST2P←<LITCON>        FOR PRODUCTION <LITCON>     ::=  .          DIGITSTR        /  NEW T1
 9  P←P-1        ST2P←<LITCON>        FOR PRODUCTION <LITCON>     ::= OIGITSTR   .
10  P←P-1        ST2P←<LITCON>        FOR PRODUCTION <LITCON>     ::=  OIGITSTR   .          OIGITSTR        /  NEW TI

11    USE SUB 406 IF ST2P IS <AEXP LIST> <ARITH EXP> <SIMP AEXP> <TERM>     <FACTOR>    <PRIMARY>   <FUNC DES>
                          <VARIABLE> I  O          <SUB VAR>   <LITCON>

12    USE SUB 401 IF ST2P IS ID

13    USE SUB 425 IF ST2P IS <AEXP LIST> <ARITH EXP> cSIMP AEXP> <TERM>     <FACTOR>    <PRIMARY>   <FUNC DES>
                          <VARIABLE> I  O          <SUB VAR>   <LITCON>

14    USE SUB 402 IF ST2P IS ID

15    USE SUB 433 IF ST2P IS <PRIMARY>   <FUNC DES>  <VARIABLE>   ID          <SUB VAR>   <LITCON>

Ib    USE SUB 435 IF ST2P IS <SIMP AEXP> <TERM>      <FACTOR>     <PRIMARY>    <FUNC DES>  <VARIABLE>  ID
                          <SUB VAR>   cLITCON>

17    USE SUB 404 IF ST2P IS <TERM>      <FACTOR>    <PRIMARY>    <FUNC DES>   <VARIABLE> I  D          <SUB VAR>
                          <LITCON>

18 EITHER
      USE SUB 400 IF ST2P  IS  <IO LIST>   IO
      USE SUB 413 IF ST2P IS <EXPLIST>   <EXPR>      <ARITH EXP> <SIMP A E X P > <TERM>     <FACTOR>    <PRIMARY>
                          <FUNC DES>  <VARIABLE>  IO         <SUB VAR>   <LITCON>    <BOOL EXP>  <IMPL>
                          <DISJ>      <CONJ>      <BOOL SEC>  <BOOL PRIM> <RELATION>
   AMBIGUITY IN SUB 18

19 EITHER
      USE SUB 427 IF ST2P IS <IDLIST>    IO
      USE SUB 426 IF ST2P IS <EXPLIST>   <EXPR>      <ARITH EXP> <SIMP AEXP> <TERM>      <FACTOR>    <PRIMARY>
                          <FUNC DES>  <VARIABLE> 1  0         <SUB VAR>   <LITCON>    <BOOL EXP>  <IMPL>
                          <DISJ>      <CONJ>      <BOOL SEC>  <BOOL PRIM> *RELATION>
      USE SUB 428 IF ST2P IS <EXPLIST>   <EXPR>      <ARITH EXP> <SIMP AEXP> <TERM>      <FACTOR>    <PRIMARY>
```

```
                                 <FUNC DES>   <VARIABLE> IO        <SUB VAR>   <LITCON>    <BOOL EXP>   <IMPL>
                                 <DISJ>       <CONJ>     <BOOL SEC> <BOOL PRIM> <RELATION>
   AMBIGUITY IN SUB 19

20 EITHER
      USE SUB 437 IF ST2P IS <ARITH EXP> <SIMP AEXP> <TERM>     <FACTOR>    <PRIMARY>   <FUNC DES>   <VARIABLE>
                                 ID           <SUB VAR>  <LITCON>
      USE SUB 408 IF ST2P IS <RELATION>

21    USE SUB 409 IF ST2P IS <CONJ>   <BOOL SEC>  <BOOL PRIM> *VARIABLE> IO        <SUB VAR>   <FUNC DES>
                                 <RELATION>

22    USE SUB 410 IF ST2P IS <DISJ>   <CONJ>      <BOOL SEC>  <BOOL PRIM> <VARIABLE> ID        <SUB VAR>
                                 <FUNC DES>   <RELATION>

23    USE SUB 411 IF ST2P IS <IMPL>   <DISJ>      <CONJ>      <BOOL SEC>  <BOOL PRIM> <VARIABLE>  10
                                 <SUB VAR>    <FUNC DES> <RELATION>

24    USE SUB 412 IF ST2P IS <BOOL EXP> <IMPL>    <DISJ>      <CONJ>      <BOOL SEC>  <BOOL PRIM> <VARIABLE>
                                 ID           <SUB VAR>  <FUNC DES>  <RELATION>

25 EITHER
      USE SUB "29 IF ST2P IS <ARITH EXP> <SIMP AEXP> <TERM>     <FACTOR>    <PRIMARY>   <FUNC DES>   <VARIABLE>
                                 ID           <SUB VAR>  <LITCON>
      USE SUB 430 IF ST2P IS <BOOL EXP> <IMPL>    <DISJ>      <CONJ>      <BOOL SEC>  <BOOL PRIM> <VARIABLE>
                                 ID           <SUB VAR>  <FUNC DES>  <RELATION>
   AMBIGUITY IN SUB 25

26    USE SUB 431 IF ST2P IS <FACTOR>  <PRIMARY>   <FUNC DES>  <VARIABLE>  10        <SUB VAR>   <LITCON>

27    USE SUB 432 IF ST2P IS <FACTOR>  <PRIMARY>   <FUNC DES>  <VARIABLE,  10        <SUB VAR>   <LITCON>

28    USE SUB 433 IF ST2P IS <FACTOR>  <PRIMARY>   <FUNC DES>  <VARIABLE> IO         <SUB VAR>   <LITCON>

29    USE SUB 434 IF ST2P IS <FACTOR>  <PRIMARY>   <FUNC DES>  <VARIABLE> IO         <SUB VAR>   <LITCON>

30    USE SUB 435 IF ST2P IS <TERM>    <FACTOR>    <PRIMARY>   <FUNC DES>  <VARIABLE> 10         <SUB VAR>
                                 <LITCON>

31    USE SUB 436 IF ST2P IS <BOOL EXP> <IMPL>    <DISJ>      <CONJ>      <BOOL SEC>  <BOOL PRIM> <VARIABLE>
                                 ID           <SUB VAR>  <FUNC DES>  <RELATION>

32 EITHER
      USE SUB 437 IF ST2P IS <ARITH EXP> <SIMP AEXP> <TERM>     <FACTOR>    <PRIMARY>   <FUNC DES>   <VARIABLE>
                                 IO           <SUB VAR>  <LITCON>
      USE SUB 438 IF ST2P IS <CL STATE>  <GO STATE>  <ASS STATE> <PROC CALL> <COMP ST>

33    USE SUB 422 IF ST2P IS ID

34    USE SUB 439 IF ST2P IS <STATEMENT> <CL STATE> <GO STATE> <ASS STATE> <PROC CALL> <COMP ST>  <OP STATE>
```

```
35    USE SUB 421 IF ST2P IS <VARIABLE>  IO              <SUB VAR>

36    USE SUB 440 I  F ST2P I  S <ARITH EXP> <SIMP AEXP> <TERM>     <FACTOR>   <PRIMARY>   <FUNC DES>  <VARIABLE>
                                 IO          <SUB VAR>   <LITCON>

37    USE SUB 441 IF ST2P IS <ARITH EXP>     <SIMP AEXP> <TERM*     <FACTOR>   <PRIMARY*   <FUNC DES>  <VARIABLE>
                                 10          <SUB VAR>   <LITCON>

38    USE SUB 442 IF STEP IS <ARITH EXP>     <SIMP AEXP> <TERM>     <FACTOR>   <PRIMARY>   <FUNC DES>  <VARIABLE>
                                 ID          <SUB VAR>   <LITCON>

39    USE SUB 443 IF ST2P IS <ARITH EXP>     <SIMP AEXP> <TERM>     <FACTOR>   *PRIMARY>   <FUNC DES>  <VARIABLE>
                                 ID          <SUB VAR>   <LITCON>

00    USE SUB 444 IF ST2P IS <BOOL PRIM> <VARIABLE>  10             <SUB VAR>  <FUNC DES>  <RELATION>

41    USE SUB 445 IF ST2P IS <BOOL SEC>  <BOOL PRIM> <VARIABLE>  ID            <SUB VAR>   <FUNC DES>  <RELATION>

42    USE SUB 446 IF ST2P IS <CONJ>      <BOOL SEC>  <BOOL PRIM> <VARIABLE>  ID            <SUB VAR>   <FUNC DES>
                                 <RELATION>

43    USE SUB 447 IF ST2P IS <DISJ>      <CONJ>      <BOOL SEC>  <BOOL PRIM> <VARIABLE>  ID            <SUB VAR>
                                 <FUNC DES>  <RELATION>

44    USE SUB 448 IF ST2P IS <IMPL>      <DISJ>      <CONJ>      <BOOL SEC>  <BOOL PRIM> <VARIABLE,    10
                                 <SUB VAR>   <FUNC DES>  <RELATION*

45    USE SUB 449 IF ST2P IS <EXPR>      <ARITH EXP> <SIMP AEXP> <TERM>      *FACTOR>    <PRIMARY>     <FUNC DES>
                                 <VARIABLE* I O          <SUB VAR>   <LITCON>    <BOOL EXP>  <IMPL>        <DISJ>
                                 <CONJ>      <BOOL SEC>  <BOOL PRIM> <RELATION,

46    USE SUB 450 IF ST2P IS <ARITH EXP><SIMP AEXP> <TERM*          <FACTOR>   <PRIMARY>   <FUNC DES>  <VARIABLE>
                                 ID          <SUB VAR>   <LITCON>

47    USE SUB 451 IF ST2P IS <LIMPAIR>

48    USE SUB 414 IF ST2P IS <ARITH EXP>     <SIMP AEXP> <TERM>     <FACTOR>   <PRIMARY*   <FUNC DES>  <VARIABLE*
                                 10          <SUB VAR>   <LITCON>

49  ST1P+ROW(  <TYPE>     VALUE              / NEW T1

50  ST1P+ROW(  <TYPE>     ARRAY              / NEW T1

51    USE SUB 400 IF ST2P IS <IDLIST>    ID

52 EITHER
      USE SUB 452 IF ST2P IS <ID LIST*   ID
      USE SUB 453 IF ST2P IS <IO LIST>   ID
   AMBIGUITY IN SUB 52

53    USE SUB 454 IF ST2P IS <ID LIST,   IO

54    USE SUB 456 IF STOP IS<IDLIST>     ID
```

```
55   USE SUB 455 IF ST2P IS <ID LIST>   ID

56   USE SUB 458 IF ST2P IS <ID LIST>   ID

57   USE SUB 457 IF ST2P IS <ID LIST>   ID

58   USE SUB 459 IF ST2P IS <SPECIFIER>

59   USE SUB 460 IF ST2P IS IO

60   USE SUB 461 IF ST2P IS <EXPR>       <ARITH EXP> <SIMP AEXP> <TERM>      <FACTOR>    <PRIMARY>  <FUNC DES>
                            <VARIABLE>   ID          <SUB VAR>   <LITCON>    <BOOL EXP>  <IMPL>     <DISJ>
                            <CONJ>       <BOOL SEC>  <BOOL PRIM> <RELATION>

61   USE SUB 415 IF ST2P IS <LIM P LIST><LIM PAIR>

62   USE SUB 462 IF ST2P IS <LIM P LIST><LIM PAIR>

63   USE SUB 463 IF ST2P IS <LIM P LIST><LIM PAIR>

64   USE SUB 454 IF ST2P IS <NAME PART*

65   USE SUB 465 IF ST2P IS <NAME PART*

66 EITHER
     USE SUB 416 IF ST2P IS <SPECLIST> <SPECIFIER>
     USE SUB 466 IF ST2P IS <SPECLIST> <SPECIFIER>
   AMBIGUITY IN SUB 66

67   USE SUB 423 IF ST2P IS <ST LIST>    <STATEMENT><CL STATE>   <GO STATE>   <ASS STATE><PROC CALL><COMP ST>
                            <OP STATE>

68   USE SUB 467 IF ST2P IS <ST LIST,    <STATEMENT> <CL STATE>  <GO STATE>   <ASS STATE><PROC CALL*<COMP ST>
                            <OP STATE>

69   USE SUB 468 IF ST2P IS <NAME PART,

70   USE SUB 469 IF ST2P IS <STATEMENT><CL STATE>  <GO STATE>  <ASS STATE> <PROC CALL> <COMP ST>   <OP STATE>


71   USE SUB 470 IF ST2P IS <DECL>

72   USE SUB 471 IF ST2P IS <ARITH EXP><SIMP AEXP> <TERM>      <FACTOR*    <PRIMARY>   <FUNC DES>  <VARIABLE>
                            10           <SUB VAR>   <LITCON>

73   USE SUB 472 IF ST2P IS <ARITH EXP><SIMP AEXP><TERM>       <FACTOR>    <PRIMARY>   <FUNC DES>  <VARIABLE>
                            ID           <SUB VAR>   <LITCON>

74   USE SUB 473 IF ST2P IS <BOOL EXP>   <IMPL>      <DISJ>      <CONJ>      <BOOL SEC>  <BOOL PRIM><VARIABLE>
                            ID           <SUB VAR>   <FUNC DES>  <RELATION>

75   USE SUB 474 IF ST2P IS <FOR L EL*   <ARITH EXP> <SIMP AEXP> <TERM>      <FACTOR>    <PRIMARY>   <FUNC DES>
                            <VARIABLE>   ID          <SUB VAR>   <LITCON>
```

```
76     USE SUB 418 IF ST2P IS <ARITH EXP> <SIMP AEXP> <TERM>       <FACTOR>    <PRIMARY>   <FUNC DES>  <VARIABLE>
                              ID          <SUB VAR>  <LITCON>

77     USE SUB 419 IF ST2P IS <ARITH EXP> <SIMP AEXP> <TERM>       <FACTOR*    <PRIMARY>   <FUNC DES>  <VARIABLE>
                              IO          <SUB VAR>  <LITCON>

78     USE SUB 476 IF ST2P IS ID

79     USE SUB 475 IF ST2P IS ID

80     USE SUB 477 IF ST2P IS <ARITH EXP> <SIMP AEXP> <TERM>       <FACTOR>    <PRIMARY>   <FUNC DES>  <VARIABLE>
                              IO          <SUB VAR>  <LITCON>

81 EITHER
       USE SUB 478 IF ST2P IS <EXPR>      <ARITH EXP> <SIMP AEXP> <TERM>     <FACTOR>    <PRIMARY>   <FUNC DES>
                              <VARIABLE> IO           <SUB VAR>   <LITCON>   <BOOL EXP>  <IMPL>      <DISJ>
                              <CONJ>      <BOOL SEC>  <BOOL PRIM> <RELATION>
       USE SUB 479 IF STEP IS <ASS STATE>

82 EITHER
       USE SUB 417 If STEP IS <DECLIST>   <DECL>
       USE SUB 481 If STEP IS <DECLIST>   <DECL>
       USE SUB 423 IF ST2P IS <ST LIST>   <STATEMENT> <CL STATE>  <GO STATE>  <ASS STATE> <PROC CALL> <COMP ST>
                              <OP STATE*
   AMBIGUITY IN SUB 82

83     USE SUB 480 IF ST2P IS <ST LIST*   <STATEMENT> <CL STATE>  <GO STATE>  <ASS STATE> <PROC CALL> <COMP ST>
                              <OP STATE>

84     USE SUB 482 IF ST2P IS <ST LIST>   <STATEMENT> <CL STATE>  <GO STATE>  <ASS STATE> <PROC CALL> <COMP ST>
                              <OP STATE>

85      P+P-1 /    ST2P+<CL STATE>     FOR PRODUCTION <CL STATE>    ::=  ID          :

86     USE SUB 483 IF ST2P IS <CL STATE>  <GO STATE>  <ASS STATE> <PROC CALL> <COMP ST>

87 EITHER
       USE SUB 483 IF ST2P IS <CL STATE>  <GO STATE> <ASS STATE> <PROC CALL> <COMP ST>
       USE SUB 484 IF ST2P IS <OP STATE>

88     USE SUB 495 IF ST2P IS IO

89     USE SUB 420 IF ST2P IS <FOR LIST>  <FOR L EL>  <ARITH EXP> <SIMP AEXP> <TERM>       <FACTOR>    <PRIMARY>
                              <FUNC DES>  <VARIABLE>  ID          <SUB VAR>   <LITCON>

90     USE SUB 486 IF ST2P IS <FOR LIST*  <FOR L EL>  <ARITH EXP> <SIMP AEXP> <TERM>       <FACTOR>    <PRIMARY>
                              <FUNC DES>  <VARIABLE>  ID          <SUB VAR>   <LITCON>

91     USE SUB 487 IF ST2P IS <CL STATE>  <GO STATE>  <ASS STATE> <PROC CALL> <COMP ST>

92 EITHER
       USE SUB 487 IF ST2P IS <CL STATE'  <GO STATE>  <ASS STATE* <PROC CALL> <COMP ST>
       USE SUB 488 IF ST2P IS <OP STATE*

93     USE SUB 489 IF ST2P IS <CL STATE>  <GO STATE>  <ASS STATE> <PROC CALL> <COMP ST>
```

```
94 EITHER
      USE SUB 499 If ST2P IS <CL STATE>  <GO STATE>  <ASS STATE> <PROC CALL> <COMP ST>
      USE SUB 490 If ST2P IS <OP STATE>

95    USE SUB 491 If ST2P IS <STATEMENT> <CL STATE>  <Go STATE>  <ASS STATE> <PROC CALL> <COMP ST>   <OP STATE>

96    USE SUB 492 If ST2P IS <STATEMENT> <CL STATE>  <GO STATE>  <ASS STATE> <PROC CALL> <COMP ST>   <OP STATE>

400  P+P+1      ST1P+ROW(  <ID  LIST>  ,            )     /  NEW  T1
401  P+P+1      ST1P+ROW(  10          [            )     /  NEW  T1
402  P+P+1      ST1P+ROW(  ID          (            )     /  NEW  T1
403  P+P+1      ST1P+ROW(  <PRIMARY>  **            )     /  NEW  T1
404  P+P+1      ST1P+ROW(  <TERM>      <* O R />    )     /  NEW  T1
405  P+P+1      ST1P+ROW(  <SIMP AEXP> <+ O  R ->   )     /  NEW  T1
406  P+P+1      ST1P+ROW(  <AEXP LIST> ,            )     /  NEW  T1
007  P+P+1      ST1P+ROW(  <ARITH EXP> <REL OP>     )     /  NEW  TI
408  P+P+1      ST1P+ROW(  <RELATION>  <REL OP>     )     /  NEW  T1
409  P+P+1      ST1P+ROW(  <CONJ>      AND          )     /  NEW  T1
410  P+P+1      ST1P+ROW(  <DISJ>      OR           )     /  NEW  T1
411  P+P+1      ST1P+ROW(  <IMPL>      IMPLIES      )     /  NEW  T1
412  P+P+1      ST1P+ROW(  <BOOL  EXP> EQUIV        )     /  NEW  T1
413  P+P+1      ST1P+ROW(  <EXP LIST>  .            )     /  NEW  T1
414  P+P+1      ST1P+ROW(  <ARITH EXP> :            )     /  NEW  T1
415  P+P+1      ST1P+ROW(  <LIM P LIST>,            )     /  NEW  T1
416  P+P+1      ST1P+ROW(  <SPEC LIST> ;            )     /  NEW  T1
417  P+P+1      ST1P+ROW(  <DEC  LIST> ;            )     /  NEW  TI
418  P+P+1      ST1P+ROW(  <ARITHEXP> STEP          )     /  NEW  T1
419  P+P+1      ST1P+ROW(  <ARITHEXP> WHILE         )     /  NEW  T1
420  P+P+1      ST1P+ROW(  <FOR  LIST>  ,           )     /  NEW  T1
421  P+P+1      ST1P+ROW(  <VARIABLE>:=             )     /  NEW  TI
```

```
422   P+P+1 /  ST1P+ROW(I  D          ;            )        / NEW  T1
423   P+P+1 /  ST1P+ROW(  <ST  LIST*  ;            )        / NEW  T1
424  P+P-1  /  ST2P+<ID LIST*      FOR PRODUCTION <ID LIST>     ::=  <ID   LIST> ,         ID
425  P+P-1  /  ST2P+<SUB VAR>      FOR PRODUCTIOh <SUBVAR>      ::=  ID          [         <AEXP LIST> ]
       NEW T1
426  P+P-1  /  ST2P+<FUNC DES>     FOR PRODUCTIOh CFUNC DES>    ::=  ID                    <EXP LIST,)
       NEW T1
427  P+P-1  /  ST2P+<NAME PART>    FOR PRODUCTIOh CNAME PART>   ::=  ID          (         <ID  LIST>  )
       NEW TI
428  P+P-1  /  ST2P+<PROC CALL,    FOR PRODUCTION <PROCCALL>    ::=  ID                    <EXP  LIST>  )
       NEW T1
429  P+P-1  /  ST2P+<PRIMARY>      FOR PRODUCTIOh <PRIMARY>     ::=  (           <ARITH EXP> )             / NEW T1
430  P+P-1  /  ST2P+<BOOL PRIM>    FOR PRODUCTION<BOOL PRIM>    ::=              <BOOL EXP>  )             / NEW T1
431  P+P-1  /  ST2P+<FACTOR>       FOR PRODUCTION<FACTOR>       ::=  <PRIMARY>*  *         <FACTOR>
432  P+P-1  /  ST2P+<FACTOR>       FOR PRODUCTION <FACTOR*      ::=  @           <FACTOR>
433  P+P-1  /  SY2P+<FACTOR>       FOR PRODUCTIOK <FACTOR>      ::=  <+ O R ->   <FACTOR>
434  P+P-1  /  ST2P+<TERM>         FOR PRODUCTIOh <TERM>        ::=  <TERM>      <* OR />  <FACTOR>
435  P+P-1  /  ST2P+<SIMP AEXP>    FOR PRODUCTION<SIMP AEXP>    ::=  <SIMP AEXP><+ O  R -> <TERM>
436 ST1P+ROW(I   F        <BOOL EXP>  THEN          )        / NEW  T1
4 3 7 ST1P+ROW(IF         <BOOL EXP>  THEN       <ARITHEXP> ELSE        )      / NEW  T1
4 3 8 ST1P+ROW(I  F       <BOOL EXP>  THEN       <CL STATE* ELSE        )      / NEW  T1
439   P+P-1 /  ST2P+<OP STATE>     FOR PRODUCTION <OP STATE>    ::=  IF          <BOOL EXP>  THEN      <STATEMENT>
440   P+P-1  / ST2P+<ARITH EXP>    FOR PRODUCTION <ARITHEXP>    ::=  IF          <BOOL EXP>  THEN      <ARITH EXP>
     ELSE      <ARITH EXP>
441  P+P-1  /  ST2P+<AEXP LIST>    FOR PRODUCTION<AEXP LIST>    ::=  <AEXP LIST>,          <ARITH EXP>
442   P+P-1 /  ST2P+<RELATION>     FOR PRODUCTION<RELATION>     ::=  <ARITH EXP> <REL OP>  <ARITH EXP>
443   P+P-1 /  ST2P+<RELATION>     FOR PRODUCTION <RELATION>    ::=  <RELATION>  <REL OP>  <ARITH EXP>
444   P+P-1 /  ST2P+<BOOL SEC>     FOR PRODUCTION<BOOL SEC>     ::=  NOT         <BOOL PRIM>
445   P+P-1 /  ST2P+<CONJ>         FOR PRODUCTION<CONJ>         ::=  <CONJ>      AND       <BOOL SEC>
```

```
446  P←P-1   /   ST2P←<DISJ>          FOR PROCUCTIOh <DISJ>        ::=  <DISJ>      OR         <CONJ>
447  P←P-1   /   ST2P←<IMPL>          FOR PROCUCTIOh <IMPL>        ::=  <IMPL>      IMPLIES    <DISJ>
448  P←P-1   /   ST2P←<BOOL EXP>      FOR PROCUCTIOh <BOOL EXP>    ::=  <BOOL EXP>  EQUIV      <IMPL>
449  P←P-1   /   ST2P←<EXP LIST>      FOR PRODUCTION <EXP LIST>    ::=  <EXP LIST>  ,          <EXPR>
450  P←P-1   /   ST2P←<LIM PAIR>      FOR PRODUCTION <LIM PAIR>    ::=  <ARITH EXP> :          <ARITH EXP>
451  P←P-1   /   ST2P←<LIM p LIST>    FOR PRODUCTION <LIM p LIST>  ::=  <LIM p LIST>,          <LIM PAIR>
452  P←P-1   /   ST2P←<SPECIFIER>     FOR PRODUCTION <SPECIFIER>   ::=  <TYPE>      <ID LIST>
453  P←P-1   /   ST2P←<DECL>          FOR PROCUCTIOh <DECL>        ::=  <TYPE>      <ID LIST>
454  P←P-1   /   ST2P←<SPECIFIER>     FOR PRODUCTION <SPECIFIER>   ::=  <TYPE>      VALUE      <ID LIST>
455  P←P-1   /   ST2P←<SPECIFIER>     FOR PRODUCTION <SPECIFIER>   ::=  <TYPE>      ARRAY      <ID LIST>
456  ST1P←ROW(  <TYPE>     ARRAY      <ID   LIST>   [           )       /  NEW  T1
457  P←P-1   /   ST2P←<SPECIFIER>     FOR PROCUCTIOh <SPECIFIER>   ::=  ARRAY       <ID LIST>
458  ST1P←ROW(  ARRAY      <ID   LIST>   [           )       /  NEW  T1
459  P←P-1   /   ST2P←<SPEC LIST>     FOR PRODUCTION <SPEC LIST>   ::=  <SPEC LIST> ;          <SPECIFIER>
460  ST1P←ROW(  CONSTANT IO       :=           )       /  NEW  T1
461  P←P-1       ST2P←<DECL>          FOR PROCUCTIOh <DECL>        ::=  CONSTANT    IO         :=         <EXPR>

462  P←P-1   /   ST2P←<DECL>          FOR PROCUCTIOh <DECL>        ::=  <TYPE>      ARRAY      <ID   LIST>   [
     <LIM P LIST>]          /  NEW  T1
463  P←P-1       ST2P←<DECL>          FOR PROOUCTIOh <DECL>        ::=  ARRAY       <ID LIST>  [          <LIM P LIST>
     ]          /  NEW T1
464  P←P-1       ST2P←<DECL>          FOR PRODUCTION <DECL>        ::=  SWITCH      <NAME PART>
465  ST1P←ROW(  PROCEDURE   <NAME PART> ;          )       /  NEW  T1
466  ST1P←ROW(  PROCEDURE   <NAME PART> ;        <SPEC LIST> ;          )       /  NEW T1
467  P←P-1       ST2P←<DECL>          FOR PRODUCTION <DECL>        ::=  PROCEDURE   <NAME PART> ;          <SPEC LIST>
     ;           <ST LIST>   END          /  NEW  TI
468  ST1P←ROW(  FUNCTION    <NAME PART> :          )       /  NEW  T1
469  P←P-1   /   ST2P←<DECL>          FOR PRODUCTION <DECL>        ::=  FUNCTION    <NAME PART> :          <STATEMENT>

470  P←P-1       ST2P←<DEC LIST>      FOR PROCUCTIOh <DEC LIST,    ::=  <DEC LIST>             <DECL>
```

```
471  ST1P←ROW(  <ARITH EXP> STEP        <ARITH EXP> UNTIL          )          /  NEW  T1
472    P←P-1 /   ST2P←<FOR L EL>     FOR PRODUCTION <FOR L EL>    ::=  <ARITH EXP> STEP        <ARITH EXP> UNTIL
     <ARITH EXP>
473   P←P-1   /   ST2P←<FOR L EL>     FOR PROCUCTIOh <FOR L EL>    ::=  <ARITH EXP> WHILE       <BOOL EXP>
474  P←P-1    /   ST2P←<FOR LIST>     FOR PRODUCTION <FOR LIST>  ::=  <FOR LIST> ,             <FOR L ELS
475    P←P-1 /    ST2P←<GO STATE,     FOR PRODUCTION <GO STATE>    ::=  GO TO        10
476  ST1P←ROW(  GO TO        10         [               )          /  NEW  T1
477    P←P-1 /    ST2P←<GO STATE>     FOR PROCUCTIOK <GO STATE>    ::=  GO TO        10           [              <ARITH EXP>
     I               /  NEW  T1
478  P←P-1        ST2P←<ASS STATE>    FOR PRODUCTION <ASS STATE>   ::=  *VARIABLE>   :=           <EXPR>
479  P←P-1        ST2P←<ASS STATE>    FOR PRODUCTION <ASS STATE>   ::=  <VARIABLE>   :=           <ASS STATE,
480  P←P-1        ST2P←<COMP S T*     FOR PROCUCTIOh <COMP ST>     ::=  BEGIN        <ST LIST>    END             /  NEW T1

481  ST1P←ROW(  BEGIN        <DEC LIST> ;                          /  NEW  T1
482  P←P-1    /   ST2P←<COMP ST>      FOR PRODUCTION <COMP ST>     ::=  BEGIN        <DEC LIST> ;              <ST LIST>
     END             /  NEW  T1
483  P←P-1    /   ST2P←<CL STATE>     FOR PRODUCTION <CL STATE>    ::=  10           :            <CL STATE>
484  P←P-1    /   ST2P←<OP STATE>     FOR PRODUCTIon cop STATE>    ::=  IO           :            <OP STATE*
485  ST1P←ROW(  FOR          10          :=              )          /  NEW  T1
486  ST1P←ROW(  FOR          10          :=           <FOR LIST>  cc          )        /  NEW  T1
487  P←P-1    /   ST2P←<CL STATE>     FOR PRODUCTION <CL STATE>    ::=  FOR          10           :=             <FOR LIST>
     co           <CL STATE>
488  P←P-1    /   ST2P←<OP STATE>     FOR PRODUcTION <OP STATE>    ::=  FOR          10           :=             <FOR LIST>
     DO           COP STATE,
489  P←P-1    /   ST2P←<CL STATE,     FOR PRODUCTION <CL STATE,    ::=  IF           <BOOL EXP>  THEN           <CL STATE>
     ELSE         <CL STATE>
490  P←P-1    /   ST2P←<OP STATE>     FOR PRODUCTION <OP STATE>    ::=  IF           <BOOL EXP>  THEN           <CL STATE>
     ELSE         <DP STATE>
491  P←P-1    /   ST2P←<ST LIST>      FOR PRODUCTION <ST LIST>     ::=  <ST LIST>    ;            <STATEMENT>
492  P←P-1    /   ST2P←<PROGRAM>      FOR PRODUCTION <PROGRAM>     ::=  PHI          <STATEMENT> PHI             /  NEW  T1
```