ZASTOSOWANIA MATEMATYKI
APPLICATIONES MATHEMATICAE
XI, 1 (1969)

J. SZCZEPKOWICZ (Wrocław)

ON TABLE-DRIVEN SYNTAX-CHECKING WITHIN ON ALGOL COMPILER

CONTENTS

1. Introduction

1.1. Summary of the paper. In this paper we describe some machine-independent and, in a sense, language independent techniques used in the ODRA-ALGOL compiler which has been developed at the Department of Numerical Methods of Wrocław University and now (July 1968) is used at some ODRA 1204 installations.

The problem of translating the input ALGOL text into the machine code of the ODRA 1204 computer is solved in a three-pass process, where each pass consists mainly in a single scan through the input text, with no back-up nor look-ahead.

The first two passes, which perform book-keeping, a complete syntax-check and a transformation of the source programme, are controlled using a compactified form of Reducing Transition Tables (RTT, see [1] for definition) constructed for two Reducing Formal Languages (RFL) chosen so as to yield a big subset of ALGOL 60, called ODRA-ALGOL. The paper contains the complete syntactic descriptions of the two RFL's together with the relevant tables in a legible form. The use of these tables is described in detail and some information is given on the preliminary transformations of the source programme and on how to provide a fairly complete semantic check during syntax-checking. Storage and time problems are also mentioned.

The third pass of the compiler, which generates the final machine coding, will be described in a separate publication on the ODRA-ALGOL compiler and is not discussed here.

1.2. Originality. In the view of the available publications it seems to the author that this paper contains the following new elements:

a. a useful (at least for the ODRA-ALGOL compiler) solution of the problem of non-reducibility of ALGOL 60 (see [1], [8]);

b. the reducing and unambiguous formal grammar of ODRA-ALGOL, which contains much of the informal semantics of ALGOL 60 (i.e. compatibility of operands);

c. a method of efficient storage of Reducing Transition Tables, which decreases considerably the amount of the necessary core store;

d. an approach to the compiler design, which makes it possible to implement complete syntax-checking and fast compilation on a small machine configuration, i.e. one with 16K 24-bit word core store and no backing storage; nevertheless there are no intermediate external outputs and the compiler is able to deal with programmes of its own object size.

Finally it should be mentioned that so far there is no publication containing reducing transition tables ready to be used in an ALGOL compiler, or even some practical remarks on their use.

1.3. Some remarks on an existing solution. To check a formal text using a RTT as the basic source of control information, a reducing and unambiguous grammar of the language has to be found (see [1]). If such a grammar does not exist, the associated RTT is either undefined or it is useless for a simple checking scheme. ALGOL 60, defined as in [5], is not a RFL since its grammar contains formulae such as

⟨array identifier⟩:: = ⟨identifier⟩
⟨procedure identifier⟩:: = ⟨identifier⟩
⟨label⟩:: = ⟨identifier⟩
. . . .
⟨subscript expression⟩:: = ⟨arithmetic expression⟩
⟨for list element⟩:: = ⟨arithmetic expression⟩
⟨expression⟩:: = ⟨arithmetic expression⟩
⟨lower bound⟩:: = ⟨arithmetic expression⟩

and so on. In connection with this M. Paul writes in [8]: "This means that in ALGOL, ⟨identifier⟩ corresponds to several syntactical elements. One may, however, identify the elements ⟨array identifier⟩ with ⟨procedure identifier⟩, and similarly certain other elements may be identified with each other so that ALGOL becomes a reducing formal language. This process of identification weakens the structure of ALGOL...".

Although this description is far from being complete, it implies that the syntax-checking process based on the "weakened" grammar of ALGOL is not so complete as required in a real ALGOL compiler, and thus a serious special mechanism is necessary to check, for instance, the proper use of various kinds of identifiers. Such a special mechanism is not required in the ODRA-ALGOL compiler, since all its functions, with operand compatibility check included, are performed by the simple syntax-checking routine described in § 2.4.1. The solution presented here goes in an opposite direction than that of Paul's: instead of weakening the structure of ALGOL, it is strengthened by discarding some too detailed formulae of the grammar, e.g. those with right parts equal to ⟨identifier⟩, and putting more of the informal semantics of ALGOL 60 into the formal syntax, e.g. including operand compatibility in the syntactical formulae.

Another problem, not mentioned in available publications and of a different kind of difficulty, is unambiguity (uniqueness) of the resulting formal language; the problem consists in defining the grammar in such a way that for a given correct text there is exactly one canonical reduction sequence (see [1] for definition). It is known that ALGOL 60 contains some ambiguities. Futher discussion on this topic will be found in § 3.2; the complete solution is given on pages 50-53.

The approach presented here simplifies considerably the design of the compiler. About a half of it consists of computer-generated tables which require no checking, and the remaining part includes a number of relatively simple routines.

1.4. The problem of generating sizeable RTT's. The algorithm for checking a Formal Language (FL) for unambiguity and generation of the RTT is given in [1]. Generation of the RTT for a real language, however, is a tremendous task which is liable to error. To cope with the problem we have programmed the algorithm described in [1] for an Elliott 803 computer. The programme reads the formal grammar of a FL, checks it for reducibility and unambiguity and then outputs the RTT in a form ready to be used to check formal texts written in the language whose grammar was originally fed into the machine. The syntactic descriptions of RFL's given on pages 43, 44, 50-53 are examples of the data for that programme.

There is also another programme to transform the RTT into a Compactified RTT (CRTT). The method of compactification may be seen by inspecting [1] and § 2.4.1 of the present paper. To give an idea of the efficiency of the method it is sufficient to state that the RTT for the FL of our pass 2 occupies 216 444 binary digits while the compactified version of it, which is used in the ODRA-ALGOL compiler, requires 67 056 binary digits.

To transform grammars into CRTT's ready to be used in the compiler, an Elliott 803 computer with 8K core store and two magnetic tapes had to be run about 12 hours. The magnetic tapes are essential for a reasonable speed of the programme; a version of it which uses no tapes had to be run about a week. Further details on the table generating programmes are available at Wrocław University.

1.5. An unsolved minimization problem. A RFLn is understood to be a reducing formal language with a maximum length of syntactic formulae equal to n, i.e. the longest syntactic formula in the grammar of a RFLn is of the form

$$\langle \text{SE} \rangle ::= \langle \text{SE1} \rangle \langle \text{SE2} \rangle \ldots \langle \text{SE}n \rangle$$

where SE stands for a name of a syntactic element of the RFLn. It is emphasised that a definition such as

$$\langle \text{identifier} \rangle ::= \langle \text{letter} \rangle \mid \langle \text{identifier} \rangle \langle \text{letter} \rangle \mid \langle \text{identifier} \rangle \langle \text{digit} \rangle$$

should be considered as an abbreviated form of three syntactic formulae, namely

$$\langle \text{identifier} \rangle ::= \langle \text{letter} \rangle$$

$\langle$identifier$\rangle$:: = $\langle$identifier$\rangle\langle$letter$\rangle$
$\langle$identifier$\rangle$:: = $\langle$identifier$\rangle\langle$digit$\rangle$

The abbreviation consists in writing a repeated left part once.

The algorithm described in [1] works only on a RFL2, but this fact does not affect the generality of the algorithm by virtue of a theorem of Paul (see [8]). Roughly speaking, the theorem states that any RFLn ($n > 2$) can be embedded into a RFL2 in such a way that the alphabet of the RFLn is identical with that of the RFL2 and a text, i.e. a string of alphabet elements, which is correct in the sense of the grammar of the RFLn is also correct in the sense of the grammar of the RFL2.

There may be, however, many RFL2's which contain a given RFLn in the sense of Paul's theorem. We shall give an example. In [1] the following RFL3 grammar is considered (the example is re-worded to be more obvious):

f1: $\langle$ifc$\rangle$:: = $\langle$if$\rangle\langle$be$\rangle\langle$then$\rangle$
f2: $\langle$ifs$\rangle$:: = $\langle$ifc$\rangle\langle$us$\rangle$
f3: $\langle$cs$\rangle$:: = $\langle$ifs$\rangle$ | $\langle$ifs$\rangle\langle$else$\rangle\langle$s$\rangle$
f4: $\langle$s$\rangle$:: = $\langle$us$\rangle$ | $\langle$cs$\rangle$

The leftmost symbols in each of the above four lines are evidently labels used for identification purposes. To generate the RTT the authors of [1] map this RFL3 into a RFL2 by replacing f1 and f3 by the following:

f11: $\langle$ifc$\rangle$:: = $\langle$if and be$\rangle\langle$then$\rangle$
f12: $\langle$if and be$\rangle$:: = $\langle$if$\rangle\langle$be$\rangle$
f31: $\langle$cs$\rangle$:: = $\langle$ifs$\rangle$ | $\langle$ifs and else$\rangle\langle$s$\rangle$
f32: $\langle$ifs and else$\rangle$:: = $\langle$ifs$\rangle\langle$else$\rangle$

The resulting RTT consists of 25 rows, as given by the authors of [1] and checked by our programme. We have verified that if f31 and f32 are replaced by

ff31: $\langle$cs$\rangle$:: = $\langle$ifs$\rangle$ | $\langle$ifs$\rangle\langle$else and s$\rangle$
ff32: $\langle$else and s$\rangle$:: = $\langle$else$\rangle\langle$s$\rangle$

then the RTT consists of 22 rows. Some experiments on the computer showed us that this difference is by no means negligible when dealing with a non-trivial language. Unfortunately, those experiments gave us only some intuitive rules which we obeyed when designing ODRA-ALGOL internal specifications. The problem of how to choose the necessary RFL2 so as to keep low the size of the RTT is surely worth some theoretical investigation.

1.6. A brief characteristics of the ODRA 1204 computer. Designer and manufacturer: Wrocławskie Zakłady Elektroniczne ELWRO, Wrocław, Poland.

General organization: Single-address parallel machine with the logic stored in a read-only memory.

Main storage and word structure: 16K core storage, extendable to 64K, with access time 6 microseconds.

A 24-bit machine word contains one single-address machine instruction or a 24-bit fixed-point number. A double word of 48 bits occupies two consecutive locations. There are machine instructions to manipulate a double word either as a double-length fixed-point number or a floating-point number with 38-bit mantissa and 10-bit exponent.

Addressing facilities: A block of 16K locations is directly addressable (14-bit address). Any kind of instruction modification provides a 16-bit address There are 3 normal index registers and indirect addressing is provided; indexing and indirect addressing are recursive by hardware. There is also a set of instructions to modify the next instruction by the content of any location or the accumulator.

Organization of peripherals: Peripherals are connected with the central processing unit by means of a standard interface system. The central unit can control simultaneously 7 communication channels and there may be up to 8 peripheral devices in each channel. During peripheral transfers the central unit may operate at full speed. An interruption system provides a facility for parallel running of several programmes. The machine is normally operated under the control of a supervisory routine.

Basic input: 5 or 8 hole paper tape read at 1500 char/sec and the control typewriter.

Basic output: 5 or 8 hole paper tape punched at 150 char/sec and the control typewriter.

Operation times (in microseconds):

Control operations	10-21
Basic fixed-point	15-25
Floating-point addition	100 (approx.)
Floating-point multiplication	280 (approx.)
Floating-point division	1000 (approx.)

Special facilities: There are also table look-up, table-shift, and table-exchange operations performed by the hardware several times faster than by the corresponding programmed loops. We make extensive use of these operations in the ODRA-ALGOL compiler.

1.7. Some features of the ODRA-ALGOL compiler. In this section we describe some features of the ODRA-ALGOL compiler, which express a kind of what is called "the philosophy of compiler design".

a. The compiler is able to compile programmes of its own size on a minimum machine configuration, i.e. one with 16K core store and no backing storage. Intermediate external outputs are not allowed for.

b. It is made impossible for a programme which has been accepted and compiled to go out of control, or to overwrite something not intended to, due to data errors or to programming errors.

c. Apart of complete syntax-checking, the compiler provides an extensive semantic check performed at translation time. The features impossible to check at translation time (e.g. whether a type procedure, left normally, is assigned a value) are checked at run time of the translated programme.

d. No effort is made to allow the compiler to continue checking after revealing an error. In our opinion it is better to give a small but undoubtedly true information than a huge list with strange information caused merely by misinterpretation of the text following the error.

e. The compiler does not try to correct even the most clerical errors in the source programme.

f. The compiler deals properly with the most powerful concepts of ALGOL 60 such as the call by name and recursion; these are often the only tools to get the job easily done.

g. The compiler itself does not contain any recursive routines.

h. The compiler does not try such actions as "removing common subexpressions". The actions of such a type usually provide a premium for lazy programmers and, on the other hand, make useless the work of those who design their programmes in a special way to achieve better numerical accuracy.

i. The system of standard procedures, designed primarily with a certain machine configuration in mind, is capable of being updated without any changes in the logic of the compiler.

j. The compiler is capable of being embedded in any reasonable operating system. In fact, there are only about 50 instructions depending on the operating system. These refer to peripheral devices.

1.8. General organization of the ODRA-ALGOL compiler. The compiler is a sequence of three co-routines called pass 1, pass 2 and pass 3 respectively. Pass 1 reads the paper tape containing an ALGOL text and generates identifier descriptions tagged together with the block structure description. The output from pass 1 is packed in the store and is used as the input data for pass 2. Pass 2 performs the complete syntax-

-check and a transformation of the programme, i.e. delimiter replacement and insertion. Pass 2 is allowed, when necessary, to overwrite the body of pass 1. The output from pass 2, packed just as that from pass 1, serves as data for pass 3 which is allowed, when necessary, to overwrite the bodies of pass 1 and pass 2. The output from pass 3 is the machine code programme ready to be obeyed to execute the algorithm described by the original ALGOL text.

After translating a medium-sized programme the compiler can be used again without any reinput, while big ALGOL texts usually cause the compiler to be overwritten almost entirely.

Such an organization makes it possible to implement (a) of § 1.7, as the size of the information on the programme slowly grows up in each pass, while the size of the necessary part of the compiler rapidly decreases after each pass is completed.

2. Organization of pass 1.

2.1. Main objectives. Pass 1 has two main objectives. The first one is to store in a compact form the meaningful portions of the input ALGOL text. The second one is to generate some lists to describe the block structure of the programme and the nature of the declared (specified) identifiers. The final goal is to make possible for pass 2 to deal with the text as if it were a string of alphabet elements of a certain reducing formal language (see [1] and [8] for definitions).

2.2. On a method of compact storage of ALGOL texts. Each input text processed by the ODRA-ALGOL compiler is considered as a sequence of syllables. A syllable is understood to be any of the following ALGOL constituents:

(0) a delimiter,
(1) an identifier,
(2) a constant (arithmetic or Boolean),
(3) a string.

The number on the left in each of the above four lines is said to be Category Number (CN) of the corresponding ALGOL constituent. Syllables are extracted from the text by a set of relatively simple routines called microanalysers which also delete the various kinds of comments. Another function of each microanalyser is to define the Identification Number (IN) for each syllable. The meaning of IN depends on the value of the associated CN and is described below.

CN = 0. If the syllable is a delimiter then IN is equal to a small integer which is the internal representation of that delimiter.

CN = 1. When pass 1 is working, the Identifier List (IL) is formed. At the beginning the IL contains only the identifiers for standard procedures. If an identifier is formed, a look-up in the IL is performed and if there is no such an identifier, the IL is extended by the new identifier. In any case the relative address of the identifier in the IL is calculated and this is the IN for the identifier in question.

It seems that such a method fails to distinguish between two distinct meanings of the identifier I in the following piece of an ALGOL text:

begin **real** I;

procedure $P(I)$; **switch** I;

It is not the case as those two meanings have to be distinguished by the block structure description (see § 2.3) and not by their graphical representations.

The built-in look-up facilities of the ODRA 1204 computer are used extensively in calculating IN's.

CN = 2. The list of constants (CL) is operated similarly as the IL. The difference consists in that the CL is empty at the beginning. The IN for a constant is a linear function of the relative address of the constant in the CL. IN = 0 and IN = 1 are reserved for the Boolean constants **false** and **true** respectively.

CN = 3. Two ALGOL strings are said to be identical if they consist of identical sequences of characters. The String List (SL) is formed just as the CL, i.e. it is empty at the beginning and each distinct string is stored once only. The IN for a string is the relative address of the beginning of that string in the SL.

Since each IN is a relative address (for CN $\neq 0$), IL, CL and SL are relocatable at translation time. This is an invaluable feature when an efficient use of the storage space should be made.

In the ODRA-ALGOL compiler each IN is subjected to the condition

$$0 \leqslant \mathrm{IN} \leqslant 1023$$

so that each IN could be stored as a ten-digit binary number.

Now we define the Syllable Representation (SR). If SYL is a syllable, CN and IN are its category number and identification number respectively, then the SR of SYL, denoted by SR(SYL), is given by the equality

$$\mathrm{SR(SYL)} = 4 \times \mathrm{IN} + \mathrm{CN}$$

It follows from the definition of CN and the restriction imposed on each IN that for a given SYL the representation SR(SYL) is a uniquely determined at most twelve-bit integer.

Now the method of storage is stated explicitly. Given a sequence of syllables

$$\mathrm{SYL1}, \mathrm{SYL2}, \ldots, \mathrm{SYL}n$$

representing an ALGOL algorithm. To store it, the lists IL, CL and SL are generated (this has to be done in some form in each compiler) and the algorithm is represented by a sequence of syllable representations

$$\mathrm{SR(SYL1)}, \mathrm{SR(SYL2)}, \ldots, \mathrm{SR(SYL}n)$$

The restrictions imposed on IN's enables us to pack two syllable representations in one 24-bit word of the ODRA 1204 computer.

It could be observed that the syllable representations usually occupy less storage space than the corresponding machine coding. This is valuable for storage administration, by the following simple facts:

(a) each pass performs a single scan with no back-up nor look-ahead,

(b) if, in a certain pass, a piece of the input list is taken, then this piece is either stacked or its processed equivalent is sent to the output; in any case it is deleted in the input list.

Now, if there would be no room for an intermediate version of the text then there would neither be room for the machine coding, not to mention object variables and arrays.

To give an idea of the relation between the input text and the corresponding machine coding, we shall consider an example. The assignment statement

$$x := \textbf{if} \neg B1 \wedge B2 \textbf{ then } sin(x+y) \textbf{ else } tan(x);$$

consists of 20 syllables. The corresponding syllable representations occupy 10 locations, while the machine code generated by pass 3 will be exactly as given below:

```
    Take B1;
    Negate accumulator;
    Collate B2;
    Jump if false to L1;
    Take floating x;
    Add floating y;
    Call subroutine for sine;
    Jump to L2;
L1: Take floating x;
    Call subroutine for tangent;
L2: Store in x;
```

The coding occupies 11 locations. It should be noted, however, that not all ALGOL statements are translated so efficiently. The object code would be longer if *sin* and *tan* were replaced by non-standard functions, or if *x* and *y* were formal parameters called by name, or both, while the sequence of syllable representations still occupies the same amount of storage space.

This is not the case when considering some perfectly valid, although somewhat unlikely, programmes such as that given below:

begin *print* ((((((((((*3.14*)))))))))) **end**

which will be stored in 12 locations, while the whole object programme generated by pass 3 will be:

Call start subroutine;
Take constant *3.14*;
Call print subroutine;
Jump to finish routine;

which requires 4 locations for the programme and 2 locations for the constant *3.14*.

Finally it is noted that the forementioned method of storage is immediately applicable to any other machine with another word length. For this purpose it is sufficient to change appropriately the restriction imposed on the identification numbers, or the number of syllable representations packed in one location, or both.

2.3. On a method of describing the block structure of an ALGOL algorithm. An ingenious technique of describing the block structure of a programme is given in [3]. The ODRA-ALGOL compiler uses it in a slightly modified form which is better suited to the ODRA 1204 computer. As we shall often have to refer to the block structure description, the topic is discussed here in detail by means of some more rigorous terminology than that used in [3].

Define Nomenclature Level (NL). In the ODRA-ALGOL compiler a NL is understood to be any of the following ALGOL constituents: a block, a procedure declaration, a for statement.

The purpose of defining a for statement to be a new NL is to provide an automatic test for jumps leading into the statement controlled by means of a for clause. Although in [5], 4.66, it is stated that such a jump is undefined, we have to check it unless the object programme is allowed to go out of control (see b in § 1.7). This is due to the fact that the statement controlled by a for clause is coded as a subroutine which is called from within the various for list elements contained in the for clause.

A jump into the controlled statement from outside fails to define the subroutine link, with the well-known results.

To describe the block structure, two lists are generated. The first list, called the Identifier Description List (IDL), contains almost complete information about each identifier which has been declared or specified. The description of an identifier contains usually its identification number (see § 2.2), type, the method of calculating the associated address and so on. The detailed organization of this list is highly machine-dependent. For the purpose of the present paper we shall denote the description of an identifier I by D(I).

To describe the second list, called the Nomenclature Level List (NLL), we have to define the Nomenclature Level Number (NLN). The NLN is set to zero before reading the first symbol of an ALGOL text, and it is increased by one each time a new NL is encountered.

The NLL has to provide means for looking-up the IDL in the appropriate order. Each element of the NLL consists of two numbers, namely the Surrounding Nomenclature Level Number (SNLN) and the Identifier Description List Pointer (IDLP). The IDLP is the relative address indicating the beginning of the associated portion of the IDL. If the portion is empty then IDLP = 0. This happens when the NL is either a parameterless procedure with no label belonging to that NL, or it is a for statement with the same feature.

To illustrate the block structure description, we shall consider an ALGOL programme taken from [3].

```
begin
  integer procedure B(C);
    integer C; B: = C/A;
  integer A, C, I;
  A: = C: = 2;
  for I: = 1 step 1 until C do
    LABE: begin
              integer E;
              E: = B(I)+A; A: = E
            end;
  for I: = 1 step 1 until C do print (I, A/I)
end
```

The two lists, the NLL and the IDL are given below in an obvious symbolic form. I1, I2, ..., In denote the identifiers for standard procedures.

Nomenclature Level List

NLN	SNLN	IDLP
0	0	Standard
1	0	IDLP1
2	1	IDLP2
3	1	IDLP3
4	3	IDLP4
5	1	0

Identifier Description List

IDLP2:	D(C)
IDLP4:	D(E)
IDLP3:	D(LABE)
IDLP1:	D(B), D(A), D(C), D(I)
Standard:	D(I1), D(I2), ..., D(In)

It is seen that if the body of the procedure B contained the statement

go to *LABE*;

the identifier *LABE* would be found to be undeclared in *B* and in all surrounding NL's.

The example shows also how the principle (i) of § 1.7 is implemented. An artificial NL for which NLN = 0 also exists and surrounds all other NL's. For a fixed system of standard procedures the standard part of the IDL is also fixed. A change of the system never refers to the logic of the compiler but only to some constants and to the bodies of standard procedures. Obviously, each programme is automatically embedded in the artificial NL during translation.

With such an approach it is observed that there are no reserved identifiers in ODRA-ALGOL. If a standard identifier is declared at a certain NL then the standard meaning of that identifier becomes unavailable at that NL by the block mechanism of ALGOL. As there is a serious reason for a very efficient manipulation of standard procedures at run-time, the information necessary for a special treatment of these procedures is also kept in the standard part of the IDL.

When searching for a D(I), where I is an identifier which occurs within a certain NL, some special situations should be considered separately. These involve

(1) the analysis of declarations in pass 2, and, in particular,

(2) the analysis of bound pair lists.

To illustrate (2), we shall consider an example; dots represent the irrelevant parts.

```
A:   begin integer n;
        n: = 10;
     . . . . . .
B:   begin array a[1:n, −n:n]; integer n;
     . . . . . . . .
     end;
     . . . . . .
     end
```

Such a programme is wrong by [5]. By the rule (b) of 4.1.3 of [5], the identifier *n* declared in block *A* is unavailable in block *B* and thus, by 5.2.4.2, the bound pairs of the array *a*, and the array itself, are undefined. In our opinion such an error should be detected by the compiler. It is an easy task when the IDL and the NLL are formed.

2.4. On automatic block structure analysis.

2.4.1. The RFL2 for pass 1 and the general algorithm for syntax--checking of a RFL2 text. To generate the block structure description, as defined in § 2.3, the block structure has to be analysed. To control this process, we shall use a RTT generated for a RFL2 (see § 1.5) which describes the block structure of an ALGOL programme. The grammar of this RFL2 is given on pages 43, 44. The reader is asked to study it before going on. It has been checked by the table generating programme that the RFL2 is unique with respect to the syntactic element called programme.

The reason for which some complex ALGOL constituents, such as for clause (FORC), unlabelled basic statement (UBS), if clause (IFC), non-procedure declaration (NPD) are basic symbols of the RFL2 is to keep the size of the RTT possibly low; see also § 2.4.2.

Now we proceed to describe the use of the Compactified RTT. The actual CRTT's are given on pages 45-48 (pass 1) and 54-83 (pass 2). A CRTT consists of three tables called Stack Table (ST), Intermediate Table (IT) and Final Table (FT) respectively. If there is no syntactic element in a certain column of ST or FT then this empty place denotes the EMPTY symbol which should be interpreted just as Ø in [1].

In the ODRA-ALGOL compiler each syntactic unit has an internal eight-bit representation and thus each row of each table can be represented as a single machine word. Thus the CRTT for pass 1 requires $173+32+$ $+102 = 307$ machine locations while the original RTT has 557 rows which require either 1114 locations with an eight-bit representation or 557 with a five-bit representation. The latter is not possible for pass 2.

To perform syntax-checking of a RFL2 text, i.e. a string of basic elements of the RFL2, by means of the CRTT generated for that RFL2, a single Stack S is used. There is also a routine called READ whose functions consist in moving the input tape appropriately so as to define the Last basic Symbol (LS). The whole algorithm for syntax-checking is given below in an ALGOL-like notation.

```
P: = 1; **comment**: P is the stack pointer for S;
S[0]: = EMPTY;
READ(LS);
S[1]: = LS; **comment**: LS is defined by the READ routine;
```

```
E3: READ(LS);
E2: TOS1: = S[P−1];
E1: TOS: = S[P]; **comment**: TOS is the TOp of Stack;
    **if** TOS1 = EMPTY ∧ TOS = DISTINGUISHED ELEMENT ∧
      ∧ LS = EMPTY
      **then go to** TEXT IS CORRECT;
    **if** the pair TOS1 TOS is found in the ST
      **then** determine the associated Intermediate Table Pointer ITP
      **else go to** SYNTAX ERROR TREATMENT;
    determine Lower Bound LB and Upper Bound UB from IT[ITP];
    **if** LS is found between lines No. LB to UB of the Final Table FT
      **then** determine the associated STATE and REDUCTION
      **else go to** SYNTAX ERROR TREATMENT;
    **go to if** STATE = 1
             **then** S1
             **else if** STATE = 2
                   **then** S2
                   **else** S3;
S1: S[P]: = REDUCTION; **go to** E1;
S2: P: = P−1; S[P]: = REDUCTION; **go to** E2;
S3: P: = P+1; S[P]: = LS; **go to** E3;
```

The above general algorithm is the basis of the main loops of pass 1 and pass 2 of the compiler. The actual loops differ in their READ routines and in the actual CRTT's. The actual loops contain also some testing of REDUCTION, e.g.

```
**if** REDUCTION = UNLABELLED BLOCK
  **then go to** BLOCK CLOSE ROUTINE;
```

Further details of the main loops are given in §§ 2.4.4, 3.3, 3.4, 3.5, 3.7 of the present paper.

The difference between our CRTT and RTT of [1] consists in storing some repeated subtables once only and in an indirect reference to these subtables via a pointer to a short intermediate table. It may be noted that some of the subtables of the FT contain identical lines. Thus a further compactification is possible which would reduce the size of the tables to about 0.9 of their present size. The resulting save of storage space is too small and the troubles too serious for the job to be done.

2.4.2. The READ routine for pass 1. The READ routine has to extract one LS (see § 2.4.1) from the input. This is trivial in pass 2 since the LS are very simply derived from the syllable representations (see § 2.2). In pass 1 the READ routine is not so simple since among the basic symbols of the RFL2 for pass 1 (see page 44) we have also the following ALGOL elements: procedure heading, non-procedure declaration, for clause, if clause, unlabelled basic statement.

The strategy is to skip and store these elements, one by one, without too much analysing their internal structure. This work is performed by some parts of the READ routine called skip routines. The main functions of skip routines are listed below.

(a) Filling up the Identifier Description Stack (IDS). The initial contents of this stack are the descriptions of standard identifiers. Each time a declarative part of an ALGOL programme is being read, the relevant identifier descriptions are formed. These are pushed into the IDS together with the current NLN. The descriptions are not complete, e.g. they do not contain such details as the addresses assigned to labels, switches and procedures, the lengths of switch lists and so on. These will be completed in later passes when the IDL is already formed.

(b) Creating a new NL at suitable points of the input text, i.e. stacking the NLN and increasing NLN by one.

(c) Discarding the declarative parts of the text no more useful when the relevant description is already in the IDS. The following items are actually discarded: type declarations, formal parameter lists, value parts, specification parts (see also the example in § 2.4.3).

(d) Delivering the delimiter **beginb** instead of a block **begin**.

(e) Inserting an explicit dummy statement symbol where necessary. The text

if B **then else go to** E;
begin end;
E: **end**;

of 14 syllables would be stored on the output from pass 1 as 17 syllables:

if B **then** $DUMMY$ **else go to** E;
begin $DUMMY$ **end**;
E: $DUMMY$ **end**;

The purpose of inserting a special symbol for dummy statement is to make possible for pass 2 to deal with dummy statements by the general simple mechanism. The grammar of the RFL2 for pass 2 fails to accept the ALGOL construction

begin
  ......
  if B **then go to** E;
  ......
E: **end**

while the construction

```
                begin
                  ......
                  if B then go to E;
                  ......
                E: DUMMY end
```

is perfectly valid in this grammar.

Now we shall briefly describe how the basic elements of the RFL2 for Pass 1 are extracted from the input text.

Type declaration skip. There are no difficulties as the type declarations cannot contain further type declarations and a simple loop can be used to check the validity of the declarations and to stack the necessary identifier descriptions.

Procedure heading skip. This is a bit more complicated since the specifications should be checked against the formal parameter list and the latter should be also compared with the value part. A new NL should be created. An obvious routine with no parameter stacking, however, can be used to deal with the task.

Unlabelled basic statement skip and other skip routines. These routines must be more complicated, as shown by the following example:

```
    if B
      then x: = if B1 then E1 else if B2 then E2 else E3
      else go to L;
```

which should be recognised in pass 1 as the following formal text

IFC UBS ELSE UBS;

To cope with the problem, we use some counting variables. Let $C_{p,q}$ denote a variable whose value is controlled by the following rules:

at the beginning of the skip process: $C_{p,q}: = 0$;
after reading a delimiter equal to p: $C_{p,q}: = C_{p,q}+1$;
after reading a delimiter equal to q: $C_{p,q}: = C_{p,q}-1$;

Now, if the UBS is correct then just before reading a basic statement terminating symbol, the expression

$$C_{\mathbf{if},\mathbf{then}} = 0 \wedge C_{\mathbf{if},\mathbf{else}} = 0$$

is assigned the value **true**. Unfortunately, during the skip process it is not known whether the **if**-**then**-**else** structure of the UBS is correct. In order to cope with this and to detect as soon as possible some delimiter structure errors, a boolean matrix $||b_{ij}||$ is used; if the i-th delimiter can occur within the j-th syntactic unit to be skipped in pass 1, then $b_{ij} = 1$, and otherwise $b_{ij} = 0$. The value of j is set appropriately at the start of each skip routine. Thus it is easy, for example, to detect a wrong delimiter within a for clause or within an array declaration. When skipping a UBS, there is an extra check of the counting variables $C_{\mathbf{if},\mathbf{then}}$ and $C_{\mathbf{if},\mathbf{else}}$. On reading a semicolon or an **end** these must be zero; if they are zero and an **else** is read, then this **else** is assumed to terminate the UBS. There may be, of course, an **if** or **then** missing in the UBS whereupon this method fails to detect the terminating **else**; if this is the case then an improper delimiter will be detected in the next part of the programme. In the example given above this would be **go to** found within the first UBS, provided that the second **then** is missing (or it is not underlined).

The above outlined method is also used to skip the following ALGOL constituents: an if clause (within a statement but not within an expression), a for clause, an array declaration (in particular a bound pair list), a switch declaration.

Such actions seems to be duplications of the effort of pass 2. This is the case but not in all respects. If the additional work was deleted, the block structure would be analysed under the assumption that the delimiter structure is correct. It is known, however, that quite a good amount of programmers' mistakes pertain to delimiter structure. As a result, it would be possible for pass 2 to "detect" some strange errors caused merely by the wrong information in the IDL due to delimiter structure errors. The additional work takes microseconds which should be compared with the milliseconds necessary to extract each syllable from the bulky input tape. Further, we do not want pass 2 to work if even a single error has been revealed in pass 1.

This approach cuts down the time taken to discard incorrect programmes.

2.4.3. An example. To illustrate how the input text should be manipulated in pass 1, we shall give an example which is intended to contain the most powerful concepts of ALGOL. The programme

```
begin
  real procedure sigma(i, j, k, term);
    value j, k; real term; integer i, j, k;
    begin
      real partial;
      partial: = 0;
      for i: = j step 1 until k do partial: = partial+term;
      sigma: = partial
    end of sigma declaration;
    integer i, j; real alpha, eps;
    read (alpha, eps);
    print (alpha, eps, sigma (i, 2, 10, sigma (j, 1, 3, alpha×j/(eps+i))))
  end
```

reads two numbers a and ε and prints in a presumed format the three numbers

$$a,\ \varepsilon,\ \sum_{i=2}^{10}\sum_{j=1}^{3} a_j/(\varepsilon+i).$$

The associated formal text for pass 1 and the corresponding reducing stack history are given on page 49; in the stack history the following conventions are used: NS is the number of the formal basic symbol which is currently in LS (see §§ 2.4.2, 2.4.1), P is the current stack pointer, L is a count increased by one after each look-up in the CRTT is performed, S(P−1) and S(P) are two topmost elements of the reducing stack, ST and RED are the STATE and REDUCTION respectively defined just after looking-up in the CRTT, as indicated by the general checking algorithm.

The reader will be able to check, at least partially, the stack history by following the algorithm given in § 2.4.1 and looking up in the CRTT on pages 45-48.

After pass 1 is completed, the following sequence of 72 syllables in the form of associated syllable representations (see § 2.2), will be found in the core store:

```
  beginb
    procedure sigma;
      beginb
        partial: = 0;
        for i: = j step 1 until k do partial: = partial+term;
        sigma: = partial
      end;
    read (alpha, eps);
    print (alpha, eps, sigma (i, 2, 10, sigma(j, 1, 3, alpha j/(eps+i))))
  end EMPTY
```

and some lists will be generated (see § 2.3), namely

IL:	I1, I2, ..., I*n*, *sigma*, *i*, *j*, *k*, *term*, *partial*, *alpha*, *eps*
CL:	*0*, *1*, *2*, *10*, *3*
SL:	*empty*
IDL:	IDLP3: D(*partial*)
	IDLP2: D(*i*), D(*j*), D(*k*), D(*term*)

IDLP1: D(*sigma*), D(*i*), D(*j*), D(*alpha*), D(*eps*)
Standard: D(I1), D(I2), ..., D(I*n*)

NLL:

NLN	SNLN	IDLP
0	0	Standard
1	0	IDLP1
2	1	IDLP2
3	2	IDLP3
4	3	0

An almost complete algorithm for generating the above lists is given in the next section. A further history of this example is given in § 3.8.

2.4.4. Generating the block structure description. Although the fully bracketed structure of ALGOL 60 makes it possible to use a single stack for analysing the input text, it is more lucid to split the stack into several ones with more specialized functions. The translation-time storage administration of the ODRA-ALGOL compiler is powerful enough to make efficient use of the available storage space even when dealing with multiple lists of varying sizes.

To generate the block structure description, pass 1 uses three stacks, namely

(1) the reducing stack S, mentioned already in § 2.4.1 and 2.4.3,

(2) the Identifier Description Stack IDS, mentioned in § 2.4.2,

(3) the Parameter Stack PS which is used to hold the values of various contextual parameters, such as nomenclature level numbers, block levels, parameters for static storage allocation and so on. In this section we shall state precisely only the block-structure functions of the PS.

Let Q be the stack pointer for the PS, and let NLN, NLL, IDL, IDLP, S, P and LS have exactly the same meaning as before (see § 2.3, 2.4 or the Index of Abbreviations at the end of this paper). Let CNLN denote the Current Nomenclature Level Number, i.e. the NLN for the currently innermost NL. We shall describe in an ALGOL-like notation the actions necessary to generate the block structure description.

2.4.4.1 Initialization of the process. The following is done at entry to pass 1:

```
Set initial list positions;
Set initial IDLP;
Fill in the IDS with the descriptions of standard identifiers;
**comment**: this is done to deal properly with a programme which is a la-
  belled block or a compound statement which consists of labelled sta-
  tements. The labels belong then to the artificial NL which surrounds
  all other NL's;
CNLN: = NLN: = 0; Q: = −1; P: = 1; S[0]: = EMPTY; READ(LS);
  S[1]: = LS;
```

2.4.4.2. Further details on the READ routine for pass 1. The READ routine has been discussed in § 2.4.2. The following is also done in this routine:

At the beginning of a new NL:

```
Q: = Q+1;
PS[Q]: = CNLN;
CNLN: = NLN: = NLN+1;
```

On reading an information which describes the nature of an identifier:

```
Form the identifier description;
Push the description into the IDS together with CNLN
```

2.4.4.3. The main loop of pass 1. The general algoritm, given in § 2.4.1, is rewritten here in a form suited to generating the block structure description:

```
E3: READ(LS);
E2: TOS1: = S[P−1];
E1: TOS: = S[P];
    Test for end of the process;
    Look-up in the CRTT and define STATE and REDUCTION;
    comment: if STATE and REDUCTION are not definable, a block
      structure error is indicated;
      go to if STATE = 1
            then S1
            else if STATE = 2
                   then S2
                   else S3;
S1: S[P]: = REDUCTION; go to E1;
S2: if REDUCTION = UNLABELLED BLOCK ∨
      REDUCTION = PROCEDURE DECLARATION ∨
      REDUCTION = FOR STATEMENT ∧ TOS1 = FOR CLAUSE
    then begin
           NLL[CNLN]: = pack parameters(PS[Q], IDLP);
           comment: at this moment the top of IDS contains
             nothing except the descriptions of all identifiers
             belonging to the current NL;
           Add the top of IDS to IDL and update IDS and IDLP;
```

```
                CNLN: = PS[Q];
                Q: = Q−1;
            end;
     P: = P−1; S[P]: = REDUCTION; go to E2;
S3: P: = P+1; S[P]: = LS; go to E3;
```

3. Organization of pass 2.

3.1. Main objectives. Pass 2 has to perform the following tasks:

(1) the complete syntax check,

(2) a proper syllable replacement and insertion so as to facilitate pass 3,

(3) generating some further information on the identifiers, e.g. the length of the switch list in a switch declaration,

(4) a semantic check of the text, as complete as possible and practicable at translation time.

The actions mentioned in (2) are performed mainly in order to enable pass 3 to distinguish between some delimiters with multiple meaning (e.g. a comma) and to find all necessary information on the identifiers without any table look-up; for this purpose all the IN's for identifiers are replaced by the value of a linear function of the associated IDLP. The other task of pass 2 is a transformation of standard procedure statements and function designators so that pass 3 will be able to code these optimally with no additional effort (see [10]).

It will be shown in the next sections that the tasks mentioned under (2)-(4) can be accomplished almost as a by-product of (1) by interrupting the latter at appropriate stages.

3.2. The RFL2 for pass 2. To check the input text for pass 2 by the general algorithm given in § 2.4.1, we have first to find a reducing and unambiguous grammar for the most essential part of ALGOL 60 (note that the input text for pass 2 differs from that contained on the original input tape; see § 2.4.2 (c) and § 2.4.3).

The very troublesome syntactic unit is ⟨arithmetic expression⟩, since it occurs in many places of the grammar of ALGOL (see § 1.3). To cope with the problem, the formula for ⟨expression⟩ is deleted and some other syntactic units are re-defined, e.g.

```
⟨actual parameter⟩:: = ⟨arithmetic expression⟩|
                       ⟨boolean expression⟩|
                       ⟨designational expression⟩|
```

⟨subscript list⟩:: = [⟨arithmetic expression⟩ |
    ⟨subscript list⟩, ⟨arithmetic expression⟩
⟨subscripted variable⟩:: = ⟨array indentifier⟩ ⟨subscript list⟩]
⟨switch list⟩:: = : = ⟨designational expression⟩ |
    ⟨switch list⟩, ⟨designational expression⟩
⟨switch declaration⟩:: = **switch** ⟨switch identifier⟩⟨switch list⟩

To avoid the use of the formula

⟨for list element⟩:: = ⟨arithmetic expression⟩

the definition of ⟨for clause⟩ should also be changed. This is done after the manner described above for ⟨switch list⟩.

Note that the above changes in the grammar of ALGOL do not affect the resulting formal language. This is not the case with some other parts of ALGOL grammar exemplified by the following formulae:

⟨actual parameter⟩:: = ⟨procedure identifier⟩ | . . . . . .
⟨procedure statement⟩:: = ⟨procedure identifier⟩⟨actual parameter part⟩
⟨actual parameter part⟩:: = ⟨empty⟩ | (⟨actual parameter list⟩)

At this point two troubles arise, the first related to reducibility and the second to ambiguity of the language. In pass 1 there are no means to introduce an explicit symbol standing for ⟨empty actual parameter part⟩ in a similar manner as for ⟨dummy statement⟩. For this reason we have to accept the following definition:

⟨procedure statement⟩:: = ⟨procedure identifier⟩ |
    ⟨procedure identifier⟩⟨actual parameter part⟩

where ⟨actual parameter part⟩ is not allowed to be ⟨empty⟩. However, the resulting grammar is not a reducing one, since there is still the formula ⟨actual parameter⟩:: = ⟨procedure identifier⟩, and it could be observed that there is no remedy to cope with it as before unless the resulting formal language is allowed to be ambiguous. To see this we shall consider an example.

begin real procedure *Z*; *Z*: = *1*;
  procedure *X*(*Y*); **procedure** *Y*;
    begin .... *Y*(*Z*); .... **end** *X*;

The specification part of *Y* does not exist and the specification parts of the corresponding actual parameters cannot be taken into account at translation time (even if they were available) as they may be quite different. Such a situation, although somewhat strange when considered statically (at translation time) may be perfectly valid from the dynamic point of view (at run time). Generally, at translation time it is not known whether Z in $Y(Z)$ represents an arithmetic expression or it is an actual procedure identifier parameter. There are three possible remedies for this trouble:

(1) Replace the appropriate part of the ALGOL grammar by the following:

$$\langle\text{procedure statement}\rangle ::= \langle\text{procedure identifier}\rangle\,()\,|\\ \langle\text{procedure identifier}\rangle\,(\langle\text{actual parameter list}\rangle)$$

which implies that in the forementioned example we could write either $Y(Z())$ or $Y(Z)$, according to the intended meaning of Z. The resulting language, however, is not contained within ALGOL.

(2) Impose on the implemented language the restriction that a formal parameter must not be specified to be a procedure. If all formal parameters have specifications then with such a restriction the compatibility of formal and actual parameters can be completely checked at translation time and thus procedures may be manipulated very efficiently at run time of the object programme.

(3) Impose the restriction of (2) on the RFL2 for pass 2, but not on the implemented language. This means that in the above example the identifier Z in $Y(Z)$ will always be recognised as an arithmetic expression at translation time and the necessary distinction will be made at run time of the programme. With this approach the compatibility of formal and actual parameters must be, in general, checked at run time and the machine coding for a procedure call is far less efficient than previously in (2), especially with respect to size.

In the first version of the ODRA-ALGOL compiler, which is intended to be used on the smallest machine configuration, we have adopted (2). In the next version, designed for the ODRA 1204 with a backing storage, the solution mentioned in (3) will be used.

To obtain a RFL2 one is also forced to impose another restriction, rather a minor one, which states that a function designator must not be used to form a proper procedure statement and, conversely, a proper procedure statement must not be used as a function designator. In this view the statement

$$\textbf{begin}\ y := sin(x)\ \textbf{end}$$

is perfectly valid, while the ALGOL statement

$$\textbf{begin}\ sin(x)\ \textbf{end}$$

is wrong in ODRA-ALGOL and, we hope, rightly so.

The complete grammar of the formal language for pass 2 is given on pages 50-53. It has been verified by the programme already mentioned that this grammar defines a RFL2 which is unique (i.e. unambiguous) with respect to the syntactic unit called programme (see [1], [8] for definition).

The associated RTT, constructed after the manner of [1], consists of 6366 rows (34 binary digits for each row with an eight-bit representation for each syntactic element) which would require 12732 24-bit locations of core store. The CRTT derived from that RTT requires 2794 24-bit locations of core store (1234+249+1311). The tables are given on pages 54-83.

It should be noted that the grammar for pass 2 distingushes between such elements as

⟨arithmetic variable⟩
⟨boolean variable⟩
⟨arithmetic array identifier⟩
⟨boolean array identifier⟩

and so on, while in [5] there are only ⟨variable⟩, ⟨array ident.⟩ etc. with the effect that in ALGOL 60 a text of the form

begin **real** x; **Boolean** $y; x := x+y; \ldots$

is forbidden by the informal semantics and not by the formal syntax. This is not the case in ODRA-ALGOL where features of this kind are taken into account in the formal grammar, thus giveng a very natural means for checking operand compatibility with no special effort.

3.3. The READ routine for pass 2. During pass 2 and pass 3 the main part of working locations contains the Input Text (IT) at the bottom and the Output Text (OT) at the top. In a single step of pass 2 IT decreases and OT increases, while the sum of the two texts is, in general, slowly growing up. This is the basis for storage administration in the READ routine which is entered at the beginning of the main loop of pass 2. The loop is derived from the general algorithm given in § 2.4.1. Assuming a proper initialization of pass 2, the actions of the READ routine are performed in the following simple steps:

(1) Store the Buffer SYLlable (BSYL) at the bottom of OT; OT is increased by one syllable.

(2) Take a syllable from the top of IT giving a new BSYL; IT is decreased by one syllable.

(3) Define LS i.e. the formal Last basic Symbol, corresponding to BSYL.

After this is completed, the next step of syntax-checking is performed, i.e. STATE and REDUCTION are defined from the associated CRTT, and further actions depend on the values of these variables. The syllable just read from the input text is kept in BSYL until the READ routine is entered again. This is convenient to perform syllable replacement and insertion, as shown in § 3.4.

Further details on step (3) of the READ routine are given in the table below:

CN	The information on BSYL	The value of LS
0	unary minus	unary arithmetic operator
	$+ \; - \; \times \; / \; \div \; \uparrow$	binary arithmetic operator
	$\neg$	unary boolean operator
	$\vee \; \wedge \; \supset \; \equiv$	binary boolean operator
	$= \; \neq \; < \; \leqslant \; > \; \geqslant$	relational operator
1	declaration **real**	SAV
	declaration **integer**	SAV
	declaration **boolean**	SBV
	declaration **real array**	AAI
	declaration **integer array**	AAI
	declaration **boolean array**	BAI
	declaration **real procedure**	AFI
	declaration **integer procedure**	AFI
	declaration **boolean procedure**	BFI
	declaration **procedure**	PI
	declaration **label**	L
	declaration **switch**	SWI
	specification **real**	SAV
	specification **integer**	SAV
	specification **boolean**	SBV
	specification **real array**	AAI
	specification **integer array**	AAI
	specification **boolean array**	BAI
	specification **label**	L
	specification **switch**	SWI
	specification **string**	STRING
2	IN $= 0$ or IN $= 1$	BC
2	IN $\neq 0$ and IN $\neq 1$	AC
3	no matter	STRING

To define the LS for an identifier I whose identification number is IN, a look-up in the IDL has to be performed. A by-product of this is the relative position of D(I) in the IDL. The IN is replaced by the value of a linear function of this relative position and thus pass 3 is able to find the description of any identifier with no table look-up, by means of simple arithmetic operations performed on the identification number.

3.4. On text-processing. After STATE and REDUCTION are defined in the main loop of pass 2, i.e. no syntax error has so far been revealed, further actions depend on the values of these variables in a manner similar to that described in §§ 2.4.1 and 2.4.4.3. In this section we shall briefly describe some actions related to text-processing in pass 2. An ALGOL--like notation is used again. The array OT denotes Output Text and OTP is the Output Text Pointer, i.e. OT[OTP] is the last syllable stored on the output side.

```
if STATE = 2 then
begin
if REDUCTION = UNLABELLED BLOCK
  then OT[OTP]: = END OF BLOCK; comment: END OF BLOCK
    is a delimiter which replaces the original end of a block;
if REDUCTION = FOR1
  then OT[OTP]: = FOR ASSIGN; comment: FOR ASSIGN is a deli-
    miter which replaces: = in a for clause;
if REDUCTION = FOR2 then
  begin
  if LS = COMMA
    then BSYL: = END OF FOR LIST ELEMENT
    else if LS = DO then
           begin OTP: = OTP+1; OT[OTP]: = END OF FOR LIST
             ELEMENT end
  end REDUCTION = FOR2, END OF FOR LIST ELEMENT is
    a delimiter which replaces the original comma in a for clause. It is
    also inserted before the delimiter do;
if REDUCTION = ARITHMETIC EXPRESSION FOLLOWED BY
    A COLON
  then OT[OTP]: = COMMA; comment: this is done to enable pass 3 to
    deal with the bound pair list just as if it were a subscript list;
end STATE = 2;
```

Further actions of this kind are performed to transform standard procedure statements and function designators so as to provide means for an efficient manipulation with them in the object programme. The strategy is to replace in pass 2 each standard procedure identifier by a delimiter which will be considered just as an algebraic operator in pass 3. For example, if the input text for pass 2 contains the statements

$$read(a, b); \quad print\big(b, sin(a+b)\big);$$

where a, b are simple real variables, then the output from pass 2 will contain

```
        **begin** **readreal** (a}; **readreal** (b} **end**;
        **begin** **printreal** (a); **printreal** (**sinoperator** (a+b)) **end**;
```

Pass 3 will transform this piece of text into the following 10 machine instructions:

```
                    Take address of a;
                    Call real number read subroutine;
                    Take address of b;
                    call real number read subroutine;
                    Take a;
                    Call print subroutine;
                    Take a;
                    Add b;
                    Call sine subroutine
                    Call print subroutine;
```

It is seen that the delimiter } in the above context causes the instruction "take operand address" to be generated instead of "take operand". Further remarks on the treatment of standard procedures can be found in [10].

3.5. Calculating the length of a switch list. If the lengths of all switch lists in switch declarations are known then pass 3 is able to code switch declarations easily and efficiently using the usual mechanism for expressions.

To calculate the length of a switch list, pass 2 uses a counting variable CV. The value of it is controlled in the main loop of pass 2 in the manner described below.

```
if STATE = 3 ∧ LS = SWITCH then CV: = 0;
if STATE = 2 then
begin
if REDUCTION = SWITCH LIST then CV: = CV+1;
if REDUCTION = SWITCH DECLARATION then insert the value
   of CV into the appropriate identifier description;
end;
```

The value of CV is never stacked in the PS as the switch declaration cannot contain further switch declarations.

3.6. Checking the length of a subscript list. In pass 2 it is tempting to test the length of each subscript list against the length of the relevant bound pair list. It is better, however, to defer such a check until run time of the object programme, by the following argument:

(1) an array identifier may occur before its declaration is read (i.e. as a non-local object within a procedure declaration). Thus some special mechanism for checking is required.

(2) This special mechanism should be occasionally ignored as the subscript lists occurring with a certain array identifier may be incompatible. If A is a formal parameter specified to be an array identifier then all the variables

$$A[i],\ A[i,j],\ A[i,j,k],\ \ldots$$

occurring in the relevant procedure body should be accepted at translation time. This is obvious by the following example:

begin
procedure $P(A,n)$; **array** A; **integer** n;
begin **integer** i, j, k, l;
. . . .
$i :=$ **if** $n = 1$ **then** $A[i]$ **else** **if** $n = 2$ **then** $A[i,j]$ **else** $A[i,j,k]$ ......
end P;
array $a[1{:}10], b[1{:}10, 1{:}10], c[1{:}10, 1{:}10, 1{:}10]$;
. . . .
$P(a, 1); P(b, 2); P(c, 3)$;
. . . .
end

This example indicates that compatibility of a subscript list with the corresponding bound pair list should be checked at run time. Then it can be generally accepted that the length of a subscript list is not calculated in pass 2. Consequently, the length of a bound pair list is not calculated in pass 2, as it is not needed.

3.7. Checking an actual parameter list. In the first version of ODRA-ALGOL a formal parameter must not be specified to be a procedure (see § 3.2). Thus a complete semantic check of each actual parameter list is performable at translation time. To check an actual parameter list, two variables are used: PIN (Procedure Identification Number) and APN (Actual Parameter Number). PIN has to point to the description of the procedure identifier (or type procedure identifier) which occurs at the beginning of the innermost procedure statement (or the function designator) which is being analysed. APN is the position of the actual parameter being analysed on the appropriate actual parameter list.

Example. Given a procedure statement

$$P(sin(y)+x,\ F(a+b,\ a-b),\ \textbf{if}\ B\ \textbf{then}\ 1\ \textbf{else}\ 10);$$

Just after the analysis of $a-b$ is completed, PIN = address of D(F) (see § 2.3) and APN = 2 since $a-b$ is the second parameter of the function designator being analysed.

It follows from the above example that both PIN and APN have to be stacked at appropriate points of the text due to the inherent recursivity of the actual parameter list. The Parameter Stack (PS) is used for this purpose.

We shall describe the checking mechanism in some detail. The description should be considered as a part of the main loop of pass 2.

```
if STATE = 3 then
begin
if TOS = PI and an opening parenthesis ∨
  TOS = AFI and an opening parenthesis ∨
  TOS = BFI and an opening parenthesis then
begin
PS[Q+1]: = APN; PS[Q+2]: = PIN; Q: = Q+2
APN: = 0;
determine new PIN
end
end STATE = 3;
if STATE = 1 ∧ (REDUCTION = ACTUAL PARAMETER LIST ∨
                REDUCTION = ACTUAL PARAMETER) then
begin
APN: = APN+1;
check TOS against the APNth specification associated with the current
  PIN
end;
if STATE = 2 ∧ REDUCTION = ACTUAL PARAMETER LIST and
  closing parenthesis then
begin
check APN against the number of formal parameters associated with PIN;
Q: = Q-2;
APN: = PS[Q+1];
PIN: = PS[Q+2];
end;
```

Parameterless procedures are considered separately.

3.8. A further history of the example of § 2.4.3. On page 84 the formal text associated with the output from pass 1 (see § 2.4.3) is given. It is worth noting that for each input syllable there is exactly one formal basic element in the formal text, i.e. there is no skip in pass 2 and each input syllable together with its context are tested carefully against the formal grammar for pass 2. Before each step of syntax checking is performed, each input identifier must be found in the IDL to see whether it is declared at the appropriate place of the original ALGOL text. Thus the checking process is complete in a strict sense of this word. The output text from pass 2 differs considerably from the associated input text. It is known that this output presents a syntax-error-free text. As indicated in the previous sections, a semantic check has also been performed.

The formal text on page 84 is followed by its reducing stack history. The reader will be able to verify it by following the general algorithm given in § 2.4.1.

To illustrate what has been done with the text during pass 2, the output from pass 2 (associated with the example of § 2.4.3) is given below in a legible form.

```
beginb
  procedure sigma;
    beginb
      for i for-assign j step 1 until k end-of-for-list-element do
        partial: = partial+term;
      sigma: = partial
    end-of-block;
  begin readreal (alpha}; readreal (eps} end;
  begin printreal (alpha); printreal (eps);
        printreal (sigma(i, 2, 10, sigma(j, 1, 3, alpha×j/(eps+i)))) end
end-of-block EMPTY
```

Without any further checking pass 3 translates this text directly into machine code during a single scan process, with no table look-up, no back-up and no look-ahead. A single stack is used for this purpose.

4. Closing remarks.

The method of syntax-checking already described is applicable to any machine with a sufficient high-speed storage. Although the CRTT's for the two passes of the ODRA-ALGOL compiler occupy 3101 24-bit locations, the remarkable simplicity and clarity of the logic of various routines controlled by means of the information found in the CRTT made it possible to keep the size of the compiler within reasonable bounds (see § 4.2). It is also worth noting that the most complicated parts of the logic of the compiler, i.e. the CRTT's, have been completely generated on a computer by means of some programmes which require a very small amount of hand-prepared data. As a result, the problem of checking and debugging the compiler is reduced considerably.

4.1. A time estimation. It would be probably interesting to give some estimation of the time taken to look-up in the CRTT in pass 2 when analysing a real programme (the corresponding time for pass 1 is negligible when compared with the time taken to deal with the bulky input tape). The time depends, obviously, on the method of searching.

All the tables given in this paper are sorted in some arithmetic order and thus dichotomic searches may be employed. This will give good results on every machine.

On a machine with built-in look-up facilities, however, some other method may give still better results. It is the case with the ODRA 1204 computer.

The basis of the method is a division of the stack table ST into a sequence of subtables SBT1, SBT2, ..., SBTn where each subtable has identical elements in TOS1 column. The number of subtables is never greater than than the number of syntactic elements of the language. If ST is considered as an one-dimensional array and ST$[m{:}n]$, where $m \leqslant n$, denotes the set

$$\{\mathrm{ST}[m], \mathrm{ST}[m+1], \ldots, \mathrm{ST}[n]\}$$

then we have

$$\begin{aligned}\mathrm{STB1} &= \mathrm{ST}[1{:}10]\\ \mathrm{STB2} &= \mathrm{ST}[11{:}12]\\ \mathrm{STB3} &= \mathrm{ST}[13{:}13]\\ \mathrm{STB4} &= \mathrm{ST}[14{:}31]\end{aligned}$$

and so on (see pages 54-66).

Let {E0, E1, E2, ..., E143} be the set of syntactic elements of the formal language for pass 2 (see page 53). A further table BTST (Bound Table for Stack Table) is used such that

$$\mathrm{BTST}[i] = \begin{cases} 0, \text{ if E}i \text{ cannot occur as TOS1 in ST} \\ \textit{packed}(m, n), \text{ if ST}[m{:}n] \text{ has E}i \text{ in TOS1 column}\end{cases}$$

Suppose that TOS1 = Ej while a look-up in the CRTT has to be performed. Then the following has to be done:

```
if BTST[j] = 0
then go to SYNTAX ERROR TREATMENT
else unpack m and n;
if the pair TOS1 TOS is found in ST[m:n]
then proceed just as described in § 2.4.1;
```

It is seen that the most time-consuming part of this work is the search in ST$[m{:}n]$ and then in the subtable of FT. The maximum length of subtables of ST is 76 and the maximum length of subtables of FT is 19. Thus T, the time taken to define STATE and REDUCTION, is estimated as follows

$$T \leqslant t_1 + (76+19)t_2 = t_1 + 95t_2$$

where t_1 is a short constant time necessary for parameter planting and starting the search loop, and t_2 is the time taken to perform this loop once. If the search loop is performed by the special hardware of the ODRA 1204 computer then $t_2 = 32$ microseconds, i.e. the computer is able to perform at least 300 searches per second. It should be noted, however, that the mean search time for an ALGOL programme taken at random will be several times less than $t_1 + 95t_2$ due to the fact that the most often appearing stack situations are found at the top of each subtable (look-up in the ST for the parts of expressions).

The other question of interest is how many searches are needed to perform the analysis of a programme. Let N be the number of syllables in the input text for pass 2. Then it follows from the general algorithm given in § 2.4.1 that STATE = 3 must be obtained exactly $N-1$ times and the same is true for STATE = 2 since N syllables must be paired $N-1$ times to obtain a single syntactic unit called programme. STATE = 1 cannot appear more than $N-1$ times. This follows from the fact that a canonical reduction sequence is obtained and unnecessary situations for which STATE = 1 are automatically avoided. For example, see the row where L = 97 in the stack history on page A44 where we have

S(P−1)	S(P)	LS	ST	RED
PI(	SAV	,	1	ACPL

The chain

ACPL:: = ACP:: = AE:: = SAE:: = APR:: = AV:: = SAV

has been completely ignored except of its leftmost and rightmost elements.

Thus the number of searches necessary to perform the analysis of a text of N syllables cannot exceed $3(N-1)$ and the time spent on searching in the CRTT cannot exceed

$$3(N-1)(t_1 + 95t_2)$$

where t_1 and t_2 depend on the object machine.

Example. To write a good algorithm for solving simultaneous linear equations (pivotal elimination), less than 300 syllables are required with input and a readable output included. The time spent on searching in CRTT will not exceed 3 seconds on the ODRA 1204 computer. It is very likely to obtain exactly this figure when dichotomic searches are used exclusively.

Finally it should be noted that the time taken to perform all other tasks of the compiler is much less than in a comparable compiler written along some other lines, due to the simplicity and linearity of most of the analysing and compiling routines of the ODRA-ALGOL compiler.

4.2. Some data on the first version of the compiler. Experience with the computer shows that the time estimation given in the previous section is rather pessimistic. For example, a real ALGOL programme of 400 syllables (see § 2.2) was checked and translated into machine code in 8 seconds, whilst pass 1 took 5 seconds, pass 2 took about 2 seconds and pass 3 about 1 second. The object programme was about 400 machine words long. It is observed that, on average, pass 1 takes $60^0/_0$ of the total translation time, pass 2 takes about $30^0/_0$ and the rest is used to generate the machine coding by pass 3, whilst the input programme tape is read at about 400 characters per second (this is system-limited).

In the 16K core store of the ODRA 1204 computer there are 13536 locations available for the whole ODRA-ALGOL system. After the compiler is read into the machine, the available store is occupied as shown on the diagram given below.

Block I, a total of 2328 locations, which is never overwritten with ALGOL programmes: Communication with the operating system of the machine, The bodies of standard procedures. Run-time subroutines, e.g. dynamic storage administration. Working variables of the compiler and of the above mentioned subroutines.
Block II, a total of 3784 locations: Working locations of pass 1. The next block is not overwritten by the compiler, or the object programme, if there is even a single free location in the present block.
Block III, a total of 1858 locations: The body of pass 1. This block can be overwritten by the next two passes, when block II is completely occupied.
Block IV, a total of 3765 locations: The body of pass 2. Subroutines used by pass 1 and pass 2. This block can be overwritten by pass 3.
Block V, a total of 1801 locations: The body of pass 3. Translation time storage administration routines used by the three passes. This block can be overwritten only at run time of the object programme.

It follows from the above diagram that pass 3 can use at most 9407 (i.e. 3784+1858+3765) locations for the object programme and the necessary information lists, and thus a maximum of 9000 object instructions can be generated. It follows from experience (and also from § 2.2) that the size of Block II is sufficient to accommodate the information equivalent to about 7000-20 000 object instructions, as far as real programmes are considered. An example of a programme which is not "real" is given at the end of § 2.2; we had successfully tried to translate a programme of this kind with 1000 left parentheses and 1000 right ones, and the compiler was then able to be used again without reinput.

4.3. Remarks on the translation-time storage administration. At the beginning of § 2.4.4.1 there is a "procedure statement"

Set initial list positions;

This means that after pass 1 is entered, the locations of Block II are distributed between 9 lists necessary for pass 1, namely
1. Identifier List IL.
2. String List SL.
3. Constant list CL.
4. Output Text OT.
5. Stack S.
6. Parameter Stack PS.
7. Identifier Description List IDL.
8. Identifier Description Stack IDS.
9. Nomenclature Level List NLL.

The lists are stored in the above order. The distribution of storage space is based on some observations concerning the number of identifiers, constants, nomenclature levels etc. within an average ALGOL algorithm in numerical analysis; the initial room for each list is sufficient to accommodate the necessary information for most programmes. Now, if an unusual programme has to be translated (e.g. with an enormous number of strings, or left parentheses one after another), then there will be no room in a certain area allocated to a list, while other areas still will contain free locations. In such a case the compiler shifts some lists in the store so as to make room for the list in question. This approach shows the following advantages when compared with the method of threaded lists (where each piece of information contains the address of the next part):

a. No space is wasted on the linking information.

b. The built-in look-up facilities of the ODRA 1204 computer can be fully employed, thus speeding-up some parts of the compiler (the facility can be used to look-up in a block of consecutive locations only, or a "block" of equally-distanced locations).

The drawback of the method is that some programmes will be translated slowly due to multiple shifts of thousands locations (but note that there are fast built-in store shift facilities in the ODRA 1204 computer). The probability that the drawback will be observed, however, is made small, at least for a class of computer applications.

Note, that the storage administration can be made more efficient if certain lists are stored in reverse order and next to a certain other list. For instance, in pass 1 IDL is formed normally, and IDS is formed in reverse order. When a nomenclature level has to be closed, the top of IDS must be transferred to IDL, but the amount of necessary storage space does not change. If IDS is stored next to IDL then the transfer of the top of IDS never causes the content of the store to be shifted. For similar reasons in pass 2 and pass 3 the input text is stored next to the output text.

In pass 2 SL and CL become fixed, and IL and IDS become obsolete. The former are packed together in the store, and then behave as a single fixed list, while the latter are deleted. The arrangement of lists in pass 2 is given below:

1. Stack S.
2,3. String List and Constant List.
4. Output Text (an increasing list).
5. Input Text (a decreasing list).
6. Parameter Stack (a varying list).
7. Identifier Description List (a fixed list).
8. Nomenclature Level List (a fixed list).

In pass 3 the arrangement becomes:

1,2,3. The store for simple object variables and cues for arrays, SL and CL packed together.
4. Output Text (the object programme, an increasing list).
5. Input Text (a decreasing list).
6. Stack (a varying list).
7. Identifier Description List (a fixed list).

After pass 3 is completed, the store contains the following lists, packed one after another:

0. The content of Block I (see the previous section).
1,2,3. Just as in pass 3.
4. The object programme
5. The auxiliary variables used for evaluating expressions in the main programme.
6. Free store, used as a run-time stack.

It has been already noted that the compiler may be overwritten at this moment, if the above mentioned lists are large enough. In any case the store contains a complete programme ready to be run just after pass 3 is completed.

Acknowledgement. The author is greatly indebted to Dr. S. Paszkowski, head of the Department of Numerical Methods (Wrocław University), for many valuable discussions on the syntax and semantics of ALGOL 60, and to Dr. W. Turski (Computation Centre of the Polish Academy of Sciences) whose criticism helped to improve a preliminary version of this paper.

Thanks are also due to Mrs. K. Jerzykiewicz, the authoress of pass 3 of the compiler, whose day-by-day and fruitful collaboration made it possible to release the compiler just after releasing the machine.

References

[1] J. Eickel, M. Paul, F. L. Bauer and K. Samelson, *A syntax controlled generator of formal language processors*, Comm. ACM 6 (1963), pp. 451-455.

[2] R. W. Floyd, *Syntactic analysis and operator precedence*, J. ACM 10 (1963), pp. 316-333.

[3] D. Gries, M. Paul and H. R. Wiehle, *Some techniques used in the ALCOR ILLINOIS 7090*, Comm. ACM 8 (1965), pp. 496-500.

[4] E. T. Irons, *The structure and use of the syntax directed compiler*, [in:] *Annual Review in Automatic Programming*, Pergamon Press 1963.

[5] P. Naur et al, *Revised Report on the Algorithmic Language ALGOL 60*, Comp. J. 5 (1963), pp. 349-367.

[6] P. Naur, *The design of the GIER-ALGOL compiler*, BIT 3 (1963), pp. 124-166.

[7] S. Paszkowski, *Język ALGOL 60*, PWN, Warszawa 1965.

[8] M. Paul, *ALGOL 60 processors and a processor generator, in Proceedings of IFIP Congress 62*, North-Holland Publishing Company, Amsterdam 1963.

[9] B. Randell and L. J. Russell, *ALGOL 60 Implementation*, Academic Press, London and New York 1964.

[10] J. Szczepkowicz, *Input and output in ODRA-ALGOL*. To appear.

DEPT. OF NUMERICAL METHODS
UNIVERSITY OF WROCŁAW

Received on 28. 11. 1967,
revised version on 10. 10. 1968

APPENDIX

Index of abbreviations

The index given below contains the abbreviations denoting translation-time parameters of the ODRA-ALGOL compiler and some general concepts of the formal language theory. The abbreviations denoting syntactic units of ODRA-ALGOL are given in the dictionary of internal syntax of ODRA-ALGOL.

ABBR.	MEANING	FIRST USED IN
APN	Actual Parameter Number	3.7
BSYL	Buffer SYLlable	3.3
CL	Constant List	2.2
CN	Category Number	2.2
CRTT	Compactified Reducing Transition Table	2.4.1
D(I)	Description of an identifier I	2.3
IDL	Identifier Description List	2.3
IDLP	Identifier Description List Pointer	2.3
IDS	Identifier Description Stack	2.4.2
IL	Identifier List	2.2
IN	Identification Number	2.2
IT	Input Text	3.3
FL	Formal Language	1.4
LS	Last basic Symbol	2.4.1
NL	Nomenclature Level	2.3
NLL	Nomenclature Level List	2.3
NLN	Nomenclature Level Number	2.3
OT	Output Text	3.3
OTP	Output Text Pointer	3.4
P	reducing stack Pointer	2.4.1
PIN	Procedure Identification Number	3.7
PS	Parameter Stack	2.4.4
Q	parameter stack pointer	2.4.4
RFL	Reducing Formal Language	1.1
RFLn	a RFL with a maximum length of syntactic formulae equal to n	1.5
RTT	Reducing Transition Table	1.1
S	reducing Stack	2.4.1
SL	String List	2.2
SNLN	Surrounding Nomenclature Level Number	2.3
SR	Syllable Representation	2.2
SYL	SYLlable	2.2
TOS	TOp of reducing Stack	2.4.1

The dictionary of internal syntax of ODRA-ALGOL

AAI	Arithmetic Array Identifier
AAL	Arithmetic Array List
AAS	Arithmetic Array Segment
AC	Arithmetic Constant
ACP	ACtual Parameter
ACPL	ACtual Parameter List
AD	Array Declaration
AE	Arithmetic Expression
AFD	Arithmetic Function Designator
AFI	Arithmetic Function Identifier
ALP	Arithmetic Left Part
ALPL	Arithmetic Left Part List
APR	Arithmetic PRimary
AS	Assignment Statement
AV	Arithmetic Variable
BAI	Boolean Array Identifier
BAL	Boolean Array List
BAS	Boolean Array Segment
BC	Boolean Constant
BE	Boolean Expression
BF	Boolean Factor
BFD	Boolean Function Designator
BFI	Boolean Function Identifier
BL	BLock
BLH	BLock Head
BLP	Boolean Left Part
BLPL	Boolean Left Part List
BP	Bound Pair
BPL	Bound Pair List
BPR	Boolean PRimary
BS	Basic Statement
BV	Boolean Variable
CMS	CoMpound Statement
CMT	CoMpound Tail
CNS	CoNditional Statement
D	Declaration
DE	Designational Expression
DUMMYS	DUMMY Statement
GOTOS	GO TO Statement
FORC	FOR Clause
FORS	FOR Statement
IAV	Indexed Arithmetic Variable
IBV	Indexed Boolean Variable
IFC	IF Clause
IFS	IF Statement
L	Label
PD	Procedure Declaration
PI	Procedure Identifier

PROGR	PROGRamme
PS	Procedure Statement
R	Relational operator
REL	RELation
RPH	Reduced Procedure Heading
S	Statement
SAE	Simple Arithmetic Expression
SAV	Simple Arithmetic Variable
SBE	Simple Boolean Expression
SBV	Simple Boolean Variable
SDE	Simple Designational Expression
SL	Subscript List
STRING	STRING
SWD	SWitch Designator
SWI	SWitch Identifier
SWID	SWItch Declaration
SWL	SWitch List
UBL	Unlabelled BLock
UBS	Unlabelled Basic Statement
UCMS	Unlabelled CoMpound Statement
US	Unconditional Statement

REDUCING TRANSITION TABLES

PASS 1. ANALYSIS OF BLOCK STRUCTURE.

THE PRESENT TEXT IS EXACTLY THE PRINTED FORM OF THE DATA FOR A PROGRAMME TO GENERATE REDUCING TRANSITION TABLES ON AN ELLIOTT 803 COMPUTER. THE DATA GIVEN BELOW CONSIST OF A SET OF SYNTACTIC FORMULAE WRITTEN IN A FORM THAT CLOSELY RESEMBLES THE BACKUS-NAUR NOTATION, NAMELY

```
    @     STANDS FOR ::=
    ,     STANDS FOR THE VERTICAL LINE SEPARATING RIGHT PARTS
          OF THE FORMULAE WITH A COMMON LEFT PART
    ?     DENOTES THE END OF THE LANGUAGE SPECIFICATION.
```

THE NAMES OF THE SYNTACTIC UNITS OF THE LANGUAGE MAY BE ARBITRARY NONVOID AND AT MOST SIX CHARACTER STRINGS THAT CONTAIN NO OF THE ABOVE MENTIONED THREE CHARACTERS AND NO PAGE LAYOUT CHARACTERS. THE LATTER ARE USED AS ADDITIONAL TERMINATING SYMBOLS WITHIN A FORMULA.

THE NAMES OF THE SYNTACTIC UNITS HAVE BEEN LISTED IN THE ORDER OF THEIR OCCURRENCE IN THE SYNTACTIC FORMULAE. THE NAMES NOT LISTED BUT OCCURRING IN A FORMULA ARE EITHER SELF-EVIDENT OR THEY HAVE SOME AUXILIARY MEANING WHICH BECOMES OBVIOUS AFTER INSPECTING THE FORMULA IN QUESTION.

NAME LIST:

FORS	FOR STATEMENT
FORC	FOR CLAUSE
S	STATEMENT
L:	LABEL TOGETHER WITH A COLON
BS	BASIC STATEMENT
UBS	UNLABELLED BASIC STATEMENT
US	UNCONDITIONAL STATEMENT
PROGR	PROGRAMME
IFS	IF STATEMENT
IFC	IF CLAUSE
CNS	CONDITIONAL STATEMENT
ELSE	THE ALGOL DELIMITER *ELSE*
CMT	COMPOUND TAIL
END	
.,	SEMICOLON
BLH	BLOCK HEAD
BEGINB	A SPECIAL DELIMITER WHICH IS DELIVERED BY THE MICROANALYSER INSTEAD OF THE BLOCK *BEGIN*.
D	DECLARATION
UCMS	UNLABELLED COMPOUND STATEMENT
BEGIN	
UBL	UNLABELLED BLOCK
CMS	COMPOUND STATEMENT
BL	BLOCK
PD	PROCEDURE DECLARATION
PH.,	PROCEDURE HEADING TOGETHER WITH A SEMICOLON
NPD	NON-PROCEDURE DECLARATION

----END OF NAME LIST----

```
FORS@ FORC S' L: FORS
BS@ UBS' L: BS
US@ BS' PROGR
IFS@ IFC US
CNS@ IFS' IFS ELSES' IFC FORS' L: CNS  ELSES@ ELSE S
S@ US' CNS' FORS
CMT@ S END' S., CMT  S.,@ S .,
BLH@ BEGINB D' BLH., D  BLH.,@ BLH .,
UCMS@ BEGIN CMT
UBL@ BLH., CMT
CMS@ UCMS' L: CMS
BL@ UBL' L: BL
PD@ PH., S
D@ PD' NPD
PROGR@ CMS' BL?    PROGR IS THE DISTINGUISHED SYNTACTIC ELEMENT.

                   ----END OF DATA----



PASS 1

                 THE SET OF SYNTACTIC ELEMENTS


  FORS   FORC      S      L:     BS     UBS     US  PROGR    IFS    IFC
   CNS  ELSES   ELSE     CMT    END     S.,     .,    BLH BEGINB      D
 BLH.,   UCMS  BEGIN     UBL    CMS      BL     PD    PH.,   NPD




                THE BASIC SYMBOLS OF THE LANGUAGE


  FORC     L:    UBS    IFC   ELSE    END     ., BEGINB  BEGIN   PH.,
   NPD
```

PASS 1

STACK TABLE (ST)

NO.	TOS1	TOS	ITP
1		L:	1
2		BLH	2
3		BEGINB	3
4		BLH.,	4
5		UCMS	5
6		BEGIN	6
7		UBL	5
8		CMS	5
9		BL	5
10	FORC	FORS	7
11	FORC	FORC	6
12	FORC	S	8
13	FORC	L:	6
14	FORC	BS	7
15	FORC	UBS	7
16	FORC	US	7
17	FORC	PROGR	7
18	FORC	IFS	9
19	FORC	IFC	10
20	FORC	CNS	7
21	FORC	BLH	2
22	FORC	BEGINB	3
23	FORC	BLH.,	4
24	FORC	UCMS	7
25	FORC	BEGIN	6
26	FORC	UBL	7
27	FORC	CMS	7
28	FORC	BL	7
29	S	END	11
30	S	.,	12
31	L:	FORS	8
32	L:	FORC	6
33	L:	L:	6
34	L:	BS	13
35	L:	UBS	14
36	L:	IFS	15
37	L:	IFC	10
38	L:	CNS	16
39	L:	BLH	2
40	L:	BEGINB	3
41	L:	BLH.,	4
42	L:	UCMS	17
43	L:	BEGIN	6
44	L:	UBL	18
45	L:	CMS	19
46	L:	BL	20
47	IFS	ELSES	16
48	IFS	ELSE	6
49	IFC	FORS	16
50	IFC	FORC	6
51	IFC	L:	10
52	IFC	BS	21
53	IFC	UBS	21
54	IFC	US	22
55	IFC	PROGR	21
56	IFC	BLH	2
57	IFC	BEGINB	3
58	IFC	BLH.,	4
59	IFC	UCMS	21
60	IFC	BEGIN	6
61	IFC	UBL	21
62	IFC	CMS	21
63	IFC	BL	21
64	ELSE	FORS	7
65	ELSE	FORC	6
66	ELSE	S	23
67	ELSE	L:	6
68	ELSE	BS	7
69	ELSE	UBS	7
70	ELSE	US	7
71	ELSE	PROGR	7
72	ELSE	IFS	9
73	ELSE	IFC	10
74	ELSE	CNS	7
75	ELSE	BLH	2
76	ELSE	BEGINB	3
77	ELSE	BLH.,	4
78	ELSE	UCMS	7
79	ELSE	BEGIN	6
80	ELSE	UBL	7
81	ELSE	CMS	7
82	ELSE	BL	7
83	S.,	FORS	7
84	S.,	FORC	6
85	S.,	S	24
86	S.,	L:	6
87	S.,	BS	7
88	S.,	UBS	7
89	S.,	US	7
90	S.,	PROGR	7
91	S.,	IFS	9
92	S.,	IFC	10
93	S.,	CNS	7
94	S.,	CMT	11
95	S.,	S.,	6
96	S.,	BLH	2
97	S.,	BEGINB	3
98	S.,	BLH.,	4
99	S.,	UCMS	7
100	S.,	BEGIN	6

```
PASS 1

                          STACK TABLE (ST)

  NO.   TOS1      TOS   ITP          NO.   TOS1      TOS   ITP

  101     S.,     UBL     7          151  BEGIN   BEGIN     6
  102     S.,     CMS     7          152  BEGIN     UBL     7
  103     S.,      BL     7          153  BEGIN     CMS     7
  104     BLH      .,    25          154  BEGIN      BL     7
  105  BEGINB       D    26          155    PH.,   FORS    30
  106  BEGINB      PD    27          156    PH.,   FORC     6
  107  BEGINB    PH.,     6          157    PH.,      S    31
  108  BEGINB     NPD    27          158    PH.,     L:     6
  109    BLH.,   FORS     7          159    PH.,     BS    30
  110    BLH.,   FORC     6          160    PH.,    UBS    30
  111    BLH.,      S    24          161    PH.,     US    30
  112    BLH.,     L:     6          162    PH.,  PROGR    30
  113    BLH.,     BS     7          163    PH.,    IFS    32
  114    BLH.,    UBS     7          164    PH.,    IFC    10
  115    BLH.,     US     7          165    PH.,    CNS    30
  116    BLH.,  PROGR     7          166    PH.,    BLH     2
  117    BLH.,    IFS     9          167    PH.,  BEGINB    3
  118    BLH.,    IFC    10          168    PH.,   BLH.,    4
  119    BLH.,    CNS     7          169    PH.,   UCMS    30
  120    BLH.,    CMT    28          170    PH.,  BEGIN     6
  121    BLH.,    S.,     6          171    PH.,    UBL    30
  122    BLH.,    BLH     2          172    PH.,    CMS    30
  123    BLH., BEGINB     3          173    PH.,     BL    30
  124    BLH.,      D    26
  125    BLH.,  BLH.,     4
  126    BLH.,   UCMS     7
  127    BLH.,  BEGIN     6
  128    BLH.,    UBL     7
  129    BLH.,    CMS     7
  130    BLH.,     BL     7
  131    BLH.,     PD    27
  132    BLH.,   PH.,     6
  133    BLH.,    NPD    27
  134   BEGIN    FORS     7
  135   BEGIN    FORC     6
  136   BEGIN       S    24
  137   BEGIN      L:     6
  138   BEGIN      BS     7
  139   BEGIN     UBS     7
  140   BEGIN      US     7
  141   BEGIN   PROGR     7
  142   BEGIN     IFS     9
  143   BEGIN     IFC    10
  144   BEGIN     CNS     7
  145   BEGIN     CMT    29
  146   BEGIN     S.,     6
  147   BEGIN     BLH     2
  148   BEGIN  BEGINB     3
  149   BEGIN   BLH.,     4
  150   BEGIN    UCMS     7
```

```
PASS 1

     INTERMEDIATE TABLE (IT)

 NO.      LB   UB        NO.      LB   UB

   1       1    3
   2       4    4
   3       5    6
   4       7   14
   5      15   15
   6      16   21
   7      22   23
   8      24   25
   9      26   28
  10      29   33
  11      34   37
  12      38   43
  13      44   46
  14      47   49
  15      50   52
  16      53   54
  17      55   58
  18      59   62
  19      63   66
  20      67   70
  21      71   73
  22      74   76
  23      77   78
  24      79   80
  25      81   88
  26      89   89
  27      90   90
  28      91   94
  29      95   98
  30      99   99
  31     100  100
  32     101  102
```

PASS 1

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDUCTION
	1	L:	3	
	2	BEGINB	3	
	3	BEGIN	3	
*	4	.,	3	
	5	PH.,	3	
	6	NPD	3	
*	7	FORC	3	
*	8	L:	3	
*	9	UBS	3	
*	10	IFC	3	
*	11	BEGINB	3	
*	12	BEGIN	3	
*	13	PH.,	3	
*	14	NPD	3	
	15		1	PROGR
*	16	FORC	3	
*	17	L:	3	
*	18	UBS	3	
*	19	IFC	3	
*	20	BEGINB	3	
*	21	BEGIN	3	
	22	END	1	S
	23	.,	1	S
*	24	END	2	FORS
*	25	.,	2	FORS
	26	ELSE	3	
	27	END	1	S
	28	.,	1	S
*	29	FORC	3	
*	30	L:	3	
*	31	UBS	3	
*	32	BEGINB	3	
*	33	BEGIN	3	
	34		2	CMT
	35	ELSE	2	CMT
	36	END	2	CMT
	37	.,	2	CMT
*	38	FORC	2	S.,
*	39	L:	2	S.,
*	40	UBS	2	S.,
*	41	IFC	2	S.,
*	42	BEGINB	2	S.,
*	43	BEGIN	2	S.,
	44	ELSE	2	BS
	45	END	2	BS
	46	.,	2	BS
*	47	ELSE	1	BS
*	48	END	1	BS
*	49	.,	1	BS
	50	ELSE	3	
	51	END	1	CNS
	52	.,	1	CNS
*	53	END	2	CNS
*	54	.,	2	CNS
	55		1	CMS
	56	ELSE	1	CMS
	57	END	1	CMS
	58	.,	1	CMS
*	59		1	BL
*	60	ELSE	1	BL
*	61	END	1	BL
*	62	.,	1	BL
	63		2	CMS
	64	ELSE	2	CMS
	65	END	2	CMS
	66	.,	2	CMS
*	67		2	BL
*	68	ELSE	2	BL
*	69	END	2	BL
*	70	.,	2	BL
	71	ELSE	1	US
	72	END	1	US
	73	.,	1	US
*	74	ELSE	2	IFS
*	75	END	2	IFS
*	76	.,	2	IFS
	77	END	2	ELSES
	78	.,	2	ELSES
*	79	END	3	
*	80	.,	3	
	81	FORC	2	BLH.,
	82	L:	2	BLH.,
	83	UBS	2	BLH.,
	84	IFC	2	BLH.,
	85	BEGINB	2	BLH.,
	86	BEGIN	2	BLH.,
	87	PH.,	2	BLH.,
	88	NPD	2	BLH.,
*	89	.,	2	BLH
	90	.,	1	D
*	91		2	UBL
*	92	ELSE	2	UBL
*	93	END	2	UBL
*	94	.,	2	UBL
	95		2	UCMS
	96	ELSE	2	UCMS
	97	END	2	UCMS
	98	.,	2	UCMS
*	99	.,	1	S
	100	.,	2	PD
*	101	ELSE	3	
*	102	.,	1	S

```
A WORKED EXAMPLE FOR PASS 1:  THE FORMAL TEXT IS FOLLOWED BY THE
CORRESPONDING REDUCING STACK HISTORY.

BEGINB PH.,
 BEGINB NPD .,
  UBS .,
  FORC UBS .,
  UBS
 END .,
 NPD .,
 UBS .,
 UBS
END '

   NS    P     L  S(P-1)    S(P)      LS   ST     RED

    2    1     1          BEGINB     PH.,   3
    3    2     2  BEGINB    PH.,  BEGINB    3
    4    3     3    PH.,  BEGINB     NPD    3
    5    4     4  BEGINB     NPD      .,    1       D
    5    4     5  BEGINB       D      .,    2     BLH
    5    3     6    PH.,     BLH      .,    3
    6    4     7     BLH      .,     UBS    2    BLH.,
    6    3     8    PH.,   BLH.,     UBS    3
    7    4     9   BLH.,     UBS      .,    1       S
    7    4    10   BLH.,       S      .,    3
    8    5    11       S      .,    FORC    2      S.,
    8    4    12   BLH.,      S.,   FORC    3
    9    5    13     S.,    FORC     UBS    3
   10    6    14    FORC     UBS      .,    1       S
   10    6    15    FORC       S      .,    2    FORS
   10    5    16     S.,    FORS      .,    1       S
   10    5    17     S.,       S      .,    3
   11    6    18       S      .,     UBS    2      S.,
   11    5    19     S.,      S.,    UBS    3
   12    6    20     S.,     UBS     END    1       S
   12    6    21     S.,       S     END    3
   13    7    22       S     END      .,    2     CMT
   13    6    23     S.,     CMT      .,    2     CMT
   13    5    24     S.,     CMT      .,    2     CMT
   13    4    25   BLH.,     CMT      .,    2     UBL
   13    3    26    PH.,     UBL      .,    1       S
   13    3    27    PH.,       S      .,    2      PD
   13    2    28  BEGINB      PD      .,    1       D
   13    2    29  BEGINB       D      .,    2     BLH
   13    1    30             BLH      .,    3
   14    2    31     BLH      .,     NPD    2    BLH.,
   14    1    32           BLH.,     NPD    3
   15    2    33   BLH.,     NPD      .,    1       D
   15    2    34   BLH.,       D      .,    2     BLH
   15    1    35             BLH      .,    3
   16    2    36     BLH      .,     UBS    2    BLH.,
   16    1    37           BLH.,     UBS    3
   17    2    38   BLH.,     UBS      .,    1       S
   17    2    39   BLH.,       S      .,    3
   18    3    40       S      .,     UBS    2      S.,
   18    2    41   BLH.,      S.,    UBS    3
   19    3    42     S.,     UBS     END    1       S
   19    3    43     S.,       S     END    3
   20    4    44       S     END            2     CMT
   20    3    45     S.,     CMT            2     CMT
   20    2    46   BLH.,     CMT            2     UBL
   20    1    47             UBL            1   PROGR
```

PASS 2. THE COMPLETE SYNTAX CHECK

THE CONVENTIONS USED TO DESCRIBE THE FORMAL LANGUAGE FOR PASS 1 ARE ALSO USED HERE. PASS 2 DEALS WITH A SLIGHTLY PREPROCESSED ALGOL TEXT: UNARY ADDING OPERATORS DIFFER NOW FROM BINARY ONES, ALL TYPE DECLARATIONS, FORMAL PARAMETER LISTS, VALUE PARTS AND SPECIFICATION PARTS HAVE BEEN DISCARDED AND THE RELEVANT INFORMATION IS NOW IN A LIST OF NAME DESCRIPTORS TAGGED TOGETHER WITH THE BLOCK STRUCTURE DESCRIPTION GENERATED IN PASS 1.

THE RESTRICTIONS IMPOSED ON THE REFERENCE LANGUAGE IN ODRA-ALGOL ARE TAKEN INTO ACCOUNT. THESE ARE MINOR AND OF NO IMPORTANCE HERE EXCEPT OF THE THREE GIVEN BELOW WHICH ARE RATHER SERIOUS, NAMELY

1. ALL FORMAL PARAMETERS MUST HAVE SPECIFICATIONS.
2. A FORMAL PARAMETER MUST NOT BE SPECIFIED TO BE A PROCEDURE.
3. A STRICT SYNTACTIC AND SEMANTIC DISTINCTION IS MADE BETWEEN PROCEDURES AND TYPE PROCEDURES. A FUNCTION DESIGNATOR MUST NOT BE USED TO FORM A PROPER PROCEDURE STATEMENT AND, CONVERSELY, A PROPER PROCEDURE STATEMENT MUST NOT BE USED TO FORM A FUNCTION DESIGNATOR.

THE ABOVE RESTRICTIONS ARE RELATED TO THE PROBLEM OF REDUCIBILITY AND UNAMBIGUITY OF THE LANGUAGE, AS NOTED ELSEWHERE. IT COULD BE OBSERVED THAT THE THIRD OF THE ABOVE RESTRICTIONS IS AN *UNEXPRESSED AGREEMENT* IN SOME BOOKS ON ALGOL 60.

THE PROBLEM OF STANDARD PROCEDURES REQUIRES NO SPECIAL ATTENTION AS IN ODRA-ALGOL THE USE OF THESE PROCEDURES DOES NOT VIOLATE THE SYNTAX OF ALGOL 60.

ATTENTION IS DRAWN TO THE FACT THAT A PART OF INFORMAL SEMANTICS OF ALGOL 60 IS PUT INTO THE FORMAL SYNTAX GIVEN BELOW.

NAME LIST:

SL	SUBSCRIPT LIST
AE	ARITHMETIC EXPRESSION
(*	LEFT SQUARE BRACKET
IAV	INDEXED ARITHMETIC VARIABLE
AAI	ARITHMETIC ARRAY IDENTIFIER
*)	RIGHT SQUARE BRACKET
AV	ARITHMETIC VARIABLE
SAV	SIMPLE ARITHMETIC VARIABLE
ACP	ACTUAL PARAMETER
BE	BOOLEAN EXPRESSION
DE	DESIGNATIONAL EXPRESSION
STRING	STRING (DO NOT MIX WITH A SPECIFIER IN THE REFERENCE LANGUAGE)
BAI	BOOLEAN ARRAY IDENTIFIER
SWI	SWITCH IDENTIFIER
ACPL	ACTUAL PARAMETER LIST
AFD	ARITHMETIC FUNCTION DESIGNATOR
AFI	ARITHMETIC FUNCTION IDENTIFIER
APR	ARITHMETIC PRIMARY
AC	ARITHMETIC CONSTANT
SAE	SIMPLE ARITHMETIC EXPRESSION
-	UNARY ARITHMETIC OPERATOR

```
%       BINARY ARITHMETIC OPERATOR
IFC     IF CLAUSE
REL     RELATION
R       RELATIONAL OPERATOR
BPR     BOOLEAN PRIMARY
BC      BOOLEAN CONSTANT
BV      BOOLEAN VARIABLE
BFD     BOOLEAN FUNCTION DESIGNATOR
BF      BOOLEAN FACTOR
£       UNARY BOOLEAN OPERATOR
SBE     SIMPLE BOOLEAN EXPRESSION
$       BINARY BOOLEAN OPERATOR
SBV     SIMPLE BOOLEAN VARIABLE
IBV     INDEXED BOOLEAN VARIABLE
BFI     BOOLEAN FUNCTION IDENTIFIER
L       LABEL
SWD     SWITCH DESIGNATOR
SDE     SIMPLE DESIGNATIONAL EXPRESSION
PS      PROCEDURE STATEMENT
PI      PROCEDURE IDENTIFIER
ALP     ARITHMETIC LEFT PART
BLP     BOOLEAN LEFT PART
ALPL    ARITHMETIC LEFT PART LIST
BLPL    BOOLEAN LEFT PART LIST
AS      ASSIGNMENT STATEMENT
GOTOS   GO TO STATEMENT
FORC    FOR CLAUSE
FORS    FOR STATEMENT
S       STATEMENT
UBS     UNLABELLED BASIC STATEMENT
DUMMYS  DUMMY STATEMENT (INSERTED INTO THE TEXT IN PASS 1, WHERE
                                                           NECESSARY)
BS      BASIC STATEMENT
US      UNCONDITIONAL STATEMENT
PROGR   PROGRAMME
IFS     IF STATEMENT
CNS     CONDITIONAL STATEMENT
CMT     COMPOUND TAIL
.,      SEMICOLON
BLH     BLOCK HEAD
BEGINB  A SPECIAL BLOCK BEGIN INTRODUCED IN PASS 1
D       DECLARATION
UCMS    UNLABELLED COMPOUND STATEMENT
UBL     UNLABELLED BLOCK
CMS     COMPOUND STATEMENT
BL      BLOCK
BP      BOUND PAIR
BPL     BOUND PAIR LIST
AAS     ARITHMETIC ARRAY SEGMENT
BAS     BOOLEAN ARRAY SEGMENT
AAL     ARITHMETIC ARRAY LIST
BAL     BOOLEAN ARRAY LIST
AD      ARRAY DECLARATION
SWL     SWITCH LIST
SWID    SWITCH DECLARATION
RPH     REDUCED (IN PASS 1) PROCEDURE HEADING
PROCED  THE ALGOL DELIMITER *PROCEDURE*
PD      PROCEDURE DECLARATION

                   ----END OF NAME LIST----
```

```
SL@ (*AE' SL, AE  (*AE@ (* AE  SL,@ SL ,
IAV@ AAI SL*)  SL*)@ SL *)
AV@ IAV' SAV
ACP@ AE' BE' DE' STRING' AAI' BAI' SWI
ACPL@ ACP' ACPL, ACP  ACPL,@ ACPL ,
AFD@ AFI' AFI( ACPL)  AFI(@ AFI (  ACPL)@ ACPL )
APR@ AV' AC' (AE )' AFD  (AE@ ( AE
SAE@ APR' - APR' SAE% APR  SAE%@ SAE %
AE@ SAE' IFC SAEEAE  SAEEAE@ SAE EAE  EAE@ ELSE AE
IFC@ IFBE THEN  IFBE@ IF BE

REL@ SAER SAE  SAER@ SAE R
BPR@ BV' BC' REL' BFD' (BE )  (BE@ ( BE
BF@ BPR' £ BPR
SBE@ BF' SBE$ BF  SBE$@ SBE $
BE@ SBE' IFC SBEEBE  SBEEBE@ SBE EBE  EBE@ ELSE BE
BV@ SBV' IBV  IBV@ BFI SL*)
BFD@ BFI' BFI( ACPL)  BFI(@ BFI (

L:@ L :
SWD@ SWI (*AE*)  (*AE*)@ (*AE *)
SDE@ L' SWD' (DE )  (DE@ ( DE
DE@  SDE' IFC SDEEDE  SDEEDE@ SDE EDE  EDE@ ELSE DE

PS@ PI' PI( ACPL)  PI(@ PI (

ALP@ AV:=' AFI :=  AV:=@ AV :=
BLP@ BV :=' BFI :=
ALPL@ ALP' ALPL ALP
BLPL@ BLP' BLPL BLP
AS@ ALPL AE' BLPL BE

GOTOS@ GOTO DE

FOR1@ FOR AV:=
ESTEP@ AESTAE UNAE  AESTAE@ AE STAE  STAE@ STEP AE  UNAE@ UNTIL AE
EWH@ AE WHBE  WHBE@ WHILE BE
FOR2@ FOR1 AE' FOR1 ESTEP' FOR1 EWH' FOR2, AE' FOR2, ESTEP' FOR2, EWH
   FOR2,@ FOR2 ,
FORC@ FOR2 DO
FORS@ FORC S' L: FORS

UBS@ AS' GOTOS' PS' DUMMYS
BS@ UBS' L: BS
US@ BS' PROGR
IFS@ IFC US
CNS@ IFS ELSES' IFC FORS' IFS' L: CNS  ELSES@ ELSE S
S@ US' CNS' FORS
CMT@ S END' S., CMT  S.,@ S .,
BLH@ BEGINB D' BLH., D  BLH.,@ BLH .,
UCMS@ BEGIN CMT
UBL@ BEGINB CMT' BLH., CMT
CMS@ UCMS' L: CMS
BL@ UBL' L: BL
```

```
BP@ AE: AE  AE:@ AE :
BPL@ BP' BPL, BP  BPL,@ BPL ,
AAS@ AAI (*BP*)' AAI, AAS  (*BP*)@ (*BPL *)  (*BPL@ (* BPL  AAI,@ AAI ,
BAS@ BAI (*BP*)' BAI, BAS  BAI,@ BAI ,
AAL@ AAS' AAL, AAS  AAL,@ AAL ,
BAL@ BAS' BAL, BAS  BAL,@ BAL ,
AD@ ARRAY AAL' ARRAY BAL

SWL@ := DE' SWL, DE  SWL,@ SWL ,
SWID@ SWITCH SWISWL  SWISWL@ SWI SWL

RPH@ PROCED AFI' PROCED BFI' PROCED PI
PD@ RPH., S  RPH.,@ RPH .,
D@ PD' AD' SWID
PROGR@ CMS' BL?  PROGR IS THE DISTINGUISHED SYNTACTIC UNIT.

                     ----END OF DATA----
```

```
PASS 2

                    THE SET OF SYNTACTIC ELEMENTS

      SL    (*AE     SL,      AE      (*       ,     IAV     AAI    SL*)      *)
      AV     SAV     ACP      BE      DE  STRING     BAI     SWI    ACPL   ACPL,
     AFD     AFI    AFI(   ACPL)       )       (     APR      AC     (AE     SAE
       -    SAE%       %     IFC  SAEEAE     EAE    ELSE    IFBE    THEN      IF
     REL    SAER       R     BPR      BC      BV     BFD     (BE      BF       £
     SBE    SBE$       $  SBEEBE     EBE     SBV     IBV     BFI    BFI(      L:
       L       :     SWD  (*AE*)     SDE     (DE  SDEEDE     EDE      PS      PI
     PI(     ALP    AV:=      :=     BLP    ALPL    BLPL      AS   GOTOS    GOTO
    FOR1     FOR   ESTEP  AESTAE    UNAE    STAE    STEP   UNTIL     EWH    WHBE
   WHILE    FOR2   FOR2,    FORC      DO    FORS       S     UBS  DUMMYS      BS
      US   PROGR     IFS     CNS   ELSES     CMT     END     S.,      .,     BLH
  BEGINB       D    BLH.,   UCMS   BEGIN     UBL     CMS      BL      BP     AE:
     BPL    BPL,     AAS  (*BP*)    AAI,   (*BPL     BAS    BAI,     AAL    AAL,
     BAL    BAL,      AD   ARRAY     SWL    SWL,    SWID  SWITCH  SWISWL     RPH
  PROCED      PD    RPH.,


                    THE BASIC SYMBOLS OF THE LANGUAGE

      (*       ,     AAI      *)     SAV  STRING     BAI     SWI     AFI       )
       (      AC       -       %    ELSE    THEN      IF       R      BC       £
       $     SBV     BFI       L       :      PI      :=    GOTO     FOR    STEP
   UNTIL   WHILE      DO  DUMMYS     END      .,  BEGINB   BEGIN   ARRAY  SWITCH
  PROCED
```

PASS 2

STACK TABLE (ST)

NO.	TOS1	TOS	ITP	NO.	TOS1	TOS	ITP
1		L:	1	51	(*	SAE%	18
2		L	2	52	(*	IFC	19
3		BLH	3	53	(*	IFBE	20
4		BEGINB	4	54	(*	IF	21
5		BLH.,	4	55	(*	BP	29
6		UCMS	5	56	(*	AE:	15
7		BEGIN	6	57	(*	BPL	30
8		UBL	5	58	(*	BPL,	15
9		CMS	5	59	AAI	SL	31
10		BL	5	60	AAI	(*AE	32
11	SL	,	7	61	AAI	SL,	15
12	SL	*)	8	62	AAI	(*	15
13	(*AE	*)	9	63	AAI	,	33
14	SL,	AE	10	64	AAI	SL*)	34
15	SL,	IAV	11	65	AAI	(*BP*)	35
16	SL,	AAI	12	66	AAI	(*BPL	36
17	SL,	AV	11	67	AV	:=	37
18	SL,	SAV	11	68	BAI	SL	31
19	SL,	AFD	11	69	BAI	(*AE	32
20	SL,	AFI	13	70	BAI	SL,	15
21	SL,	AFI(	14	71	BAI	(*	15
22	SL,	(	15	72	BAI	,	38
23	SL,	APR	11	73	BAI	SL*)	39
24	SL,	AC	11	74	BAI	(*BP*)	40
25	SL,	(AE	16	75	BAI	(*BPL	36
26	SL,	SAE	17	76	SWI	(*AE	36
27	SL,	-	18	77	SWI	(*	15
28	SL,	SAE%	18	78	SWI	(*AE*)	41
29	SL,	IFC	19	79	SWI	:=	42
30	SL,	IFBE	20	80	SWI	SWL	43
31	SL,	IF	21	81	SWI	SWL,	42
32	AE	:	22	82	ACPL	,	44
33	AE	STAE	23	83	ACPL	)	45
34	AE	STEP	15	84	ACPL,	AE	46
35	AE	WHBE	24	85	ACPL,	IAV	47
36	AE	WHILE	21	86	ACPL,	AAI	48
37	(*	AE	25	87	ACPL,	AV	47
38	(*	IAV	26	88	ACPL,	SAV	47
39	(*	AAI	12	89	ACPL,	ACP	49
40	(*	AV	26	90	ACPL,	BE	46
41	(*	SAV	26	91	ACPL,	DE	46
42	(*	AFD	26	92	ACPL,	STRING	46
43	(*	AFI	27	93	ACPL,	BAI	48
44	(*	AFI(	14	94	ACPL,	SWI	48
45	(*	(	15	95	ACPL,	AFD	47
46	(*	APR	26	96	ACPL,	AFI	50
47	(*	AC	26	97	ACPL,	AFI(	14
48	(*	(AE	16	98	ACPL,	(	51
49	(*	SAE	28	99	ACPL,	APR	47
50	(*	-	18	100	ACPL,	AC	47

PASS 2

STACK TABLE (ST)

NO.	TOS1	TOS	ITP	NO.	TOS1	TOS	ITP
101	ACPL,	(AE	16	151	AFI(	-	18
102	ACPL,	SAE	52	152	AFI(	SAE%	18
103	ACPL,	-	18	153	AFI(	IFC	53
104	ACPL,	SAE%	18	154	AFI(	IFBE	20
105	ACPL,	IFC	53	155	AFI(	IF	21
106	ACPL,	IFBE	20	156	AFI(	REL	68
107	ACPL,	IF	21	157	AFI(	SAER	19
108	ACPL,	REL	54	158	AFI(	BPR	68
109	ACPL,	SAER	19	159	AFI(	BC	68
110	ACPL,	BPR	54	160	AFI(	BV	68
111	ACPL,	BC	54	161	AFI(	BFD	68
112	ACPL,	BV	54	162	AFI(	(BE	16
113	ACPL,	BFD	54	163	AFI(	BF	68
114	ACPL,	(BE	16	164	AFI(	£	55
115	ACPL,	BF	54	165	AFI(	SBE	69
116	ACPL,	£	55	166	AFI(	SBE$	57
117	ACPL,	SBE	56	167	AFI(	SBV	68
118	ACPL,	SBE$	57	168	AFI(	IBV	68
119	ACPL,	SBV	54	169	AFI(	BFI	70
120	ACPL,	IBV	54	170	AFI(	BFI(	14
121	ACPL,	BFI	58	171	AFI(	L	61
122	ACPL,	BFI(	14	172	AFI(	SWD	61
123	ACPL,	L	46	173	AFI(	SDE	61
124	ACPL,	SWD	46	174	AFI(	(DE	16
125	ACPL,	SDE	46	175	(	AE	71
126	ACPL,	(DE	16	176	(	IAV	72
127	AFI	(	59	177	(	AAI	12
128	AFI	:=	60	178	(	AV	72
129	AFI(	AE	61	179	(	SAV	72
130	AFI(	IAV	62	180	(	BE	73
131	AFI(	AAI	63	181	(	DE	74
132	AFI(	AV	62	182	(	BAI	12
133	AFI(	SAV	62	183	(	SWI	12
134	AFI(	ACP	61	184	(	AFD	72
135	AFI(	BE	61	185	(	AFI	75
136	AFI(	DE	61	186	(	AFI(	14
137	AFI(	STRING	61	187	(	(	51
138	AFI(	BAI	63	188	(	APR	72
139	AFI(	SWI	63	189	(	AC	72
140	AFI(	ACPL	64	190	(	(AE	16
141	AFI(	ACPL,	14	191	(	SAE	76
142	AFI(	AFD	62	192	(	-	18
143	AFI(	AFI	65	193	(	SAE%	18
144	AFI(	AFI(	14	194	(	IFC	53
145	AFI(	ACPL)	66	195	(	IFBE	20
146	AFI(	(	51	196	(	IF	21
147	AFI(	APR	62	197	(	REL	77
148	AFI(	AC	62	198	(	SAER	19
149	AFI(	(AE	16	199	(	BPR	77
150	AFI(	SAE	67	200	(	BC	77

PASS 2

STACK TABLE (ST)

NO.	TOS1	TOS	ITP	NO.	TOS1	TOS	ITP
201	(	BV	77	251	IFC	AFI(	14
202	(	BFD	77	252	IFC	(	51
203	(	(BE	16	253	IFC	APR	90
204	(	BF	77	254	IFC	AC	90
205	(	£	55	255	IFC	(AE	16
206	(	SBE	78	256	IFC	SAE	92
207	(	SBE$	57	257	IFC	-	18
208	(	SBV	77	258	IFC	SAE%	18
209	(	IBV	77	259	IFC	SAEEAE	93
210	(	BFI	79	260	IFC	REL	94
211	(	BFI(	14	261	IFC	SAER	19
212	(	L	80	262	IFC	BPR	94
213	(	SWD	80	263	IFC	BC	94
214	(	SDE	80	264	IFC	BV	95
215	(	(DE	16	265	IFC	BFD	94
216	(AE	)	81	266	IFC	(BE	16
217	SAE	%	82	267	IFC	BF	94
218	SAE	EAE	83	268	IFC	£	55
219	SAE	ELSE	15	269	IFC	SBE	96
220	SAE	R	84	270	IFC	SBE$	57
221	-	IAV	85	271	IFC	SBEEBE	97
222	-	AAI	12	272	IFC	SBV	98
223	-	AV	85	273	IFC	IBV	98
224	-	SAV	85	274	IFC	BFI	99
225	-	AFD	85	275	IFC	BFI(	14
226	-	AFI	86	276	IFC	L:	100
227	-	AFI(	14	277	IFC	L	101
228	-	(	15	278	IFC	SWD	102
229	-	APR	87	279	IFC	SDE	103
230	-	AC	85	280	IFC	(DE	16
231	-	(AE	16	281	IFC	SDEEDE	104
232	SAE%	IAV	85	282	IFC	PS	105
233	SAE%	AAI	12	283	IFC	PI	106
234	SAE%	AV	85	284	IFC	PI(	14
235	SAE%	SAV	85	285	IFC	ALP	107
236	SAE%	AFD	85	286	IFC	AV:=	107
237	SAE%	AFI	86	287	IFC	BLP	108
238	SAE%	AFI(	14	288	IFC	ALPL	15
239	SAE%	(	15	289	IFC	BLPL	21
240	SAE%	APR	87	290	IFC	AS	105
241	SAE%	AC	85	291	IFC	GOTOS	105
242	SAE%	(AE	16	292	IFC	GOTO	42
243	IFC	IAV	88	293	IFC	FOR1	15
244	IFC	AAI	12	294	IFC	FOR	109
245	IFC	AV	89	295	IFC	FOR2	110
246	IFC	SAV	88	296	IFC	FOR2,	15
247	IFC	BAI	12	297	IFC	FORC	6
248	IFC	SWI	12	298	IFC	FORS	111
249	IFC	AFD	90	299	IFC	UBS	105
250	IFC	AFI	91	300	IFC	DUMMYS	105

PASS 2

STACK TABLE (ST)

NO.	TOS1	TOS	ITP	NO.	TOS1	TOS	ITP
301	IFC	BS	105	351	ELSE	SWD	128
302	IFC	US	112	352	ELSE	SDE	128
303	IFC	PROGR	105	353	ELSE	(DE	16
304	IFC	BLH	3	354	ELSE	PS	129
305	IFC	BEGINB	4	355	ELSE	PI	130
306	IFC	BLH.,	4	356	ELSE	PI(	14
307	IFC	UCMS	105	357	ELSE	ALP	107
308	IFC	BEGIN	6	358	ELSE	AV:=	107
309	IFC	UBL	105	359	ELSE	BLP	108
310	IFC	CMS	105	360	ELSE	ALPL	15
311	IFC	BL	105	361	ELSE	BLPL	21
312	ELSE	AE	113	362	ELSE	AS	129
313	ELSE	IAV	114	363	ELSE	GOTOS	129
314	ELSE	AAI	12	364	ELSE	GOTO	42
315	ELSE	AV	115	365	ELSE	FOR1	15
316	ELSE	SAV	114	366	ELSE	FOR	109
317	ELSE	BE	116	367	ELSE	FOR2	110
318	ELSE	DE	117	368	ELSE	FOR2,	15
319	ELSE	BAI	12	369	ELSE	FORC	6
320	ELSE	SWI	12	370	ELSE	FORS	129
321	ELSE	AFD	118	371	ELSE	S	131
322	ELSE	AFI	119	372	ELSE	UBS	129
323	ELSE	AFI(	14	373	ELSE	DUMMYS	129
324	ELSE	(	51	374	ELSE	BS	129
325	ELSE	APR	118	375	ELSE	US	129
326	ELSE	AC	118	376	ELSE	PROGR	129
327	ELSE	(AE	16	377	ELSE	IFS	132
328	ELSE	SAE	120	378	ELSE	CNS	129
329	ELSE	-	18	379	ELSE	BLH	3
330	ELSE	SAE%	18	380	ELSE	BEGINB	4
331	ELSE	IFC	121	381	ELSE	BLH.,	4
332	ELSE	IFBE	20	382	ELSE	UCMS	129
333	ELSE	IF	21	383	ELSE	BEGIN	6
334	ELSE	REL	122	384	ELSE	UBL	129
335	ELSE	SAER	19	385	ELSE	CMS	129
336	ELSE	BPR	122	386	ELSE	BL	129
337	ELSE	BC	122	387	IFBE	THEN	133
338	ELSE	BV	123	388	IF	IAV	134
339	ELSE	BFD	122	389	IF	AAI	12
340	ELSE	(BE	16	390	IF	AV	134
341	ELSE	BF	122	391	IF	SAV	134
342	ELSE	£	55	392	IF	BE	135
343	ELSE	SBE	124	393	IF	BAI	12
344	ELSE	SBE$	57	394	IF	AFD	134
345	ELSE	SBV	125	395	IF	AFI	136
346	ELSE	IBV	125	396	IF	AFI(	14
347	ELSE	BFI	126	397	IF	(	21
348	ELSE	BFI(	14	398	IF	APR	134
349	ELSE	L:	6	399	IF	AC	134
350	ELSE	L	127	400	IF	(AE	16

PASS 2

STACK TABLE (ST)

NO.	TOS1	TOS	ITP	NO.	TOS1	TOS	ITP
401	IF	SAE	137	451	£	-	18
402	IF	-	18	452	£	SAE%	18
403	IF	SAE%	18	453	£	REL	146
404	IF	IFC	57	454	£	SAER	19
405	IF	IFBE	20	455	£	BPR	147
406	IF	IF	21	456	£	BC	146
407	IF	REL	138	457	£	BV	146
408	IF	SAER	19	458	£	BFD	146
409	IF	BPR	138	459	£	(BE	16
410	IF	BC	138	460	£	SBV	146
411	IF	BV	138	461	£	IBV	146
412	IF	BFD	138	462	£	BFI	148
413	IF	(BE	16	463	£	BFI(	14
414	IF	BF	138	464	SBE	ELSE	21
415	IF	£	55	465	SBE	$	149
416	IF	SBE	139	466	SBE	EBE	150
417	IF	SBE$	57	467	SBE$	IAV	134
418	IF	SBV	138	468	SBE$	AAI	12
419	IF	I3V	138	469	SBE$	AV	134
420	IF	BFI	140	470	SBE$	SAV	134
421	IF	BFI(	14	471	SBE$	BAI	12
422	SAER	IAV	141	472	SBE$	AFD	134
423	SAER	AAI	12	473	SBE$	AFI	136
424	SAER	AV	141	474	SBE$	AFI(	14
425	SAER	SAV	141	475	SBE$	(	21
426	SAER	AFD	141	476	SBE$	APR	134
427	SAER	AFI	142	477	SBE$	AC	134
428	SAER	AFI(	14	478	SBE$	(AE	16
429	SAER	(	15	479	SBE$	SAE	137
430	SAER	APR	141	480	SBE$	-	18
431	SAER	AC	141	481	SBE$	SAE%	18
432	SAER	(AE	16	482	SBE$	REL	151
433	SAER	SAE	143	483	SBE$	SAER	19
434	SAER	-	18	484	SBE$	BPR	151
435	SAER	SAE%	18	485	SBE$	BC	151
436	BV	:=	144	486	SBE$	BV	151
437	(BE	)	145	487	SBE$	BFD	151
438	£	IAV	134	488	SBE$	(BE	16
439	£	AAI	12	489	SBE$	BF	152
440	£	AV	134	490	SBE$	£	55
441	£	SAV	134	491	SBE$	SBV	151
442	£	BAI	12	492	SBE$	IBV	151
443	£	AFD	134	493	SBE$	BFI	153
444	£	AFI	136	494	SBE$	BFI(	14
445	£	AFI(	14	495	BFI	(	154
446	£	(	21	496	BFI	:=	144
447	£	APR	134	497	BFI(	AE	61
448	£	AC	134	498	BFI(	IAV	62
449	£	(AE	16	499	BFI(	AAI	63
450	£	SAE	137	500	BFI(	AV	62

PASS 2

STACK TABLE (ST)

NO.	TOS1	TOS	ITP
501	BFI(	SAV	62
502	BFI(	ACP	61
503	BFI(	BE	61
504	BFI(	DE	61
505	BFI(	STRING	61
506	BFI(	BAI	63
507	BFI(	SWI	63
508	BFI(	ACPL	64
509	BFI(	ACPL,	14
510	BFI(	AFD	62
511	BFI(	AFI	65
512	BFI(	AFI(	14
513	BFI(	ACPL)	155
514	BFI(	(	51
515	BFI(	APR	62
516	BFI(	AC	62
517	BFI(	(AE	16
518	BFI(	SAE	67
519	BFI(	-	18
520	BFI(	SAE%	18
521	BFI(	IFC	53
522	BFI(	IFBE	20
523	BFI(	IF	21
524	BFI(	REL	68
525	BFI(	SAER	19
526	BFI(	BPR	68
527	BFI(	BC	68
528	BFI(	BV	68
529	BFI(	BFD	68
530	BFI(	(BE	16
531	BFI(	BF	68
532	BFI(	£	55
533	BFI(	SBE	69
534	BFI(	SBE$	57
535	BFI(	SBV	68
536	BFI(	IBV	68
537	BFI(	BFI	70
538	BFI(	BFI(	14
539	BFI(	L	61
540	BFI(	SWD	61
541	BFI(	SDE	61
542	BFI(	(DE	16
543	L:	IAV	156
544	L:	AAI	12
545	L:	AV	157
546	L:	SAV	156
547	L:	BAI	12
548	L:	AFI	157
549	L:	IFC	100
550	L:	IFBE	20
551	L:	IF	21
552	L:	BV	157
553	L:	SBV	158
554	L:	IBV	158
555	L:	BFI	157
556	L:	L:	6
557	L:	L	2
558	L:	PS	159
559	L:	PI	160
560	L:	PI(	14
561	L:	ALP	107
562	L:	AV:=	107
563	L:	BLP	108
564	L:	ALPL	15
565	L:	BLPL	21
566	L:	AS	159
567	L:	GOTOS	159
568	L:	GOTO	42
569	L:	FOR1	15
570	L:	FOR	109
571	L:	FOR2	110
572	L:	FOR2,	15
573	L:	FORC	6
574	L:	FORS	161
575	L:	UBS	159
576	L:	DUMMYS	159
577	L:	BS	162
578	L:	IFS	163
579	L:	CNS	111
580	L:	BLH	3
581	L:	BEGINB	4
582	L:	BLH.,	4
583	L:	UCMS	164
584	L:	BEGIN	6
585	L:	UBL	165
586	L:	CMS	166
587	L:	BL	167
588	L	:	168
589	SDE	ELSE	42
590	SDE	EDE	169
591	(DE	)	170
592	PI	(	171
593	PI(	AE	61
594	PI(	IAV	62
595	PI(	AAI	63
596	PI(	AV	62
597	PI(	SAV	62
598	PI(	ACP	61
599	PI(	BE	61
600	PI(	DE	61

PASS 2

STACK TABLE (ST)

NO.	TOS1	TOS	ITP	NO.	TOS1	TOS	ITP
601	PI(	STRING	61	651	ALPL	AAI	12
602	PI(	BAI	63	652	ALPL	AV	178
603	PI(	SWI	63	653	ALPL	SAV	177
604	PI(	ACPL	64	654	ALPL	AFD	179
605	PI(	ACPL,	14	655	ALPL	AFI	180
606	PI(	AFD	62	656	ALPL	AFI(	14
607	PI(	AFI	65	657	ALPL	(	15
608	PI(	AFI(	14	658	ALPL	APR	179
609	PI(	ACPL)	172	659	ALPL	AC	179
610	PI(	(	51	660	ALPL	(AE	16
611	PI(	APR	62	661	ALPL	SAE	181
612	PI(	AC	62	662	ALPL	-	18
613	PI(	(AE	16	663	ALPL	SAE%	18
614	PI(	SAE	67	664	ALPL	IFC	19
615	PI(	-	18	665	ALPL	IFBE	20
616	PI(	SAE%	18	666	ALPL	IF	21
617	PI(	IFC	53	667	ALPL	ALP	182
618	PI(	IFBE	20	668	ALPL	AV:=	183
619	PI(	IF	21	669	BLPL	IAV	134
620	PI(	REL	68	670	BLPL	AAI	12
621	PI(	SAER	19	671	BLPL	AV	134
622	PI(	BPR	68	672	BLPL	SAV	134
623	PI(	BC	68	673	BLPL	BE	176
624	PI(	BV	68	674	BLPL	BAI	12
625	PI(	BFD	68	675	BLPL	AFD	134
626	PI(	(BE	16	676	BLPL	AFI	136
627	PI(	BF	68	677	BLPL	AFI(	14
628	PI(	£	55	678	BLPL	(	21
629	PI(	SBE	69	679	BLPL	APR	134
630	PI(	SBE$	57	680	BLPL	AC	134
631	PI(	SBV	68	681	BLPL	(AE	16
632	PI(	IBV	68	682	BLPL	SAE	137
633	PI(	BFI	70	683	BLPL	-	18
634	PI(	BFI(	14	684	BLPL	SAE%	18
635	PI(	L	61	685	BLPL	IFC	57
636	PI(	SWD	61	686	BLPL	IFBE	20
637	PI(	SDE	61	687	BLPL	IF	21
638	PI(	(DE	16	688	BLPL	REL	184
639	:=	DE	173	689	BLPL	SAER	19
640	:=	SWI	12	690	BLPL	BPR	184
641	:=	(	42	691	BLPL	BC	184
642	:=	IFC	174	692	BLPL	BV	185
643	:=	IFBE	20	693	BLPL	BFD	184
644	:=	IF	21	694	BLPL	(BE	16
645	:=	L	175	695	BLPL	BF	184
646	:=	SWD	175	696	BLPL	£	55
647	:=	SDE	175	697	BLPL	SBE	186
648	:=	(DE	16	698	BLPL	SBE$	57
649	ALPL	AE	176	699	BLPL	SBV	187
650	ALPL	IAV	177	700	BLPL	IBV	187

PASS 2

STACK TABLE (ST)

NO.	TOS1	TOS	ITP
701	BLPL	BFI	188
702	BLPL	BFI(	14
703	BLPL	BLP	189
704	GOTO	DE	190
705	GOTO	SWI	12
706	GOTO	(	42
707	GOTO	IFC	174
708	GOTO	IFBE	20
709	GOTO	IF	21
710	GOTO	L	191
711	GOTO	SWD	191
712	GOTO	SDE	191
713	GOTO	(DE	16
714	FOR1	AE	192
715	FOR1	IAV	193
716	FOR1	AAI	12
717	FOR1	AV	193
718	FOR1	SAV	193
719	FOR1	AFD	193
720	FOR1	AFI	194
721	FOR1	AFI(	14
722	FOR1	(	15
723	FOR1	APR	193
724	FOR1	AC	193
725	FOR1	(AE	16
726	FOR1	SAE	195
727	FOR1	-	18
728	FOR1	SAE%	18
729	FOR1	IFC	19
730	FOR1	IFBE	20
731	FOR1	IF	21
732	FOR1	ESTEP	196
733	FOR1	AESTAE	197
734	FOR1	EWH	196
735	FOR	IAV	156
736	FOR	AAI	12
737	FOR	AV	157
738	FOR	SAV	156
739	FOR	AV:=	198
740	AESTAE	UNAE	199
741	AESTAE	UNTIL	15
742	STEP	AE	200
743	STEP	IAV	201
744	STEP	AAI	12
745	STEP	AV	201
746	STEP	SAV	201
747	STEP	AFD	201
748	STEP	AFI	202
749	STEP	AFI(	14
750	STEP	(	15
751	STEP	APR	201
752	STEP	AC	201
753	STEP	(AE	16
754	STEP	SAE	203
755	STEP	-	18
756	STEP	SAE%	18
757	STEP	IFC	19
758	STEP	IFBE	20
759	STEP	IF	21
760	UNTIL	AE	204
761	UNTIL	IAV	205
762	UNTIL	AAI	12
763	UNTIL	AV	205
764	UNTIL	SAV	205
765	UNTIL	AFD	205
766	UNTIL	AFI	206
767	UNTIL	AFI(	14
768	UNTIL	(	15
769	UNTIL	APR	205
770	UNTIL	AC	205
771	UNTIL	(AE	16
772	UNTIL	SAE	207
773	UNTIL	-	18
774	UNTIL	SAE%	18
775	UNTIL	IFC	19
776	UNTIL	IFBE	20
777	UNTIL	IF	21
778	WHILE	IAV	134
779	WHILE	AAI	12
780	WHILE	AV	134
781	WHILE	SAV	134
782	WHILE	BE	208
783	WHILE	BAI	12
784	WHILE	AFD	134
785	WHILE	AFI	136
786	WHILE	AFI(	14
787	WHILE	(	21
788	WHILE	APR	134
789	WHILE	AC	134
790	WHILE	(AE	16
791	WHILE	SAE	137
792	WHILE	-	18
793	WHILE	SAE%	18
794	WHILE	IFC	57
795	WHILE	IFBE	20
796	WHILE	IF	21
797	WHILE	REL	209
798	WHILE	SAER	19
799	WHILE	BPR	209
800	WHILE	BC	209

PASS 2

STACK TABLE (ST)

NO.	TOS1	TOS	ITP	NO.	TOS1	TOS	ITP
801	WHILE	BV	209	851	FORC	PI	130
802	WHILE	BFD	209	852	FORC	PI(	14
803	WHILE	(BE	16	853	FORC	ALP	107
804	WHILE	BF	209	854	FORC	AV:=	107
805	WHILE	£	55	855	FORC	BLP	108
806	WHILE	SBE	210	856	FORC	ALPL	15
807	WHILE	SBE$	57	857	FORC	BLPL	21
808	WHILE	SBV	209	858	FORC	AS	129
809	WHILE	IBV	209	859	FORC	GOTOS	129
810	WHILE	BFI	211	860	FORC	GOTO	42
811	WHILE	BFI(	14	861	FORC	FOR1	15
812	FOR2	,	212	862	FORC	FOR	109
813	FOR2	DO	213	863	FORC	FOR2	110
814	FOR2,	AE	192	864	FORC	FOR2,	15
815	FOR2,	IAV	193	865	FORC	FORC	6
816	FOR2,	AAI	12	866	FORC	FORS	129
817	FOR2,	AV	193	867	FORC	S	161
818	FOR2,	SAV	193	868	FORC	UBS	129
819	FOR2,	AFD	193	869	FORC	DUMMYS	129
820	FOR2,	AFI	194	870	FORC	BS	129
821	FOR2,	AFI(	14	871	FORC	US	129
822	FOR2,	(	15	872	FORC	PROGR	129
823	FOR2,	APR	193	873	FORC	IFS	132
824	FOR2,	AC	193	874	FORC	CNS	129
825	FOR2,	(AE	16	875	FORC	BLH	3
826	FOR2,	SAE	195	876	FORC	BEGINB	4
827	FOR2,	-	18	877	FORC	BLH.,	4
828	FOR2,	SAE%	18	878	FORC	UCMS	129
829	FOR2,	IFC	19	879	FORC	BEGIN	6
830	FOR2,	IFBE	20	880	FORC	UBL	129
831	FOR2,	IF	21	881	FORC	CMS	129
832	FOR2,	ESTEP	196	882	FORC	BL	129
833	FOR2,	AESTAE	197	883	S	END	214
834	FOR2,	EWH	196	884	S	.,	215
835	FORC	IAV	156	885	IFS	ELSE	6
836	FORC	AAI	12	886	IFS	ELSES	111
837	FORC	AV	157	887	S.,	IAV	156
838	FORC	SAV	156	888	S.,	AAI	12
839	FORC	BAI	12	889	S.,	AV	157
840	FORC	AFI	157	890	S.,	SAV	156
841	FORC	IFC	100	891	S.,	BAI	12
842	FORC	IFBE	20	892	S.,	AFI	157
843	FORC	IF	21	893	S.,	IFC	100
844	FORC	BV	157	894	S.,	IFBE	20
845	FORC	SBV	158	895	S.,	IF	21
846	FORC	IBV	158	896	S.,	BV	157
847	FORC	BFI	157	897	S.,	SBV	158
848	FORC	L:	6	898	S.,	IBV	158
849	FORC	L	2	899	S.,	BFI	157
850	FORC	PS	129	900	S.,	L:	6

PASS 2

STACK TABLE (ST)

NO.	TOS1	TOS	ITP
901	S.,	L	2
902	S.,	PS	129
903	S.,	PI	130
904	S.,	PI(	14
905	S.,	ALP	107
906	S.,	AV:=	107
907	S.,	BLP	108
908	S.,	ALPL	15
909	S.,	BLPL	21
910	S.,	AS	129
911	S.,	GOTOS	129
912	S.,	GOTO	42
913	S.,	FOR1	15
914	S.,	FOR	109
915	S.,	FOR2	110
916	S.,	FOR2,	15
917	S.,	FORC	6
918	S.,	FORS	129
919	S.,	S	216
920	S.,	UBS	129
921	S.,	DUMMYS	129
922	S.,	BS	129
923	S.,	US	129
924	S.,	PROGR	129
925	S.,	IFS	132
926	S.,	CNS	129
927	S.,	CMT	214
928	S.,	S.,	6
929	S.,	BLH	3
930	S.,	BEGINB	4
931	S.,	BLH.,	4
932	S.,	UCMS	129
933	S.,	BEGIN	6
934	S.,	UBL	129
935	S.,	CMS	129
936	S.,	BL	129
937	BLH	.,	217
938	BEGINB	IAV	156
939	BEGINB	AAI	12
940	BEGINB	AV	157
941	BEGINB	SAV	156
942	BEGINB	BAI	12
943	BEGINB	AFI	157
944	BEGINB	IFC	100
945	BEGINB	IFBE	20
946	BEGINB	IF	21
947	BEGINB	BV	157
948	BEGINB	SBV	158
949	BEGINB	IBV	158
950	BEGINB	BFI	157
951	BEGINB	L:	6
952	BEGINB	L	2
953	BEGINB	PS	129
954	BEGINB	PI	130
955	BEGINB	PI(	14
956	BEGINB	ALP	107
957	BEGINB	AV:=	107
958	BEGINB	BLP	108
959	BEGINB	ALPL	15
960	BEGINB	BLPL	21
961	BEGINB	AS	129
962	BEGINB	GOTOS	129
963	BEGINB	GOTO	42
964	BEGINB	FOR1	15
965	BEGINB	FOR	109
966	BEGINB	FOR2	110
967	BEGINB	FOR2,	15
968	BEGINB	FORC	6
969	BEGINB	FORS	129
970	BEGINB	S	216
971	BEGINB	UBS	129
972	BEGINB	DUMMYS	129
973	BEGINB	BS	129
974	BEGINB	US	129
975	BEGINB	PROGR	129
976	BEGINB	IFS	132
977	BEGINB	CNS	129
978	BEGINB	CMT	218
979	BEGINB	S.,	6
980	BEGINB	BLH	3
981	BEGINB	BEGINB	4
982	BEGINB	D	219
983	BEGINB	BLH.,	4
984	BEGINB	UCMS	129
985	BEGINB	BEGIN	6
986	BEGINB	UBL	129
987	BEGINB	CMS	129
988	BEGINB	BL	129
989	BEGINB	AD	220
990	BEGINB	ARRAY	221
991	BEGINB	SWID	220
992	BEGINB	SWITCH	222
993	BEGINB	RPH	3
994	BEGINB	PROCED	223
995	BEGINB	PD	220
996	BEGINB	RPH.,	6
997	BLH.,	IAV	156
998	BLH.,	AAI	12
999	BLH.,	AV	157
1000	BLH.,	SAV	156

PASS 2

STACK TABLE (ST)

NO.	TOS1	TOS	ITP
1001	BLH.,	BAI	12
1002	BLH.,	AFI	157
1003	BLH.,	IFC	100
1004	BLH.,	IFBE	20
1005	BLH.,	IF	21
1006	BLH.,	BV	157
1007	BLH.,	SBV	158
1008	BLH.,	IBV	158
1009	BLH.,	BFI	157
1010	BLH.,	L:	6
1011	BLH.,	L	2
1012	BLH.,	PS	129
1013	BLH.,	PI	130
1014	BLH.,	PI(	14
1015	BLH.,	ALP	107
1016	BLH.,	AV:=	107
1017	BLH.,	BLP	108
1018	BLH.,	ALPL	15
1019	BLH.,	BLPL	21
1020	BLH.,	AS	129
1021	BLH.,	GOTOS	129
1022	BLH.,	GOTO	42
1023	BLH.,	FOR1	15
1024	BLH.,	FOR	109
1025	BLH.,	FOR2	110
1026	BLH.,	FOR2,	15
1027	BLH.,	FORC	6
1028	BLH.,	FORS	129
1029	BLH.,	S	216
1030	BLH.,	UBS	129
1031	BLH.,	DUMMYS	129
1032	BLH.,	BS	129
1033	BLH.,	US	129
1034	BLH.,	PROGR	129
1035	BLH.,	IFS	132
1036	BLH.,	CNS	129
1037	BLH.,	CMT	218
1038	BLH.,	S.,	6
1039	BLH.,	BLH	3
1040	BLH.,	BEGINB	4
1041	BLH.,	D	219
1042	BLH.,	BLH.,	4
1043	BLH.,	UCMS	129
1044	BLH.,	BEGIN	6
1045	BLH.,	UBL	129
1046	BLH.,	CMS	129
1047	BLH.,	BL	129
1048	BLH.,	AD	220
1049	BLH.,	ARRAY	221
1050	BLH.,	SWID	220
1051	BLH.,	SWITCH	222
1052	BLH.,	RPH	3
1053	BLH.,	PROCED	223
1054	BLH.,	PD	220
1055	BLH.,	RPH.,	6
1056	BEGIN	IAV	156
1057	BEGIN	AAI	12
1058	BEGIN	AV	157
1059	BEGIN	SAV	156
1060	BEGIN	BAI	12
1061	BEGIN	AFI	157
1062	BEGIN	IFC	100
1063	BEGIN	IFBE	20
1064	BEGIN	IF	21
1065	BEGIN	BV	157
1066	BEGIN	SBV	158
1067	BEGIN	IBV	158
1068	BEGIN	BFI	157
1069	BEGIN	L:	6
1070	BEGIN	L	2
1071	BEGIN	PS	129
1072	BEGIN	PI	130
1073	BEGIN	PI(	14
1074	BEGIN	ALP	107
1075	BEGIN	AV:=	107
1076	BEGIN	BLP	108
1077	BEGIN	ALPL	15
1078	BEGIN	BLPL	21
1079	BEGIN	AS	129
1080	BEGIN	GOTOS	129
1081	BEGIN	GOTO	42
1082	BEGIN	FOR1	15
1083	BEGIN	FOR	109
1084	BEGIN	FOR2	110
1085	BEGIN	FOR2,	15
1086	BEGIN	FORC	6
1087	BEGIN	FORS	129
1088	BEGIN	S	216
1089	BEGIN	UBS	129
1090	BEGIN	DUMMYS	129
1091	BEGIN	BS	129
1092	BEGIN	US	129
1093	BEGIN	PROGR	129
1094	BEGIN	IFS	132
1095	BEGIN	CNS	129
1096	BEGIN	CMT	224
1097	BEGIN	S.,	6
1098	BEGIN	BLH	3
1099	BEGIN	BEGINB	4
1100	BEGIN	BLH.,	4

PASS 2

STACK TABLE (ST)

NO.	TOS1	TOS	ITP	NO.	TOS1	TOS	ITP
1101	BEGIN	UCMS	129	1151	BAI,	BAI,	234
1102	BEGIN	BEGIN	6	1152	AAL	,	235
1103	BEGIN	UBL	129	1153	AAL,	AAI	231
1104	BEGIN	CMS	129	1154	AAL,	AAS	236
1105	BEGIN	BL	129	1155	AAL,	AAI,	232
1106	AE:	AE.	225	1156	BAL	,	237
1107	AE:	IAV	11	1157	BAL,	BAI	231
1108	AE:	AAI	12	1158	BAL,	BAS	238
1109	AE:	AV	11	1159	BAL,	BAI,	234
1110	AE:	SAV	11	1160	ARRAY	AAI	231
1111	AE:	AFD	11	1161	ARRAY	BAI	231
1112	AE:	AFI	13	1162	ARRAY	AAS	239
1113	AE:	AFI(	14	1163	ARRAY	AAI,	232
1114	AE:	(	15	1164	ARRAY	BAS	240
1115	AE:	APR	11	1165	ARRAY	BAI,	234
1116	AE:	AC	11	1166	ARRAY	AAL	241
1117	AE:	(AE	16	1167	ARRAY	AAL,	232
1118	AE:	SAE	17	1168	ARRAY	BAL	241
1119	AE:	-	18	1169	ARRAY	BAL,	234
1120	AE:	SAE%	18	1170	SWL	,	242
1121	AE:	IFC	19	1171	SWL,	DE	173
1122	AE:	IFBE	20	1172	SWL,	SWI	12
1123	AE:	IF	21	1173	SWL,	(	42
1124	BPL	,	226	1174	SWL,	IFC	174
1125	BPL,	AE	2	1175	SWL,	IFBE	20
1126	BPL,	IAV	227	1176	SWL,	IF	21
1127	BPL,	AAI	12	1177	SWL,	L	175
1128	BPL,	AV	227	1178	SWL,	SWD	175
1129	BPL,	SAV	227	1179	SWL,	SDE	175
1130	BPL,	AFD	227	1180	SWL,	(DE	16
1131	BPL,	AFI	228	1181	SWITCH	SWI	157
1132	BPL,	AFI(	14	1182	SWITCH	SWISWL	243
1133	BPL,	(	15	1183	RPH	.,	244
1134	BPL,	APR	227	1184	PROCED	AFI	245
1135	BPL,	AC	227	1185	PROCED	BFI	245
1136	BPL,	(AE	16	1186	PROCED	PI	245
1137	BPL,	SAE	229	1187	RPH.,	IAV	156
1138	BPL,	-	18	1188	RPH.,	AAI	12
1139	BPL,	SAE%	18	1189	RPH.,	AV	157
1140	BPL,	IFC	19	1190	RPH.,	SAV	156
1141	BPL,	IFBE	20	1191	RPH.,	BAI	12
1142	BPL,	IF	21	1192	RPH.,	AFI	157
1143	BPL,	BP	230	1193	RPH.,	IFC	100
1144	BPL,	AE:	15	1194	RPH.,	IFBE	20
1145	AAI,	AAI	231	1195	RPH.,	IF	21
1146	AAI,	AAS	35	1196	RPH.,	BV	157
1147	AAI,	AAI,	232	1197	RPH.,	SBV	158
1148	(*BPL	*)	233	1198	RPH.,	IBV	158
1149	BAI,	BAI	231	1199	RPH.,	BFI	157
1150	BAI,	BAS	40	1200	RPH.,	L:	6

```
PASS 2

                         STACK TABLE (ST)

  NO.   TOS1      TOS  ITP          NO.   TOS1     TOS  ITP

 1201  RPH.,        L    2
 1202  RPH.,       PS  246
 1203  RPH.,       PI  247
 1204  RPH.,      PI(   14
 1205  RPH.,      ALP  107
 1206  RPH.,     AV:=  107
 1207  RPH.,      BLP  108
 1208  RPH.,     ALPL   15
 1209  RPH.,     BLPL   21
 1210  RPH.,       AS  246
 1211  RPH.,    GOTOS  246
 1212  RPH.,     GOTO   42
 1213  RPH.,     FOR1   15
 1214  RPH.,      FOR  109
 1215  RPH.,     FOR2  110
 1216  RPH.,    FOR2,   15
 1217  RPH.,     FORC    6
 1218  RPH.,     FORS  246
 1219  RPH.,        S  248
 1220  RPH.,      UBS  246
 1221  RPH.,  DUMMYS  246
 1222  RPH.,       BS  246
 1223  RPH.,       US  246
 1224  RPH.,    PROGR  246
 1225  RPH.,      IFS  249
 1226  RPH.,      CNS  246
 1227  RPH.,      BLH    3
 1228  RPH.,  BEGINB    4
 1229  RPH.,    BLH.,    4
 1230  RPH.,     UCMS  246
 1231  RPH.,    BEGIN    6
 1232  RPH.,      UBL  246
 1233  RPH.,      CMS  246
 1234  RPH.,       BL  246
```

```
PASS 2

     INTERMEDIATE TABLE (IT)

NO.       LB   UB       NO.       LB   UB

  1        1    3        51      256  269
  2        4    4        52      270  273
  3        5    5        53      274  286
  4        6   22        54      287  289
  5       23   23        55      290  299
  6       24   37        56      300  302
  7       38   44        57      303  313
  8       45   60        58      314  317
  9       61   65        59      318  332
 10       66   67        60      333  339
 11       68   70        61      340  341
 12       71   71        62      342  345
 13       72   75        63      346  348
 14       76   90        64      349  350
 15       91   97        65      351  355
 16       98   98        66      356  370
 17       99  101        67      371  374
 18      102  106        68      375  377
 19      107  112        69      378  380
 20      113  113        70      381  384
 21      114  125        71      385  385
 22      126  132        72      386  388
 23      133  133        73      389  389
 24      134  135        74      390  390
 25      136  138        75      391  394
 26      139  142        76      395  397
 27      143  147        77      398  399
 28      148  151        78      400  401
 29      152  153        79      402  404
 30      154  155        80      405  405
 31      156  157        81      406  420
 32      158  159        82      421  425
 33      160  160        83      426  436
 34      161  176        84      437  442
 35      177  178        85      443  457
 36      179  179        86      458  473
 37      180  186        87      474  488
 38      187  187        88      489  492
 39      188  196        89      493  496
 40      197  198        90      497  499
 41      199  203        91      500  504
 42      204  207        92      505  507
 43      208  209        93      508  518
 44      210  224        94      519  520
 45      225  239        95      521  523
 46      240  241        96      524  525
 47      242  245        97      526  532
 48      246  248        98      533  535
 49      249  250        99      536  539
 50      251  255       100      540  552
```

```
PASS 2

       INTERMEDIATE TABLE (IT)

 NO.       LB   UB      NO.      LB   UB

 101      553  554      151     898  905
 102      555  555      152     906  913
 103      556  556      153     914  922
 104      557  561      154     923  937
 105      562  564      155     938  945
 106      565  568      156     946  946
 107      569  575      157     947  947
 108      576  587      158     948  948
 109      588  589      159     949  951
 110      590  591      160     952  955
 111      592  593      161     956  957
 112      594  596      162     958  960
 113      597  607      163     961  963
 114      608  621      164     964  967
 115      622  635      165     968  971
 116      636  642      166     972  975
 117      643  647      167     976  979
 118      648  660      168     980  993
 119      661  675      169     994  998
 120      676  688      170     999 1003
 121      689  707      171    1004 1018
 122      708  715      172    1019 1021
 123      716  724      173    1022 1023
 124      725  732      174    1024 1026
 125      733  741      175    1027 1028
 126      742  751      176    1029 1031
 127      752  757      177    1032 1036
 128      758  762      178    1037 1041
 129      763  764      179    1042 1045
 130      765  767      180    1046 1051
 131      768  769      181    1052 1055
 132      770  772      182    1056 1062
 133      773  791      183    1063 1069
 134      792  793      184    1070 1073
 135      794  794      185    1074 1078
 136      795  797      186    1079 1082
 137      798  799      187    1083 1087
 138      800  801      188    1088 1093
 139      802  803      189    1094 1105
 140      804  806      190    1108 1108
 141      807  815      191    1109 1111
 142      816  825      192    1112 1115
 143      826  834      193    1116 1120
 144      835  846      194    1121 1126
 145      847  854      195    1127 1131
 146      855  862      196    1132 1133
 147      863  870      197    1134 1134
 148      871  879      198    1135 1141
 149      880  890      199    1142 1143
 150      891  897      200    1144 1144
```

```
PASS 2

     INTERMEDIATE TABLE (IT)

 NO.      LB   UB        NO.      LB   UB

 201     1145 1146
 202     1147 1149
 203     1150 1151
 204     1152 1153
 205     1154 1156
 206     1157 1160
 207     1161 1163
 208     1164 1165
 209     1166 1168
 210     1169 1171
 211     1172 1175
 212     1176 1182
 213     1183 1196
 214     1197 1200
 215     1201 1214
 216     1215 1216
 217     1217 1233
 218     1234 1237
 219     1238 1238
 220     1239 1239
 221     1240 1241
 222     1242 1242
 223     1243 1245
 224     1246 1249
 225     1250 1251
 226     1252 1258
 227     1259 1260
 228     1261 1263
 229     1264 1265
 230     1266 1267
 231     1268 1269
 232     1270 1270
 233     1271 1272
 234     1273 1273
 235     1274 1274
 236     1275 1276
 237     1277 1277
 238     1278 1279
 239     1280 1281
 240     1282 1283
 241     1284 1285
 242     1286 1289
 243     1290 1290
 244     1291 1304
 245     1305 1305
 246     1306 1306
 247     1307 1308
 248     1309 1309
 249     1310 1311
```

PASS 2

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDU-CTION	*	NO.	LAST SYMBOL	STATE	REDU-CTION
	1	L	3		*	51	R	2	SL*)
	2	BEGINB	3		*	52	$	2	SL*)
	3	BEGIN	3		*	53	:	2	SL*)
*	4	:	3		*	54	:=	2	SL*)
	5	.,	3		*	55	STEP	2	SL*)
*	6	AAI	3		*	56	UNTIL	2	SL*)
*	7	SAV	3		*	57	WHILE	2	SL*)
*	8	BAI	3		*	58	DO	2	SL*)
*	9	AFI	3		*	59	END	2	SL*)
*	10	IF	3		*	60	.,	2	SL*)
*	11	SBV	3			61	,	2	(*AE*)
*	12	BFI	3			62	)	2	(*AE*)
*	13	L	3			63	ELSE	2	(*AE*)
*	14	PI	3			64	END	2	(*AE*)
*	15	GOTO	3			65	.,	2	(*AE*)
*	16	FOR	3		*	66	,	2	SL
*	17	DUMMYS	3		*	67	*)	2	SL
*	18	BEGINB	3			68	,	1	AE
*	19	BEGIN	3			69	*)	1	AE
*	20	ARRAY	3			70	%	1	SAE
*	21	SWITCH	3		*	71	(*	3	
*	22	PROCED	3			72	,	1	AE
	23		1	PROGR		73	*)	1	AE
*	24	AAI	3			74	(	3	
*	25	SAV	3			75	%	1	SAE
*	26	BAI	3		*	76	AAI	3	
*	27	AFI	3		*	77	SAV	3	
*	28	IF	3		*	78	STRING	3	
*	29	SBV	3		*	79	BAI	3	
*	30	BFI	3		*	80	SWI	3	
*	31	L	3		*	81	AFI	3	
*	32	PI	3		*	82	(	3	
*	33	GOTO	3		*	83	AC	3	
*	34	FOR	3		*	84	-	3	
*	35	DUMMYS	3		*	85	IF	3	
*	36	BEGINB	3		*	86	BC	3	
*	37	BEGIN	3		*	87	£	3	
	38	AAI	2	SL,	*	88	SBV	3	
	39	SAV	2	SL,	*	89	BFI	3	
	40	AFI	2	SL,	*	90	L	3	
	41	(	2	SL,		91	AAI	3	
	42	AC	2	SL,		92	SAV	3	
	43	-	2	SL,		93	AFI	3	
	44	IF	2	SL,		94	(	3	
*	45	,	2	SL*)		95	AC	3	
*	46	*)	2	SL*)		96	-	3	
*	47	)	2	SL*)		97	IF	3	
*	48	%	2	SL*)	*	98	)	3	
*	49	ELSE	2	SL*)		99	,	1	AE
*	50	THEN	2	SL*)		100	*)	1	AE

PASS 2

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDU-CTION	*	NO.	LAST SYMBOL	STATE	REDU-CTION
	101	%	3		*	151	:	1	AE
*	102	AAI	3			152	,	1	BPL
*	103	SAV	3			153	*)	1	BPL
*	104	AFI	3		*	154	,	3	
*	105	(	3		*	155	*)	2	(*BPL
*	106	AC	3			156	,	3	
	107	AAI	3			157	*)	3	
	108	SAV	3		*	158	,	1	SL
	109	AFI	3		*	159	*)	1	SL
	110	(	3			160	AAI	2	AAI,
	111	AC	3		*	161	,	2	IAV
	112	-	3		*	162	*)	2	IAV
*	113	THEN	3		*	163	)	2	IAV
	114	AAI	3		*	164	%	2	IAV
	115	SAV	3		*	165	ELSE	2	IAV
	116	BAI	3		*	166	THEN	2	IAV
	117	AFI	3		*	167	R	2	IAV
	118	(	3		*	168	$	2	IAV
	119	AC	3		*	169	:	2	IAV
	120	-	3		*	170	:=	2	IAV
	121	IF	3		*	171	STEP	2	IAV
	122	BC	3		*	172	UNTIL	2	IAV
	123	£	3		*	173	WHILE	2	IAV
	124	SBV	3		*	174	DO	2	IAV
	125	BFI	3		*	175	END	2	IAV
*	126	AAI	2	AE:	*	176	.,	2	IAV
*	127	SAV	2	AE:		177	,	2	AAS
*	128	AFI	2	AE:		178	.,	2	AAS
*	129	(	2	AE:	*	179	*)	3	
*	130	AC	2	AE:		180	AAI	2	AV:=
*	131	-	2	AE:		181	SAV	2	AV:=
*	132	IF	2	AE:		182	AFI	2	AV:=
	133	UNTIL	2	AESTAE		183	(	2	AV:=
*	134	,	2	EWH		184	AC	2	AV:=
*	135	DO	2	EWH		185	-	2	AV:=
	136	,	2	(*AE		186	IF	2	AV:=
	137	*)	2	(*AE	*	187	BAI	2	BAI,
	138	:	3			188	,	2	IBV
*	139	,	1	AE		189	)	2	IBV
*	140	*)	1	AE		190	ELSE	2	IBV
*	141	%	1	SAE		191	THEN	2	IBV
*	142	:	1	AE		192	$	2	IBV
	143	,	1	AE		193	:=	2	IBV
	144	*)	1	AE		194	DO	2	IBV
	145	(	3			195	END	2	IBV
	146	%	1	SAE		196	.,	2	IBV
	147	:	1	AE	*	197	,	2	BAS
*	148	,	1	AE	*	198	.,	2	BAS
*	149	*)	1	AE		199	,	2	SWD
*	150	%	3			200	)	2	SWD

PASS 2

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDU-CTION	*	NO.	LAST SYMBOL	STATE	REDU-CTION
	201	ELSE	2	SWD	*	251	,	1	ACP
	202	END	2	SWD	*	252	)	1	ACP
	203	•,	2	SWD	*	253	(	3	
*	204	SWI	3		*	254	%	1	SAE
*	205	(	3		*	255	R	1	SAE
*	206	IF	3			256	AAI	3	
*	207	L	3			257	SAV	3	
	208	,	3			258	BAI	3	
	209	•,	2	SWISWL		259	SWI	3	
*	210	AAI	2	ACPL,		260	AFI	3	
*	211	SAV	2	ACPL,		261	(	3	
*	212	STRING	2	ACPL,		262	AC	3	
*	213	BAI	2	ACPL,		263	-	3	
*	214	SWI	2	ACPL,		264	IF	3	
*	215	AFI	2	ACPL,		265	BC	3	
*	216	(	2	ACPL,		266	£	3	
*	217	AC	2	ACPL,		267	SBV	3	
*	218	-	2	ACPL,		268	BFI	3	
*	219	IF	2	ACPL,		269	L	3	
*	220	BC	2	ACPL,	*	270	,	1	ACP
*	221	£	2	ACPL,	*	271	)	1	ACP
*	222	SBV	2	ACPL,	*	272	%	3	
*	223	BFI	2	ACPL,	*	273	R	3	
*	224	L	2	ACPL,		274	AAI	3	
	225	,	2	ACPL)		275	SAV	3	
	226	*)	2	ACPL)		276	BAI	3	
	227	)	2	ACPL)		277	SWI	3	
	228	%	2	ACPL)		278	AFI	3	
	229	ELSE	2	ACPL)		279	(	3	
	230	THEN	2	ACPL)		280	AC	3	
	231	R	2	ACPL)		281	-	3	
	232	$	2	ACPL)		282	BC	3	
	233	:	2	ACPL)		283	£	3	
	234	STEP	2	ACPL)		284	SBV	3	
	235	UNTIL	2	ACPL)		285	BFI	3	
	236	WHILE	2	ACPL)		286	L	3	
	237	DO	2	ACPL)	*	287	,	1	ACP
	238	END	2	ACPL)	*	288	)	1	ACP
	239	•,	2	ACPL)	*	289	$	1	SBE
*	240	,	1	ACP		290	AAI	3	
*	241	)	1	ACP		291	SAV	3	
	242	,	1	ACP		292	BAI	3	
	243	)	1	ACP		293	AFI	3	
	244	%	1	SAE		294	(	3	
	245	R	1	SAE		295	AC	3	
*	246	(*	3			296	-	3	
*	247	,	1	ACP		297	BC	3	
*	248	)	1	ACP		298	SBV	3	
	249	,	2	ACPL		299	BFI	3	
	250	)	2	ACPL	*	300	,	1	ACP

PASS 2

FINAL TABLE (FT)

*	NO.	LAST	STATE	REDU-	*	NO.	LAST	STATE	REDU-
		SYMBOL		CTION			SYMBOL		CTION
*	301	)	1	ACP		351	,	1	ACPL
*	302	$	3			352	)	1	ACPL
	303	AAI	3			353	(	3	
	304	SAV	3			354	%	1	SAE
	305	BAI	3			355	R	1	SAE
	306	AFI	3		*	356	,	2	AFD
	307	(	3		*	357	*)	2	AFD
	308	AC	3		*	358	)	2	AFD
	309	-	3		*	359	%	2	AFD
	310	BC	3		*	360	ELSE	2	AFD
	311	£	3		*	361	THEN	2	AFD
	312	SBV	3		*	362	R	2	AFD
	313	BFI	3		*	363	$	2	AFD
*	314	,	1	ACP	*	364	:	2	AFD
*	315	)	1	ACP	*	365	STEP	2	AFD
*	316	(	3		*	366	UNTIL	2	AFD
*	317	$	1	SBE	*	367	WHILE	2	AFD
	318	AAI	2	AFI(	*	368	DO	2	AFD
	319	SAV	2	AFI(	*	369	END	2	AFD
	320	STRING	2	AFI(	*	370	.,	2	AFD
	321	BAI	2	AFI(		371	,	1	ACPL
	322	SWI	2	AFI(		372	)	1	ACPL
	323	AFI	2	AFI(		373	%	3	
	324	(	2	AFI(		374	R	3	
	325	AC	2	AFI(	*	375	,	1	ACPL
	326	-	2	AFI(	*	376	)	1	ACPL
	327	IF	2	AFI(	*	377	$	1	SBE
	328	BC	2	AFI(		378	,	1	ACPL
	329	£	2	AFI(		379	)	1	ACPL
	330	SBV	2	AFI(		380	$	3	
	331	BFI	2	AFI(	*	381	,	1	ACPL
	332	L	2	AFI(	*	382	)	1	ACPL
*	333	AAI	2	ALP	*	383	(	3	
*	334	SAV	2	ALP	*	384	$	1	SBE
*	335	AFI	2	ALP		385	)	2	(AE
*	336	(	2	ALP	*	386	)	1	AE
*	337	AC	2	ALP	*	387	%	1	SAE
*	338	-	2	ALP	*	388	R	1	SAE
*	339	IF	2	ALP		389	)	2	(BE
	340	,	1	ACPL	*	390	)	2	(DE
	341	)	1	ACPL		391	)	1	AE
*	342	,	1	ACPL		392	(	3	
*	343	)	1	ACPL		393	%	1	SAE
*	344	%	1	SAE		394	R	1	SAE
*	345	R	1	SAE	*	395	)	1	AE
	346	(*	3		*	396	%	3	
	347	,	1	ACPL	*	397	R	3	
	348	)	1	ACPL		398	)	1	BE
*	349	,	3			399	$	1	SBE
*	350	)	3		*	400	)	1	BE

PASS 2

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDU-CTION	*	NO.	LAST SYMBOL	STATE	REDU-CTION
*	401	$	3			451	:	1	APR
	402	)	1	BE		452	STEP	1	APR
	403	(	3			453	UNTIL	1	APR
	404	$	1	SBE		454	WHILE	1	APR
*	405	)	1	DE		455	DO	1	APR
	406	,	2	APR		456	END	1	APR
	407	*)	2	APR		457	.,	1	APR
	408	)	2	APR	*	458	,	1	APR
	409	%	2	APR	*	459	*)	1	APR
	410	ELSE	2	APR	*	460	)	1	APR
	411	THEN	2	APR	*	461	(	3	
	412	R	2	APR	*	462	%	1	APR
	413	$	2	APR	*	463	ELSE	1	APR
	414	:	2	APR	*	464	THEN	1	APR
	415	STEP	2	APR	*	465	R	1	APR
	416	UNTIL	2	APR	*	466	$	1	APR
	417	WHILE	2	APR	*	467	:	1	APR
	418	DO	2	APR	*	468	STEP	1	APR
	419	END	2	APR	*	469	UNTIL	1	APR
	420	.,	2	APR	*	470	WHILE	1	APR
*	421	AAI	2	SAE%	*	471	DO	1	APR
*	422	SAV	2	SAE%	*	472	END	1	APR
*	423	AFI	2	SAE%	*	473	.,	1	APR
*	424	(	2	SAE%		474	,	2	SAE
*	425	AC	2	SAE%		475	*)	2	SAE
	426	,	2	SAEEAE		476	)	2	SAE
	427	*)	2	SAEEAE		477	%	2	SAE
	428	)	2	SAEEAE		478	ELSE	2	SAE
	429	ELSE	2	SAEEAE		479	THEN	2	SAE
	430	:	2	SAEEAE		480	R	2	SAE
	431	STEP	2	SAEEAE		481	$	2	SAE
	432	UNTIL	2	SAEEAE		482	:	2	SAE
	433	WHILE	2	SAEEAE		483	STEP	2	SAE
	434	DO	2	SAEEAE		484	UNTIL	2	SAE
	435	END	2	SAEEAE		485	WHILE	2	SAE
	436	.,	2	SAEEAE		486	DO	2	SAE
*	437	AAI	2	SAER		487	END	2	SAE
*	438	SAV	2	SAER		488	.,	2	SAE
*	439	AFI	2	SAER	*	489	%	1	SAE
*	440	(	2	SAER	*	490	ELSE	1	SAE
*	441	AC	2	SAER	*	491	R	1	SAE
*	442	-	2	SAER	*	492	:=	1	AV
	443	,	1	APR		493	%	1	SAE
	444	*)	1	APR		494	ELSE	1	SAE
	445	)	1	APR		495	R	1	SAE
	446	%	1	APR		496	:=	3	
	447	ELSE	1	APR	*	497	%	1	SAE
	448	THEN	1	APR	*	498	ELSE	1	SAE
	449	R	1	APR	*	499	R	1	SAE
	450	$	1	APR		500	(	3	

PASS 2

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDU-CTION	*	NO.	LAST SYMBOL	STATE	REDU-CTION
	501	%	1	SAE	*	551	BEGINB	3	
	502	ELSE	1	SAE	*	552	BEGIN	3	
	503	R	1	SAE		553	ELSE	1	SDE
	504	:=	3			554	:	3	
*	505	%	3		*	555	ELSE	1	SDE
*	506	ELSE	3			556	ELSE	3	
*	507	R	3		*	557	,	2	DE
	508	,	2	AE	*	558	)	2	DE
	509	*)	2	AE	*	559	ELSE	2	DE
	510	)	2	AE	*	560	END	2	DE
	511	ELSE	2	AE	*	561	.,	2	DE
	512	:	2	AE		562	ELSE	1	US
	513	STEP	2	AE		563	END	1	US
	514	UNTIL	2	AE		564	.,	1	US
	515	WHILE	2	AE	*	565	(	3	
	516	DO	2	AE	*	566	ELSE	1	US
	517	END	2	AE	*	567	END	1	US
	518	.,	2	AE	*	568	.,	1	US
*	519	ELSE	1	SBE		569	AAI	1	ALPL
*	520	$	1	SBE		570	SAV	1	ALPL
	521	ELSE	1	SBE		571	AFI	1	ALPL
	522	$	1	SBE		572	(	1	ALPL
	523	:=	3			573	AC	1	ALPL
*	524	ELSE	3			574	-	1	ALPL
*	525	$	3			575	IF	1	ALPL
	526	,	2	BE	*	576	AAI	1	BLPL
	527	)	2	BE	*	577	SAV	1	BLPL
	528	ELSE	2	BE	*	578	BAI	1	BLPL
	529	THEN	2	BE	*	579	AFI	1	BLPL
	530	DO	2	BE	*	580	(	1	BLPL
	531	END	2	BE	*	581	AC	1	BLPL
	532	.,	2	BE	*	582	-	1	BLPL
*	533	ELSE	1	SBE	*	583	IF	1	BLPL
*	534	$	1	SBE	*	584	BC	1	BLPL
*	535	:=	1	BV	*	585	£	1	BLPL
	536	(	3		*	586	SBV	1	BLPL
	537	ELSE	1	SBE	*	587	BFI	1	BLPL
	538	$	1	SBE		588	AAI	3	
	539	:=	3			589	SAV	3	
*	540	AAI	3		*	590	,	3	
*	541	SAV	3		*	591	DO	3	
*	542	BAI	3			592	END	2	CNS
*	543	AFI	3			593	.,	2	CNS
*	544	SBV	3		*	594	ELSE	2	IFS
*	545	BFI	3		*	595	END	2	IFS
*	546	L	3		*	596	.,	2	IFS
*	547	PI	3			597	,	2	EAE
*	548	GOTO	3			598	*)	2	EAE
*	549	FOR	3			599	)	2	EAE
*	550	DUMMYS	3			600	ELSE	2	EAE

PASS 2

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDU-CTION	*	NO.	LAST SYMBOL	STATE	REDU-CTION
	601	:	2	• EAE	*	651	%	1	SAE
	602	STEP	2	EAE	*	652	ELSE	1	AE
	603	UNTIL	2	EAE	*	653	R	1	SAE
	604	WHILE	2	EAE	*	654	:	1	AE
	605	DO	2	EAE	*	655	STEP	1	AE
	606	END	2	EAE	*	656	UNTIL	1	AE
	607	.,	2	EAE	*	657	WHILE	1	AE
*	608	,	1	AE	*	658	DO	1	AE
*	609	*)	1	AE	*	659	END	1	AE
*	610	)	1	AE	*	660	.,	1	AE
*	611	%	1	SAE		661	,	1	AE
*	612	ELSE	1	AE		662	*)	1	AE
*	613	R	1	SAE		663	)	1	AE
*	614	:	1	AE		664	(	3	
*	615	:=	1	AV		665	%	1	SAE
*	616	STEP	1	AE		666	ELSE	1	AE
*	617	UNTIL	1	AE		667	R	1	SAE
*	618	WHILE	1	AE		668	:	1	AE
*	619	DO	1	AE		669	:=	3	
*	620	END	1	AE		670	STEP	1	AE
*	621	.,	1	AE		671	UNTIL	1	AE
	622	,	1	AE		672	WHILE	1	AE
	623	*)	1	AE		673	DO	1	AE
	624	)	1	AE		674	END	1	AE
	625	%	1	SAE		675	.,	1	AE
	626	ELSE	1	AE	*	676	,	1	AE
	627	R	1	SAE	*	677	*)	1	AE
	628	:	1	AE	*	678	)	1	AE
	629	:=	3		*	679	%	3	
	630	STEP	1	AE	*	680	ELSE	1	AE
	631	UNTIL	1	AE	*	681	R	3	
	632	WHILE	1	AE	*	682	:	1	AE
	633	DO	1	AE	*	683	STEP	1	AE
	634	END	1	AE	*	684	UNTIL	1	AE
	635	.,	1	AE	*	685	WHILE	1	AE
*	636	,	2	EBE	*	686	DO	1	AE
*	637	)	2	EBE	*	687	END	1	AE
*	638	ELSE	2	EBE	*	688	.,	1	AE
*	639	THEN	2	EBE		689	AAI	3	
*	640	DO	2	EBE		690	SAV	3	
*	641	END	2	EBE		691	BAI	3	
*	642	.,	2	EBE		692	SWI	3	
	643	,	2	EDE		693	AFI	3	
	644	)	2	EDE		694	(	3	
	645	ELSE	2	EDE		695	AC	3	
	646	END	2	EDE		696	-	3	
	647	.,	2	EDE		697	BC	3	
*	648	,	1	AE		698	£	3	
*	649	*)	1	AE		699	SBV	3	
*	650	)	1	AE		700	BFI	3	

PASS 2

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDU- OTION	*	NO.	LAST SYMBOL	STATE	REDU- CTION
	701	L	3		*	751	.,	1	BE
	702	PI	3			752	,	1	DE
	703	GOTO	3			753	)	1	DE
	704	FOR	3			754	ELSE	1	DE
	705	DUMMYS	3			755	:	3	
	706	BEGINB	3			756	END	1	DE
	707	BEGIN	3			757	.,	1	DE
*	708	,	1	BE	*	758	,	1	DE
*	709	)	1	BE	*	759	)	1	DE
*	710	ELSE	1	BE	*	760	ELSE	1	DE
*	711	THEN	1	BE	*	761	END	1	DE
*	712	$	1	SBE	*	762	.,	1	DE
*	713	DO	1	BE		763	END	1	S
*	714	END	1	BE		764	.,	1	S
*	715	.,	1	BE	*	765	(	3	
	716	,	1	BE	*	766	END	1	S
	717	)	1	BE	*	767	.,	1	S
	718	ELSE	1	BE		768	END	2	ELSES
	719	THEN	1	BE		769	.,	2	ELSES
	720	$	1	SBE	*	770	ELSE	3	
	721	:=	3		*	771	END	1	S
	722	DO	1	BE	*	772	.,	1	S
	723	END	1	BE		773	AAI	2	IFC
	724	.,	1	BE		774	SAV	2	IFC
*	725	,	1	BE		775	BAI	2	IFC
*	726	)	1	BE		776	SWI	2	IFC
*	727	ELSE	1	BE		777	AFI	2	IFC
*	728	THEN	1	BE		778	(	2	IFC
*	729	$	3			779	AC	2	IFC
*	730	DO	1	BE		780	-	2	IFC
*	731	END	1	BE		781	BC	2	IFC
*	732	.,	1	BE		782	£	2	IFC
	733	,	1	BE		783	SBV	2	IFC
	734	)	1	BE		784	BFI	2	IFC
	735	ELSE	1	BE		785	L	2	IFC
	736	THEN	1	BE		786	PI	2	IFC
	737	$	1	SBE		787	GOTO	2	IFC
	738	:=	1	BV		788	FOR	2	IFC
	739	DO	1	BE		789	DUMMYS	2	IFC
	740	END	1	BE		790	BEGINB	2	IFC
	741	.,	1	BE		791	BEGIN	2	IFC
*	742	,	1	BE	*	792	%	1	SAE
*	743	)	1	BE	*	793	R	1	SAE
*	744	(	3			794	THEN	2	IFBE
*	745	ELSE	1	BE	*	795	(	3	
*	746	THEN	1	BE	*	796	%	1	SAE
*	747	$	1	SBE	*	797	R	1	SAE
*	748	:=	3			798	%	3	
*	749	DO	1	BE		799	R	3	
*	750	END	1	BE	*	800	THEN	1	BE

PASS 2

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDU-CTION	*	NO.	LAST SYMBOL	STATE	REDU-CTION
*	801	$	1	SBE		851	$	2	BPR
	802	THEN	1	BE		852	DO	2	BPR
	803	$	3			853	END	2	BPR
*	804	(	3			854	•,	2	BPR
*	805	THEN	1	BE	*	855	,	1	BPR
*	806	$	1	SBE	*	856	)	1	BPR
	807	,	1	SAE	*	857	ELSE	1	BPR
	808	)	1	SAE	*	858	THEN	1	BPR
	809	%	1	SAE	*	859	$	1	BPR
	810	ELSE	1	SAE	*	860	DO	1	BPR
	811	THEN	1	SAE	*	861	END	1	BPR
	812	$	1	SAE	*	862	•,	1	BPR
	813	DO	1	SAE		863	,	2	BF
	814	END	1	SAE		864	)	2	BF
	815	•,	1	SAE		865	ELSE	2	BF
*	816	,	1	SAE		866	THEN	2	BF
*	817	)	1	SAE		867	$	2	BF
*	818	(	3			868	DO	2	BF
*	819	%	1	SAE		869	END	2	BF
*	820	ELSE	1	SAE		870	•,	2	BF
*	821	THEN	1	SAE	*	871	,	1	BPR
*	822	$	1	SAE	*	872	)	1	BPR
*	823	DO	1	SAE	*	873	(	3	
*	824	END	1	SAE	*	874	ELSE	1	BPR
*	825	•,	1	SAE	*	875	THEN	1	BPR
	826	,	2	REL	*	876	$	1	BPR
	827	)	2	REL	*	877	DO	1	BPR
	828	%	3		*	878	END	1	BPR
	829	ELSE	2	REL	*	879	•,	1	BPR
	830	THEN	2	REL		880	AAI	2	SBE$
	831	$	2	REL		881	SAV	2	SBE$
	832	DO	2	REL		882	BAI	2	SBE$
	833	END	2	REL		883	AFI	2	SBE$
	834	•,	2	REL		884	(	2	SBE$
*	835	AAI	2	BLP		885	AC	2	SBE$
*	836	SAV	2	BLP		886	-	2	SBE$
*	837	BAI	2	BLP		887	BC	2	SBE$
*	838	AFI	2	BLP		888	£	2	SBE$
*	839	(	2	BLP		889	SBV	2	SBE$
*	840	AC	2	BLP		890	BFI	2	SBE$
*	841	-	2	BLP	*	891	,	2	SBEEBE
*	842	IF	2	BLP	*	892	)	2	SBEEBE
*	843	BC	2	BLP	*	893	ELSE	2	SBEEBE
*	844	£	2	BLP	*	894	THEN	2	SBEEBE
*	845	SBV	2	BLP	*	895	DO	2	SBEEBE
*	846	BFI	2	BLP	*	896	END	2	SBEEBE
	847	,	2	BPR	*	897	•,	2	SBEEBE
	848	)	2	BPR		898	,	1	BF
	849	ELSE	2	BPR		899	)	1	BF
	850	THEN	2	BPR		900	ELSE	1	BF

PASS 2

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDU-CTION	*	NO.	LAST SYMBOL	STATE	REDU-CTION
	901	THEN	1	BF		951	.,	1	BS
	902	$	1	BF	*	952	(	3	
	903	DO	1	BF	*	953	ELSE	1	BS
	904	END	1	BF	*	954	END	1	BS
	905	.,	1	BF	*	955	.,	1	BS
*	906	,	2	SBE		956	END	2	FORS
*	907	)	2	SBE		957	.,	2	FORS
*	908	ELSE	2	SBE	*	958	ELSE	2	BS
*	909	THEN	2	SBE	*	959	END	2	BS
*	910	$	2	SBE	*	960	.,	2	BS
*	911	DO	2	SBE		961	ELSE	3	
*	912	END	2	SBE		962	END	1	CNS
*	913	.,	2	SBE		963	.,	1	CNS
	914	,	1	BF	*	964		1	CMS
	915	)	1	BF	*	965	ELSE	1	CMS
	916	(	3		*	966	END	1	CMS
	917	ELSE	1	BF	*	967	.,	1	CMS
	918	THEN	1	BF		968		1	BL
	919	$	1	BF		969	ELSE	1	BL
	920	DO	1	BF		970	END	1	BL
	921	END	1	BF		971	.,	1	BL
	922	.,	1	BF	*	972	.	2	CMS
*	923	AAI	2	BFI(	*	973	ELSE	2	CMS
*	924	SAV	2	BFI(	*	974	END	2	CMS
*	925	STRING	2	BFI(	*	975	.,	2	CMS
*	926	BAI	2	BFI(		976		2	BL
*	927	SWI	2	BFI(		977	ELSE	2	BL
*	928	AFI	2	BFI(		978	END	2	BL
*	929	(	2	BFI(		979	.,	2	BL
*	930	AC	2	BFI(	*	980	AAI	2	L:
*	931	-	2	BFI(	*	981	SAV	2	L:
*	932	IF	2	BFI(	*	982	BAI	2	L:
*	933	BC	2	BFI(	*	983	AFI	2	L:
*	934	£	2	BFI(	*	984	IF	2	L:
*	935	SBV	2	BFI(	*	985	SBV	2	L:
*	936	BFI	2	BFI(	*	986	BFI	2	L:
*	937	L	2	BFI(	*	987	L	2	L:
	938	,	2	BFD	*	988	PI	2	L:
	939	)	2	BFD	*	989	GOTO	2	L:
	940	ELSE	2	BFD	*	990	FOR	2	L:
	941	THEN	2	BFD	*	991	DUMMYS	2	L:
	942	$	2	BFD	*	992	BEGINB	2	L:
	943	DO	2	BFD	*	993	BEGIN	2	L:
	944	END	2	BFD		994	,	2	SDEEDE
	945	.,	2	BFD		995	)	2	SDEEDE
*	946	:=	1	AV		996	ELSE	2	SDEEDE
	947	:=	3			997	END	2	SDEEDE
*	948	:=	1	BV		998	.,	2	SDEEDE
	949	ELSE	1	BS	*	999	,	2	SDE
	950	END	1	BS	*	1000	)	2	SDE

PASS 2

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDU-CTION	*	NO.	LAST SYMBOL	STATE	REDU-CTION
*	1001	ELSE	2	SDE	*	1051	.,	1	AE
*	1002	END	2	SDE		1052	%	3	
*	1003	.,	2	SDE		1053	ELSE	1	AE
	1004	AAI	2	PI(		1054	END	1	AE
	1005	SAV	2	PI(		1055	.,	1	AE
	1006	STRING	2	PI(	*	1056	AAI	2	ALPL
	1007	BAI	2	PI(	*	1057	SAV	2	ALPL
	1008	SWI	2	PI(	*	1058	AFI	2	ALPL
	1009	AFI	2	PI(	*	1059	(	2	ALPL
	1010	(	2	PI(	*	1060	AC	2	ALPL
	1011	AC	2	PI(	*	1061	-	2	ALPL
	1012	-	2	PI(	*	1062	IF	2	ALPL
	1013	IF	2	PI(		1063	AAI	1	ALP
	1014	BC	2	PI(		1064	SAV	1	ALP
	1015	£	2	PI(		1065	AFI	1	ALP
	1016	SBV	2	PI(		1066	(	1	ALP
	1017	BFI	2	PI(		1067	AC	1	ALP
	1018	L	2	PI(		1068	-	1	ALP
*	1019	ELSE	2	PS		1069	IF	1	ALP
*	1020	END	2	PS	*	1070	ELSE	1	BE
*	1021	.,	2	PS	*	1071	$	1	SBE
	1022	,	2	SWL	*	1072	END	1	BE
	1023	.,	2	SWL	*	1073	.,	1	BE
*	1024	SWI	3			1074	ELSE	1	BE
*	1025	(	3			1075	$	1	SBE
*	1026	L	3			1076	:=	3	
	1027	,	1	DE		1077	END	1	BE
	1028	.,	1	DE		1078	.,	1	BE
*	1029	ELSE	2	AS	*	1079	ELSE	1	BE
*	1030	END	2	AS	*	1080	$	3	
*	1031	.,	2	AS	*	1081	END	1	BE
	1032	%	1	SAE	*	1082	.,	1	BE
	1033	ELSE	1	AE		1083	ELSE	1	BE
	1034	:=	1	AV		1084	$	1	SBE
	1035	END	1	AE		1085	:=	1	BV
	1036	.,	1	AE		1086	END	1	BE
*	1037	%	1	SAE		1087	.,	1	BE
*	1038	ELSE	1	AE	*	1088	(	3	
*	1039	:=	3		*	1089	ELSE	1	BE
*	1040	END	1	AE	*	1090	$	1	SBE
*	1041	.,	1	AE	*	1091	:=	3	
	1042	%	1	SAE	*	1092	END	1	BE
	1043	ELSE	1	AE	*	1093	.,	1	BE
	1044	END	1	AE		1094	AAI	2	BLPL
	1045	.,	1	AE		1095	SAV	2	BLPL
*	1046	(	3			1096	BAI	2	BLPL
*	1047	%	1	SAE		1097	AFI	2	BLPL
*	1048	ELSE	1	AE		1098	(	2	BLPL
*	1049	:=	3			1099	AC	2	BLPL
*	1050	END	1	AE		1100	-	2	BLPL

PASS 2

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDU-CTION	*	NO.	LAST SYMBOL	STATE	REDU-CTION
	1101	IF	2	BLPL		1151	UNTIL	1	AE
	1102	BC	2	BLPL	*	1152	,	2	UNAE
	1103	£	2	BLPL	*	1153	DO	2	UNAE
	1104	SBV	2	BLPL		1154	,	1	AE
	1105	BFI	2	BLPL		1155	%	1	SAE
*	1106	ELSE	2	GOTOS		1156	DO	1	AE
*	1107	END	2	GOTOS	*	1157	,	1	AE
*	1108	.,	2	GOTOS	*	1158	(	3	
	1109	ELSE	1	DE	*	1159	%	1	SAE
	1110	END	1	DE	*	1160	DO	1	AE
	1111	.,	1	DE		1161	,	1	AE
*	1112	,	2	FOR2		1162	%	3	
*	1113	STEP	3			1163	DO	1	AE
*	1114	WHILE	3		*	1164	,	2	WHBE
*	1115	DO	2	FOR2	*	1165	DO	2	WHBE
	1116	,	1	AE		1166	,	1	BE
	1117	%	1	SAE		1167	$	1	SBE
	1118	STEP	1	AE		1168	DO	1	BE
	1119	WHILE	1	AE	*	1169	,	1	BE
	1120	DO	1	AE	*	1170	$	3	
*	1121	,	1	AE	*	1171	DO	1	BE
*	1122	(	3			1172	,	1	BE
*	1123	%	1	SAE		1173	(	3	
*	1124	STEP	1	AE		1174	$	1	SBE
*	1125	WHILE	1	AE		1175	DO	1	BE
*	1126	DO	1	AE	*	1176	AAI	2	FOR2,
	1127	,	1	AE	*	1177	SAV	2	FOR2,
	1128	%	3		*	1178	AFI	2	FOR2,
	1129	STEP	1	AE	*	1179	(	2	FOR2,
	1130	WHILE	1	AE	*	1180	AC	2	FOR2,
	1131	DO	1	AE	*	1181	-	2	FOR2,
*	1132	,	2	FOR2	*	1182	IF	2	FOR2,
*	1133	DO	2	FOR2		1183	AAI	2	FORC
	1134	UNTIL	3			1184	SAV	2	FORC
*	1135	AAI	2	FOR1		1185	BAI	2	FORC
*	1136	SAV	2	FOR1		1186	AFI	2	FORC
*	1137	AFI	2	FOR1		1187	IF	2	FORC
*	1138	(	2	FOR1		1188	SBV	2	FORC
*	1139	AC	2	FOR1		1189	BFI	2	FORC
*	1140	-	2	FOR1		1190	L	2	FORC
*	1141	IF	2	FOR1		1191	PI	2	FORC
	1142	,	2	ESTEP		1192	GOTO	2	FORC
	1143	DO	2	ESTEP		1193	FOR	2	FORC
*	1144	UNTIL	2	STAE		1194	DUMMYS	2	FORC
	1145	%	1	SAE		1195	BEGINB	2	FORC
	1146	UNTIL	1	AE		1196	BEGIN	2	FORC
*	1147	(	3		*	1197		2	CMT
*	1148	%	1	SAE	*	1198	ELSE	2	CMT
*	1149	UNTIL	1	AE	*	1199	END	2	CMT
	1150	%	3		*	1200	.,	2	CMT

PASS 2

FINAL TABLE (FT)

*	NO.	LAST SYMBOL	STATE	REDU-CTION	*	NO.	LAST SYMBOL	STATE	REDU-CTION
	1201	AAI	2	S.,		1251	*)	2	BP
	1202	SAV	2	S.,	*	1252	AAI	2	BPL,
	1203	BAI	2	S.,	*	1253	SAV	2	BPL,
	1204	AFI	2	S.,	*	1254	AFI	2	BPL,
	1205	IF	2	S.,	*	1255	(	2	BPL,
	1206	SBV	2	S.,	*	1256	AC	2	BPL,
	1207	BFI	2	S.,	*	1257	-	2	BPL,
	1208	L	2	S.,	*	1258	IF	2	BPL,
	1209	PI	2	S.,		1259	%	1	SAE
	1210	GOTO	2	S.,		1260	:	1	AE
	1211	FOR	2	S.,	*	1261	(	3	
	1212	DUMMYS	2	S.,	*	1262	%	1	SAE
	1213	BEGINB	2	S.,	*	1263	:	1	AE
	1214	BEGIN	2	S.,		1264	%	3	
*	1215	END	3			1265	:	1	AE
*	1216	.,	3		*	1266	,	2	BPL
	1217	AAI	2	BLH.,	*	1267	*)	2	BPL
	1218	SAV	2	BLH.,		1268	(*	3	
	1219	BAI	2	BLH.,		1269	,	3	
	1220	AFI	2	BLH.,	*	1270	AAI	3	
	1221	IF	2	BLH.,		1271	,	2	(*BP*)
	1222	SBV	2	BLH.,		1272	.,	2	(*BP*)
	1223	BFI	2	BLH.,	*	1273	BAI	3	
	1224	L	2	BLH.,		1274	AAI	2	AAL,
	1225	PI	2	BLH.,	*	1275	,	2	AAL
	1226	GOTO	2	BLH.,	*	1276	.,	2	AAL
	1227	FOR	2	BLH.,		1277	BAI	2	BAL,
	1228	DUMMYS	2	BLH.,	*	1278	,	2	BAL
	1229	BEGINB	2	BLH.,	*	1279	.,	2	BAL
	1230	BEGIN	2	BLH.,		1280	,	1	AAL
	1231	ARRAY	2	BLH.,		1281	.,	1	AAL
	1232	SWITCH	2	BLH.,	*	1282	,	1	BAL
	1233	PROCED	2	BLH.,	*	1283	.,	1	BAL
*	1234		2	UBL		1284	,	3	
*	1235	ELSE	2	UBL		1285	.,	2	AD
*	1236	END	2	UBL	*	1286	SWI	2	SWL,
*	1237	.,	2	UBL	*	1287	(	2	SWL,
	1238	.,	2	BLH	*	1288	IF	2	SWL,
*	1239	.,	1	U	*	1289	L	2	SWL,
	1240	AAI	3			1290	.,	2	SWID
	1241	BAI	3		*	1291	AAI	2	RPH.,
*	1242	SWI	3		*	1292	SAV	2	RPH.,
	1243	AFI	3		*	1293	BAI	2	RPH.,
	1244	BFI	3		*	1294	AFI	2	RPH.,
	1245	PI	3		*	1295	IF	2	RPH.,
*	1246		2	UCMS	*	1296	SBV	2	RPH.,
*	1247	ELSE	2	UCMS	*	1297	BFI	2	RPH.,
*	1248	END	2	UCMS	*	1298	L	2	RPH.,
*	1249	.,	2	UCMS	*	1299	PI	2	RPH.,
	1250	,	2	BP	*	1300	GOTO	2	RPH.,

```
PASS 2
                          FINAL TABLE (FT)

*  NO.  LAST   STATE REDU-    *  NO.  LAST   STATE REDU-
        SYMBOL       CTION            SYMBOL       CTION

* 1301     FOR   2   RPH.,
* 1302  DUMMYS   2   RPH.,
* 1303  BEGINB   2   RPH.,
* 1304   BEGIN   2   RPH.,
  1305      .,   2     RPH
* 1306      .,   1       S
  1307       (   3
  1308      .,   1       S
* 1309      .,   2      PD
  1310    ELSE   3
  1311      .,   1       S
```

A WORKED EXAMPLE FOR PASS 2: THE FORMAL TEXT IS FOLLOWED BY THE CORRESPONDING REDUCING STACK HISTORY.

```
BEGINB
 PROCED AFI .,
  BEGINB
   SAV := AC .,
   FOR SAV := SAV STEP AC UNTIL SAV DO SAV := SAV % SAV .,
   AFI := SAV
  END .,
 PI ( SAV , SAV ) .,
 PI ( SAV , SAV , AFI ( SAV , AC , AC , AFI ( SAV , AC , AC ,
                         SAV % SAV % ( SAV % SAV ) ) ) )
END '
```

NS	P	L	S(P-1)	S(P)	LS	ST	RED
2	1	1		BEGINB	PROCED	3	
3	2	2	BEGINB	PROCED	AFI	3	
4	3	3	PROCED	AFI	.,	2	RPH
4	2	4	BEGINB	RPH	.,	3	
5	3	5	RPH	.,	BEGINB	2	RPH.,
5	2	6	BEGINB	RPH.,	BEGINB	3	
6	3	7	RPH.,	BEGINB	SAV	3	
7	4	8	BEGINB	SAV	:=	1	AV
7	4	9	BEGINB	AV	:=	3	
8	5	10	AV	:=	AC	2	AV:=
8	4	11	BEGINB	AV:=	AC	1	ALPL
8	4	12	BEGINB	ALPL	AC	3	
9	5	13	ALPL	AC	.,	1	AE
9	5	14	ALPL	AE	.,	2	AS
9	4	15	BEGINB	AS	.,	1	S
9	4	16	BEGINB	S	.,	3	
10	5	17	S	.,	FOR	2	S.,
10	4	18	BEGINB	S.,	FOR	3	
11	5	19	S.,	FOR	SAV	3	
12	6	20	FOR	SAV	:=	1	AV
12	6	21	FOR	AV	:=	3	
13	7	22	AV	:=	SAV	2	AV:=
13	6	23	FOR	AV:=	SAV	2	FOR1
13	5	24	S.,	FOR1	SAV	3	
14	6	25	FOR1	SAV	STEP	1	AE
14	6	26	FOR1	AE	STEP	3	
15	7	27	AE	STEP	AC	3	
16	8	28	STEP	AC	UNTIL	1	AE
16	8	29	STEP	AE	UNTIL	2	STAE
16	7	30	AE	STAE	UNTIL	2	AESTAE
16	6	31	FOR1	AESTAE	UNTIL	3	
17	7	32	AESTAE	UNTIL	SAV	3	
18	8	33	UNTIL	SAV	DO	1	AE
18	8	34	UNTIL	AE	DO	2	UNAE
18	7	35	AESTAE	UNAE	DO	2	ESTEP
18	6	36	FOR1	ESTEP	DO	2	FOR2
18	5	37	S.,	FOR2	DO	3	
19	6	38	FOR2	DO	SAV	2	FORC
19	5	39	S.,	FORC	SAV	3	
20	6	40	FORC	SAV	:=	1	AV
20	6	41	FORC	AV	:=	3	
21	7	42	AV	:=	SAV	2	AV:=
21	6	43	FORC	AV:=	SAV	1	ALPL

NS	P	L	S(P-1)	S(P)	LS	ST	RED
21	6	44	FORC	ALPL	SAV	3	
22	7	45	ALPL	SAV	%	1	SAE
22	7	46	ALPL	SAE	%	3	
23	8	47	SAE	%	SAV	2	SAE%
23	7	48	ALPL	SAE%	SAV	3	
24	8	49	SAE%	SAV	.,	1	APR
24	8	50	SAE%	APR	.,	2	SAE
24	7	51	ALPL	SAE	.,	1	AE
24	7	52	ALPL	AE	.,	2	AS
24	6	53	FORC	AS	.,	1	S
24	6	54	FORC	S	.,	2	FORS
24	5	55	S.,	FORS	.,	1	S
24	5	56	S.,	S	.,	3	
25	6	57	S	.,	AFI	2	S.,
25	5	58	S.,	S.,	AFI	3	
26	6	59	S.,	AFI	:=	3	
27	7	60	AFI	:=	SAV	2	ALP
27	6	61	S.,	ALP	SAV	1	ALPL
27	6	62	S.,	ALPL	SAV	3	
28	7	63	ALPL	SAV	END	1	AE
28	7	64	ALPL	AE	END	2	AS
28	6	65	S.,	AS	END	1	S
28	6	66	S.,	S	END	3	
29	7	67	S	END	.,	2	CMT
29	6	68	S.,	CMT	.,	2	CMT
29	5	69	S.,	CMT	.,	2	CMT
29	4	70	BEGINB	CMT	.,	2	UBL
29	3	71	RPH.,	UBL	.,	1	S
29	3	72	RPH.,	S	.,	2	PD
29	2	73	BEGINB	PD	.,	1	D
29	2	74	BEGINB	D	.,	2	BLH
29	1	75		BLH	.,	3	
30	2	76	BLH	.,	PI	2	BLH.,
30	1	77		BLH.,	PI	3	
31	2	78	BLH.,	PI	(	3	
32	3	79	PI	(	SAV	2	PI(
32	2	80	BLH.,	PI(	SAV	3	
33	3	81	PI(	SAV	,	1	ACPL
33	3	82	PI(	ACPL	,	3	
34	4	83	ACPL	,	SAV	2	ACPL,
34	3	84	PI(	ACPL,	SAV	3	
35	4	85	ACPL,	SAV	)	1	ACP
35	4	86	ACPL,	ACP	)	2	ACPL
35	3	87	PI(	ACPL	)	3	
36	4	88	ACPL	)	.,	2	ACPL)
36	3	89	PI(	ACPL)	.,	2	PS
36	2	90	BLH.,	PS	.,	1	S
36	2	91	BLH.,	S	.,	3	
37	3	92	S	.,	PI	2	S.,
37	2	93	BLH.,	S.,	PI	3	
38	3	94	S.,	PI	(	3	
39	4	95	PI	(	SAV	2	PI(
39	3	96	S.,	PI(	SAV	3	
40	4	97	PI(	SAV	,	1	ACPL
40	4	98	PI(	ACPL	,	3	
41	5	99	ACPL	,	SAV	2	ACPL,
41	4	100	PI(	ACPL,	SAV	3	
42	5	101	ACPL,	SAV	,	1	ACP
42	5	102	ACPL,	ACP	,	2	ACPL

NS	P	L	S(P-1)	S(P)	LS	ST	RED
42	4	103	PI(	ACPL	,	3	
43	5	104	ACPL	,	AF I	2	ACPL,
43	4	105	PI(	ACPL,	AF I	3	
44	5	106	ACPL,	AF I	(	3	
45	6	107	AF I	(	SAV	2	AF I(
45	5	108	ACPL,	AF I(	SAV	3	
46	6	109	AF I(	SAV	,	1	ACPL
46	6	110	AF I(	ACPL	,	3	
47	7	111	ACPL	,	AC	2	ACPL,
47	6	112	AF I(	ACPL,	AC	3	
48	7	113	ACPL,	AC	,	1	ACP
48	7	114	ACPL,	ACP	,	2	ACPL
48	6	115	AF I(	ACPL	,	3	
49	7	116	ACPL	,	AC	2	ACPL,
49	6	117	AF I(	ACPL,	AC	3	
50	7	118	ACPL,	AC	,	1	ACP
50	7	119	ACPL,	ACP	,	2	ACPL
50	6	120	AF I(	ACPL	,	3	
51	7	121	ACPL	,	AF I	2	ACPL,
51	6	122	AF I(	ACPL,	AF I	3	
52	7	123	ACPL,	AF I	(	3	
53	8	124	AF I	(	SAV	2	AF I(
53	7	125	ACPL,	AF I(	SAV	3	
54	8	126	AF I(	SAV	,	1	ACPL
54	8	127	AF I(	ACPL	,	3	
55	9	128	ACPL	,	AC	2	ACPL,
55	8	129	AF I(	ACPL,	AC	3	
56	9	130	ACPL,	AC	,	1	ACP
56	9	131	ACPL,	ACP	,	2	ACPL
56	8	132	AF I(	ACPL	,	3	
57	9	133	ACPL	,	AC	2	ACPL,
57	8	134	AF I(	ACPL,	AC	3	
58	9	135	ACPL,	AC	,	1	ACP
58	9	136	ACPL,	ACP	,	2	ACPL
58	8	137	AF I(	ACPL	,	3	
59	9	138	ACPL	,	SAV	2	ACPL,
59	8	139	AF I(	ACPL,	SAV	3	
60	9	140	ACPL,	SAV	%	1	SAE
60	9	141	ACPL,	SAE	%	3	
61	10	142	SAE	%	SAV	2	SAE%
61	9	143	ACPL,	SAE%	SAV	3	
62	10	144	SAE%	SAV	%	1	APR
62	10	145	SAE%	APR	%	2	SAE
62	9	146	ACPL,	SAE	%	3	
63	10	147	SAE	%	(	2	SAE%
63	9	148	ACPL,	SAE%	(	3	
64	10	149	SAE%	(	SAV	3	
65	11	150	(	SAV	%	1	SAE
65	11	151	(	SAE	%	3	
66	12	152	SAE	%	SAV	2	SAE%
66	11	153	(	SAE%	SAV	3	
67	12	154	SAE%	SAV	)	1	APR
67	12	155	SAE%	APR	)	2	SAE
67	11	156	(	SAE	)	1	AE
67	11	157	(	AE	)	2	(AE
67	10	158	SAE%	(AE	)	3	
68	11	159	(AE	)	)	2	APR
68	10	160	SAE%	APR	)	2	SAE
68	9	161	ACPL,	SAE	)	1	ACP
68	9	162	ACPL,	ACP	)	2	ACPL

NS	P	L	S(P-1)	S(P)	LS	ST	RED
68	8	163	AFI(	ACPL	)	3	
69	9	164	ACPL	)	)	2	ACPL)
69	8	165	AFI(	ACPL)	)	2	AFD
69	7	166	ACPL,	AFD	)	1	ACP
69	7	167	ACPL,	ACP	)	2	ACPL
69	6	168	AFI(	ACPL	)	3	
70	7	169	ACPL	)	)	2	ACPL)
70	6	170	AFI(	ACPL)	)	2	AFD
70	5	171	ACPL,	AFD	)	1	ACP
70	5	172	ACPL,	ACP	)	2	ACPL
70	4	173	PI(	ACPL	)	3	
71	5	174	ACPL	)	END	2	ACPL)
71	4	175	PI(	ACPL)	END	2	PS
71	3	176	S.,	PS	END	1	S
71	3	177	S.,	S	END	3	
72	4	178	S	END		2	CMT
72	3	179	S.,	CMT		2	CMT
72	2	180	BLH.,	CMT		2	UBL
72	1	181		UBL		1	PROGR

J. SZCZEPKOWICZ (Wrocław)

O TABLICOWYM STEROWANIU PROCESEM WERYFIKACJI SKŁADNIOWEJ W TRANSLATORZE ALGOLU

STRESZCZENIE

Praca zawiera opis pewnych ogólnych metod zastosowanych do konstrukcji translatora ALGOLu dla maszyny cyfrowej ODRA 1204. Realizacja ALGOLu dla tej maszyny i translator zostały opracowane w Katedrze Metod Numerycznych Uniwersytetu Wrocławskiego.

Opisywany translator składa się z trzech kolejno pracujących części, przy czym każda część translatora przegląda tekst tłumaczonego programu tylko raz, bez wybiegania naprzód i bez cofania się wstecz.

Część I czyta taśmę z programem w ALGOLu i zapamiętuje istotne części jego tekstu w zwartej postaci. W czasie zapamiętywania tekstu wytwarza się opis struktury blokowej programu i niekompletne jeszcze charakterystyki obiektów używanych w tym programie. Następnie sterowanie przekazuje się do części II, która w razie istotnej potrzeby może zniszczyć zapis części I w pamięci maszyny.

Wykorzystując zebrane dotąd informacje o programie, część II traktuje ten program jako tekst formalny w pewnym języku redukcyjnym i wykonuje pełną weryfikację składniową tego tekstu. Równolegle z badaniem składni tekstu programu część II wykonuje wstępne przekształcenie tego tekstu i bada jego poprawność semantyczną.

Po przekazaniu sterowania do części III dopuszcza się — w razie istotnej potrzeby — zniszczenie zapisu obu poprzednich części w pamięci maszyny. Część III pracuje przy założeniu, że w czasie działania obu poprzednich części nie wykryto żadnego błędu w tekście programu tłumaczonego. Zadaniem tej części translatora jest utworzenie możliwie optymalnego ciągu rozkazów w kodzie wewnętrznym maszyny, równoważnego przeczytanemu programowi w ALGOLu.

Dwie pierwsze części translatora są sterowane za pomocą skróconych tablic przejść redukcyjnych otrzymanych w wyniku przekształcenia zwykłych tablic przejść redukcyjnych zbudowanych dla odpowiednich języków formalnych. Omawiane tablice zostały utworzone przez autora na maszynie Elliott 803. Danymi dla maszyny były pewne gramatyki formalne zapisane w ogólnie znanej symbolice. Praca zawiera m. in. te gramatyki wraz ze skróconymi tablicami przejść i szczegółowy opis stosowania tych tablic w translatorze. Omówiono także sposoby zapamiętywania podstawowych informacji w maszynie oraz czas trwania tłumaczenia.

Część III translatora nie jest w pracy omówiona i będzie przedmiotem oddzielnej publikacji.

Й. ЩЕПКОВИЧ (Вроцлав)

ТАБЛИЧНО УПРЯВЛЯЕМЫЙ КОНТРОЛЬ СИНТАКСИЧЕСКОЙ ПРАВИЛЬНОСТИ В ТРАНСЛЯТОРЕ ЯЗЫКА АЛГОЛ

РЕЗЮМЕ

В работе описаны некоторые общие методы, использованные в трансляторе языка АЛГОЛ для польской электронной вычислительной машины ОДРА 1204. Эти методы остаются важными для некоторой категории вычислительных машин и некоторой категории формальных языков.

Проблема трансляции с АЛГОЛа на внутренний код машины ОДРА 1204 решается в трех фазах. В каждой фазе совершается один просмотр текста.

В первой фазе выполняется простое преобразование входной программы на внутреннюю репрезентацию и анализируется блочная структура этой программы. Во второй фазе рассматривается внутренняя репрезентация программы как некоторый формальный текст и выполняется полный синтаксический контроль этого текста. Семантические свойства тоже контролируются.

Третья фаза работает только на правильной внутренней репрезентации программы и переводит её на почти оптимальную последовалельность машинных команд, равносильную входной программе.

Две первые фазы упрявляются с помощью синтаксических таблиц, которые были построены вычислительной машиной на основе некоторых формальных грамматик. В работе приведены эти грамматики в таком виде, в каком они были представлены машине, а также полученные таблицы в явной форме. Подробно описывается использование этих таблиц в трансляторе и некоторые проблемы времени трансляции и памяти. Полученный транслятор анализирует и переводит входной текст со скоростью 3000 машинных слов в минуту (скорость ЭВМ ОДРА 1204-50 000 операций в секунду).

Последняя фаза транслятора в работе не затрагивается.