GRAMMATICAL MANIPULATION PACKAGE

EXPLORATORY STEPS TOWARDS
A
GRAMMATICAL MANIPULATION PACKAGE (GRAMPA)

By

KEITH ROGER BARNES, B.Sc.

A Project
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree
Master of Science

McMaster University
November 1972

MASTER OF SCIENCE (Computation)

McMASTER UNIVERSITY Hamilton, Ontario.

TITLE: Exploratory Steps Towards a Grammatical Manipulation Package (GRAMPA)

AUTHOR: Keith Roger Barnes, B.Sc. (Birmingham University)

SUPERVISOR: Dr. Derick Wood

NUMBER OF PAGES: viii, 81, A1(7), A2(5), A3(3), A4(2), A5(37)

ACKNOWLEDGMENTS

The author gratefully acknowledges the guidance of his supervisor, *Dr. Derick Wood.*
The author also expresses his gratitude to his wife, *Maryke,* for her encouragement and understanding.

Special thanks go to *Mrs. Joanne Straughan,* for her patience in typing this project.

TABLE OF CONTENTS

INTRODUCTION 1

CHAPTER 1: DEFINITIONS AND TERMINOLOGY

1.1 Introduction 4

1.2 Context Free Grammars 5

1.3 Admissible Grammars 7

1.4 Left and Right sets 9

1.5 Simple Precedence Grammars 10

1.6 Removing Precedence Conflicts 13

1.7 Precedence Functions 15

1.8 Extended Precedence 16

CHAPTER 2: THE ALGORITHMS AND PROCEDURES IN GRAMPA

2.1 Introduction 18

PART I GRAMMAR INPUT 19

2.2 Table Construction 19

2.3 The Syntax Graph 19

2.4 Symbol Tables 27

2.5 Procedure INGRAMMAR 29

2.6 Procedure PRINT TABLES 42

2.7 Procedure PRINT GRAMMAR 42

TABLE OF CONTENTS CONT'D

PART II GRAMMAR ANALYSES 43

2.8 Admissability of Grammars 43

2.9 Algorithms and Procedures for Simple Precedence Analysis 47

2.10 Checks for Recursion 58

CHAPTER 3: USE OF GRAMPA WITH EXAMPLES

3.1 Programming Considerations 60

3.2 Example 1: Simple Phrase Structure Language 64

3.3 Example 2: Admissable Test 72

3.4 Example 3: Euler 75

CHAPTER 4: FUTURE DIRECTIONS 77

REFERENCES 80

APPENDIX 1: OUTPUT FROM GRAMPA FOR SIMPLE PHRASE STRUCTURE LANGUAGE

APPENDIX 2: FIRST PART OF GRAMPA OUTPUT FOR EULER

APPENDIX 3: HASH CODING TECHNIQUE OF MORRIS

APPENDIX 4: CENTRAL PROCESSOR TIME MODEL

APPENDIX 5: PROGRAM LISTING

TABLE OF DIAGRAMS

Figure		Page
2.1	Expression Tree for the nonterminal Simple Arithmetic Expression, SAE	23
Text	Representation of SAE in the syntax graph	25
2.2	Outline of the Hashing Procedure for Terminal Symbols	26
2.3	Procedures Used by INGRAMMAR	30
2.4	Operation of INGRAMMAR	
	Entry State	34
	State 2	35
	State 3	36
2.5	Illustration of the backward list kept in the Syntax Graph	40
Text	Illustration of the use of RHSTABLE in the procedure TRANSPOSE SYNTAX GRAPH	
3.1	Overall Structure of GRAMPA	

Exhibits

3.1	First page of GRAMPA output for the Simple Phrase Structure Language of Wirth-Weber	61
3.2	Modified Simple Phrase Language to Introduce Conflicts	68
3.3	Output from GRAMPA for Modified Simple Phrase Language	69
3.4	Modified Simple Phrase Language After Removal of Conflicts	70

TABLE OF DIAGRAMS CONT'D

3.5 Symbols and Syntax Graph for Example Grammar to Test ADMISSABLE 73

3.6 Output from ADMISSABLE 74

Tables

1 Allowable Symbols and Their GRAMPA Values 32

PREFACE

Very often, grammars constructed for computer languages are not in a concise form for simple parsing. For example some symbols may be unreachable or useless. If a simple precedence grammar is required, artificial symbols may have to be introduced to remove conflicts.

This report describes exploratory steps taken towards the development of an Algol program to automatically manipulate grammars. Procedures are described which read and set up a grammar in a list structure form suitable for analysis and manipulation. The procedures manipulate the grammar to remove useless and unreachable symbols, and precedence conflicts, and they analyse the grammar for recursion, precedence etc.

INTRODUCTION

Modern science is expanding rapidly in all disciplines with a subsequent increasing demand on the computer for data analysis and manipulation. However, the computer user still has to translate his problem into computer terms via programming languages. The most common general purpose languages are often cumbersome to adapt and use in specialized problem areas, hence a need exists for special purpose languages with which the user can "talk" to the computer in his own terms. These special purpose languages, often called Problem Oriented Languages (POLs), may be very particular to a discipline, needed quickly, and needed only for a relatively short period of time. It is essential, therefore, that automatic techniques are developed for producing compilers or translators for these languages. It should only be necessary for the specialist or analyst to specify the syntax and semantics of the language in some standard form to the computer to obtain the translator.

Most POLs are generated using context-free grammars. However, very often the grammar developed is ambiguous or unsuitable for straightforward parsing. It may have

to be changed or manipulated several times before it is in a concise, acceptable form.

This report describes exploratory steps taken towards the development of an automatic grammatical manipulation system. An Algol program, nicknamed GRAMPA (GRAmmatical Manipulation PAckage), which comprises various analysis and manipulation procedures for context-free grammars, is described in detail.

The procedures in GRAMPA accept a grammar specified in an "inverse" Backus-Naur Form on cards, and analyze it for such things as recursion, precedence, the usefulness and reachability of productions, etc.;and manipulate the grammar to remove useless productions and precedence conflicts. Many tables are produced in the course of the analysis which can be used in the syntax analysis section of compiler for the language generated by the grammar. Specifically, the procedures will produce:

i) a neat, readible listing of the grammar

ii) a syntax graph of the grammar,

iii) tables of the terminal and nonterminal symbols with cross reference hash tables,

iv) lists of the left, right and embedded recursive symbols,

v) the left and right sets of the grammar (Wirth-Weber[14]),

vi) a simple precedence matrix (Wirth-Weber[14]) with an explanation of conflicts,

vii) the precedence functions,

viii) lookup tables for use in a syntax analyser;

and will manipulate the grammar to remove:

(i) useless productions,

(ii) unreachable nonterminals

(iii) precedence conflicts.

The body of this report is divided into three chapters. The first chapter introduces the terminology and notation used in relation to context-free grammars. The second chapter describes the algorithms and procedures themselves, while the third chapter describes the use of the system with examples. A fourth chapter on possible future directions is included.

CHAPTER I

DEFINITIONS AND TERMINOLOGY

1.1 Introduction

The development of Algol 60[12] led to the development of a meta-language called Backus-Naur Form or BNF, named after two of the developers of the language. This meta-language was used to define the syntactic structure of Algol 60 programs irrespective of their meaning; where, informally, we consider syntax to be a specification of the well-formed statements of a language, usually incorporating a mechanism for structural descriptions. (Semantics, on the other hand, can be thought of as the specification of how these statements are to be executed by a real or abstract computer)[(5)]. BNF can in fact be used to describe the structure of any context-free grammar[(17)]. It is this kind of grammar which is used most often in practical language development, or for Problem Oriented Languages. Therefore, the GRAMPA system is designed to handle only context-free grammars.

This chapter of the report serves to introduce some definitions and notations used in relation to grammars. The definitions will be restricted to the family of

context-free grammars. The symbols and terminology used are those of Graham[8]. A good list of alternative notations used by other authors can be found in McKeeman[10].

1.2 Context Free Grammars (CFGs)

1.2.1 A context-free phrase structure grammar G is a 4-tuple $G=(V_N, V_T, P, S)$, where V_N is a finite nonempty set of symbols, V_T is a finite set of symbols, P is a finite nonempty set of productions (rules) and S is a distinguished (initial or sentence) symbol. Now $V_N \cap V_T = \phi$, the empty set, and $V= V_N \cup V_T$ is the vocabulary of the grammar. V^* is the set of all strings over V, and ε denotes the empty string. V^+ denotes the set of nonempty strings over V. Thus $V^+ = V^* -\{\varepsilon\}$. Elements of V are denoted by capital roman letters; elements of V^* are denoted by small Greek letters and elements of V^+ are denoted by small roman letters. A production is of the form $A \to \alpha$, where A is called the left part or left hand side of the production and α is the right part or right hand side. V_N is the set of symbols which occur as left parts of productions--nonterminal symbols. V_T is the set of terminal symbols which do not occur as left parts. S is the unique nonterminal symbol which does not occur in a right part.

1.2.2 With respect to a grammar G, we say $a \Rightarrow b$ if there exist σ, π in V^*, U in V_N and $U \rightarrow \mu$ in P such that $a = \sigma U \pi$ and $b = \sigma \mu \pi$. If $a_0 \Rightarrow a_1 \Rightarrow \ldots\ldots \Rightarrow a_n$, where a_i is in V^+ for $0 \leq i \leq n$, then $a_0 \overset{*}{\Rightarrow} a_n$ if $n \geq 0$ (reflexive transitive closure of $\Rightarrow$), and $a_0 \overset{+}{\Rightarrow} a_n$ if we require $n > 0$ (transitive closure of $\Rightarrow$). The sequence $a_0 \Rightarrow a_1 \Rightarrow \ldots\ldots \Rightarrow a_n$ is called the <u>derivation</u> of a_n from a_0 of length n.

1.2.3 A string u is a U-<u>derivative</u> if $U \overset{+}{\Rightarrow} u$. It is an <u>immediate</u> U-derivative if $U \Rightarrow u$ (i.e. there is a production $U \rightarrow u$).

1.2.4 A string u is a <u>sentential form</u> if u is an S-derivative (where S is the distinguished symbol). A <u>sentence</u> is a sentential form consisting only of terminals. The <u>language</u> defined (or generated) by G is denoted L(G) and is the set of sentences of G. Thus $L(G) = \{u \mid S \overset{+}{\Rightarrow} u$ and $u \in V_T^+\}$.

1.2.5 If $a \overset{+}{\Rightarrow} b$ where $a = \sigma U \pi$, $b = \sigma u \pi$ and $U \overset{+}{\Rightarrow} u$, then u is a <u>phrase</u> in b. If $a \Rightarrow b$ (and $U \rightarrow u$) then u is an <u>immediate</u> or <u>simple phrase</u> (of U) in b. The process of constructing a derivation of a sentence starting from the sentence and working back to the distinguished symbol is called <u>parsing</u>, and the derivation so obtained is called a parse of the sentence. If $a \Rightarrow b$ and u is an immediate

phrase of U in b, then the parsing step from b to a is called a reduction of u to U.

1.2.6 Given a derivation $a=x_1 \Rightarrow x_2 \ldots\ldots \Rightarrow x_n=b$, the derivation is a right-most derivation if, for $1 \leq i < n$, $x_i = \sigma_i U_i \pi_i$, $x_{i+1} = \sigma_i u_i \pi_i$ where $U_i \rightarrow u_i$ is a production and $\pi_i \varepsilon V_T^*$. We take the right-most derivation to be the canonical derivation of the set of derivations that differ only in the order of application of productions. A parse which represents a canonical derivation is a canonical (or left-to-right) parse; reduction of a left-most immediate phrase of a sentential form is a canonical reduction.

A grammar G is ambiguous if some sentence of G has more than one canonical derivation.

1.3 Admissable Grammars [17]

A grammar is an admissable or reduced grammar if:

(i) for all X in V, there is a sentential derivation, $S \overset{*}{\Rightarrow} uXv$, for some u,v in V^* i.e. X is reachable, and

(ii) for all X in V_N, there is at least one derivation $X \overset{+}{\Rightarrow} x$, where x is a terminal word i.e. X is useful (otherwise X is useless).

If a symbol in a grammar is unreachable, then it clearly plays no part in the language generated by the

grammar. Similarly, if a nonterminal is reachable but it is useless, then it cannot participate in the generation of terminal words, by definition.

Example:

let $G = (V_N, V_T, P, S)$, where

$V_N = \{S, X, Y, Z\}$

$V_T = \{a, b\}$

P: $S \to aX | aYb$

$X \to aX | aaX$

$Y \to aYb | \varepsilon$

$Z \to aZ | b$,

trivially Z is unreachable as it appears nowhere on the right hand side of a nonterminal other than itself, and X is useless. We can, therefore, replace G by:

$G^1 = (\{S,Y\}, \{a,b\}, \{S \to aYb, Y \to aYb | \varepsilon\}, S)$.

If a nonterminal had no rules associated with it, then it would be useless under the definition given above. This implies that each nonterminal, apart from the sentence or distinguished symbol, must have at least one rule associated with it.

The algorithms for determining reachability and usefulness will be presented with the Algol procedures in section 2.8.

1.4 Left and Right Sets

1.4.1 With respect to a grammar G, we define the left set or left part of a nonterminal X, denoted by $\mathcal{L}(X)$, to be the set of all symbols which can occur as the left-most symbol of an X-derivative. Thus $\mathcal{L}(X)=\{A|X\overset{+}{\Longrightarrow}A....\}$. Analogously, we define the right set or right part of X: $\mathcal{R}(X) = \{A|X\overset{+}{\Longrightarrow}....A\}$. We can extend this notation to all symbols of a vocabulary by defining $\mathcal{L}(X) = \mathcal{R}(X) = \phi$ for every terminal symbol, X.

Example (Wirth-Weber)

$G=(V_N, V_T, P, S)$,

where $V_N=\{S,H\}$

$V_T=\{\lambda,]\}$

P: $S\rightarrow H]$

$H\rightarrow]$

$H\rightarrow H\lambda$

$H\rightarrow HS$

Then the left and right sets of the nonterminals, S and H are as follows:

U	$\mathcal{L}(U)$	$\mathcal{R}(U)$
S	]H	]
H	]H	]λS

The computer alogorithm used for finding the left and right sets is given in the second part of chapter 2.

1.4.2 A nonterminal X is <u>left</u> (<u>right</u>) <u>recursive</u> if $X \in \mathcal{L}(X)$ ($X \in \mathcal{R}(X)$); it is self-imbedding if $X \overset{+}{\Rightarrow} aXb$. If $X \to x$ and $x \overset{*}{\Rightarrow} X \ldots$ ($x \overset{*}{\Rightarrow} \ldots X$, $x \overset{*}{\Rightarrow} aXb$) then $X \to x$ is a <u>left recursive</u> (<u>right recursive</u>, <u>self-imbedding</u>) <u>rule</u>. $X \to x$ is <u>directly</u> left recursive (directly right recursive, directly self-imbedding) if $x = X \ldots$ ($x = \ldots X$, $x = aXb$).

1.5 Simple Precedence Grammars

The notion of simple precedence grammars (and simple precedence languages) introduced by Wirth and Weber[14] is the following:

Let $G = (V_N, V_T, P, S)$ be a context free grammar.

For any A, B, ε V, we define the following simple precedence relations:

SP1) $A \doteq B$ iff P contains a rule of the form
$X \to \ldots AB \ldots$ for some $X \in V_N$

SP2) $A <\cdot B$ iff P contains a rule of the form
$X \to \ldots AY..$ for some $X, Y \in V_N$ and
$Y \underset{G}{\overset{+}{\Rightarrow}} B \ldots$ (that is, $B \in \mathcal{L}(Y)$)

SP3) $A \cdot> B$ iff P contains a rule of the form
$X \to \ldots YZ \ldots$ for some $X, Y \in V_N$, $Z \in V$
and $Y \underset{G}{\overset{+}{\Rightarrow}} \ldots A$ and $Z \underset{G}{\overset{*}{\Rightarrow}} B \ldots$
(that is, $Z = B$ or $B \in \mathcal{L}(Z)$ and $A \in \mathcal{R}(Y)$).

The relations have the following interpretation (with respect to sentential forms of G).

I1) The relation $\doteq$ holds between all (left-to-right) adjacent symbols in an immediate phrase.

I2) The relation $<\cdot$ holds between the symbol immediately preceding a phrase and the leftmost symbol of the phrase.

I3) The relation $\cdot>$ holds between the rightmost symbol of a phrase and the symbol immediately following it.

If $A <\cdot B$ or $A \doteq B$ or $A \cdot> B$ then $A \mathcal{R} B$ (that is, at least one simple precedence relation holds between A and B). Given any CF grammar, it is possible to determine which relations hold between any two symbols[14].

If a grammar has two (or more) rules of the form $X \rightarrow a$, $Y \rightarrow a$ where $X \neq Y$, we say that the grammar has <u>common right parts</u> a or <u>common right part rules</u> $X \rightarrow a$ and $Y \rightarrow a$.

A grammar $G = (V_N, V_T, P, S)$ is a <u>simple precedence grammar</u> if both the following conditions are satisfied:

1. for every (ordered) pair $A, B \in V$, at most one simple precedence relation holds between A and B. (If this condition holds, we say that G has <u>unique simple precedence relations</u>.)
2. G has no common right part rules.

If more than one simple precedence relation holds between some $A,B \in V$, we say that there is a precedence <u>conflict</u> between A and B. It is a <u>right conflict</u> for A and a <u>left conflict</u> for B. A grammar $G = (V_N, V_T, P, S)$ is said to have a precedence conflict if there is a precedence conflict between two of its symbols.

If $A \lessdot B$ or $A \doteq B$ we say that $A \leq\!\cdot\, B$. If $A \gtrdot B$ and $A \doteq B$, we say there is a <u>$\cdot\!\geq$-conflict</u> between A and B. If $A \lessdot B$ and $A \doteq B$ then there is a <u>$\leq\!\cdot$-conflict</u> between A and B.

Example (from Wirth and Weber[14])

$$V = \{A, B, [,], ;, X\}$$

1) $S \rightarrow A$

2) $A \rightarrow B;\ B$

3) $B \rightarrow [A]$

4) $B \rightarrow [X]$

5) $B \rightarrow X$

$[\doteq X$	(rule 4)	$X \doteq]$	(rule 4)
$[\lessdot X$	(rule 3)	$X \gtrdot]$	(rule 3)

There is a $\leq\!\cdot$ ($\lessdot$, $\doteq$) conflict between [and X; it is a left conflict for X and a right conflict for [. There is a $\cdot\!\geq$ conflict between X and].

The precedence relations which exist between the symbols of a grammar are best expressed by a precedence

matrix. For example, consider the following grammar from Wirth-Weber[14]:

$G=(V_N, V_T, P, S)$

$V_N = \{S, H\}$

$V_T = \{\lambda, [,]\}$

P: S→H]

H→[

H→Hλ

H→HS

Applying the rules to determine the precedence relations, we arrive at:

	S	H	λ	[	]
S	·>	·>	·>	·>	·>
H	≐	<·	≐	<·	≐
λ	·>	·>	·>	·>	·>
[	·>	·>	·>	·>	·>
]	·>	·>	·>	·>	·>

The derivation of this matrix and its representation in GRAMPA will be discussed later in chapter 2.

1.6 Removing Precedence Conflicts

Precedence conflicts can be removed by several means. One such method is to treat the grammar as a more general case of a precedence grammar called extended

precedence. This is described in section 1.8. The method described here will be restricted such that it does not cause a change in the terminal language that in turn requires a change in the associated semantics of any production of the grammar. The following definitions are given[7]:

(1) An <u>artificial production</u> is a production with no associated semantics and only one element on the right side (also called an intermediate production).

(2) A <u>left restricted expansion</u> (LRE) of the nonterminal A, replaces A on the right sides of all productions, except where it is the left-most symbol, by a new nonterminal A_i, and adds the artificial production $A_i \to A$ to the grammar.

(3) A <u>right restricted expansion</u> (RRE) of A replaces A in the right sides of all productions, except where it is the right-most symbol, by a new nonterminal A_i, and adds the artificial production $A_i \to A$ to the grammar.

The following rules (proven in George[7]) now hold for context free grammars:

(1) The precedence relation $\doteq$ between two symbols A and B can be changed to $<\cdot$ by an LRE of B.

i.e. a production of the form $U \to xABy$ becomes $U \to xAB_1y$,

and $B_1 \to B$ is added to the grammar - then $A \doteq B_1$ and $A <\cdot B$.
(2) The precedence relation $\doteq$ between two symbols A and B can be changed to $\cdot>$ by an RRE of A.
(3) The precedence relation $<\cdot$ between A and B can be changed to $\cdot>$ by an RRE of A.

The precedence conflicts which can occur between any two symbols are $(\doteq, <\cdot)$, $(\doteq, \cdot>)$, $(<\cdot, \cdot>)$ and $(=, <\cdot, \cdot>)$. The precedence conflict $(\doteq, <\cdot)$ between A and B can be removed by an LRE of B. The precedence violation $(=, \cdot>)$ between two symbols A and B can be removed by an RRE of A. The violation $(<\cdot, \cdot>)$ can be removed by an RRE of A, and the violation $(\doteq, <\cdot, \cdot>)$ can also be removed by an RRE of A. With all of these transformations, new violations can be introduced, thus the procedure is recursive.

1.7 Precedence Functions

A precedence matrix to be used in a syntax analyser would have n^2 elements where n is the number of symbols in the vocabulary. This often requires a large amount of storage for practical compilers. Often the precedence relations are such that two numeric functions (f,g) ranging over the set of symbols, can be found such that for all ordered pairs (X_i, X_j):

(a) $f(X_i) = g(X_j) \equiv X_i \doteq X_j$

(b) $f(X_i) < g(X_j) \equiv X_i <\cdot X_j$

(c) $f(X_i) > g(X_j) \equiv X_i \cdot> X_j$

We now only require 2n locations to store these functions (if they exist).

The precedence relationships for the example grammar in section 1.5 can be represented by the two functions f and g, where:

X=	S	H	λ	[	]
f(X)	3	1	3	3	3
g(X)	1	2	1	2	1

The procedure for producing precedence functions is described in section 2.9.5.

1.8 <u>Extended Precedence</u>

Wirth and Weber[14] point out that sometimes precedence conflicts between symbols in a grammar can be resolved by looking to the left or to the right of the pair in conflict; that is, by extending the definitions of the precedence relations to strings of symbols. For instance, in the previous conflict example, [≐X] and [<·X;, also [X≐] and ;X >]. Wirth and Weber present some informal discussion and then formal definitions for the extended relations. However, the definitions they give do not correspond to the interpretations of the relations with respect to phrase detection.

Graham[8] defines the extended precedence relations with respect to canonical derivations, since these are the derivations which the parsing method Wirth and Weber present is intended to construct. Since the extensions are intended to resolve conflicts by looking at preceding and succeeding symbols, Graham extends the relations to strings which are adjacent in canonical sentential forms (rather than restricting the definitions to strings generated from the same rule, as Wirth and Weber do).

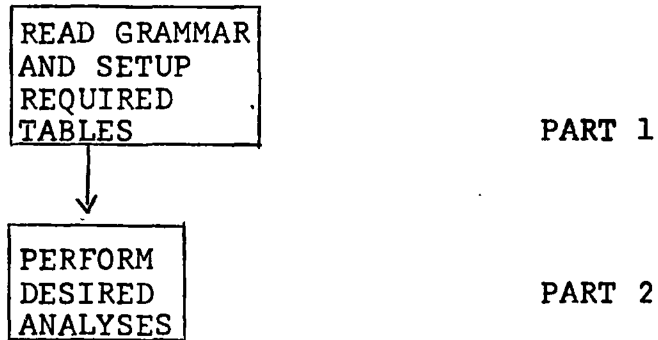
The definitions of the extended precedence relations will not be given here (see [8]), since they have not been incorporated into GRAMPA.

CHAPTER 2

THE ALGORITHMS AND PROCEDURES IN GRAMPA

2.1 Introduction

The procedures which comprise GRAMPA are written in Algol 60 for the CDC 6400 computer. The package (from now on called the program) is divided into two main sections:



The structure and format of the tables will be discussed first, followed by explanation of the mechanisms used for reading in the grammar. Finally, each of the analysis and manipulation procedures will be described in detail.

PART I: GRAMMAR INPUT

2.2 Table Construction

The first section of the program reads the productions from cards and sets up the following tables:

1) a syntax graph,
2) a table of the terminal symbols and a corresponding cross-reference lookup table,
3) a table of the nonterminal symbols and a corresponding cross-reference lookup table,
4) tables to enable the left-hand symbol of a production to be obtained starting with the first symbol of the right hand side (for use in parsing).

The reader is referred to a listing of this first section of GRAMPA given in appendix 5.

2.3 The Syntax Graph

2.3.1 Description

Backus-Naur form (BNF) is a very useful way of displaying a grammar for human comprehension. However, other forms are needed to represent the grammar in the computer, which can be used for analyzing the structure of the grammar or for parsing. Cheatham and Sattley[3] have shown how to represent a grammar by a pair of tables

which yields a simple top-down parsing algorithm. The syntax graph used in GRAMPA represents a grammar by a graph which is a slight modification of Cohen and Gotlieb's[4] (which in turn is the equivalent of Cheatham's tables in list structure form). The syntax graph is not only a means whereby the structure of a grammar can be checked and manipulated, but it can be used for top-down parsing of the grammar, and in a reversed form for bottom-up parsing.

The following paragraph describing the construction of the syntax graph is taken from Cohen and Gotlieb[4].

"To construct the syntax graph for a context-free language, all productions starting with the same nonterminal are combined into one string, called the expression for the nonterminal; in this the OR symbol, |,separates the different alternatives. For example, the expression for the nonterminal Simple Arithmetic Expression (SAE) of Algol is written: SAE → TRM|AOP TRM|SAE AOP TRM, where AOP is the abbreviation of Addition Operator and TRM is the abbreviation for Term. Each expression is represented in the syntax graph by a tree, called the expression tree of the nonterminal. The set of all the expression trees forms a disjoint set of sub-graphs of the syntax graph.

These expression trees, together with connecting links, constitute the syntax graph."

Each node of the syntax graph is described by a quintuplet: its value, or name, which is the internal representation of the element in the vocabulary set of the grammar (either a terminal or nonterminal), and four pointers, pointing away from the node, and labeled DEFinition, ALTernative, SUCcessor and Left-Hand Symbol. The nodes within an expression tree are linked internally through the ALT and SUC links, and the complete syntax graph is formed by interconnecting the expression trees through the DEF links.

The rules for constructing the expression tree for the expression are as follows:

1) A node is created for each element on the right-hand side of the expression. The VALue of the node is the name or some internal representation of the element.

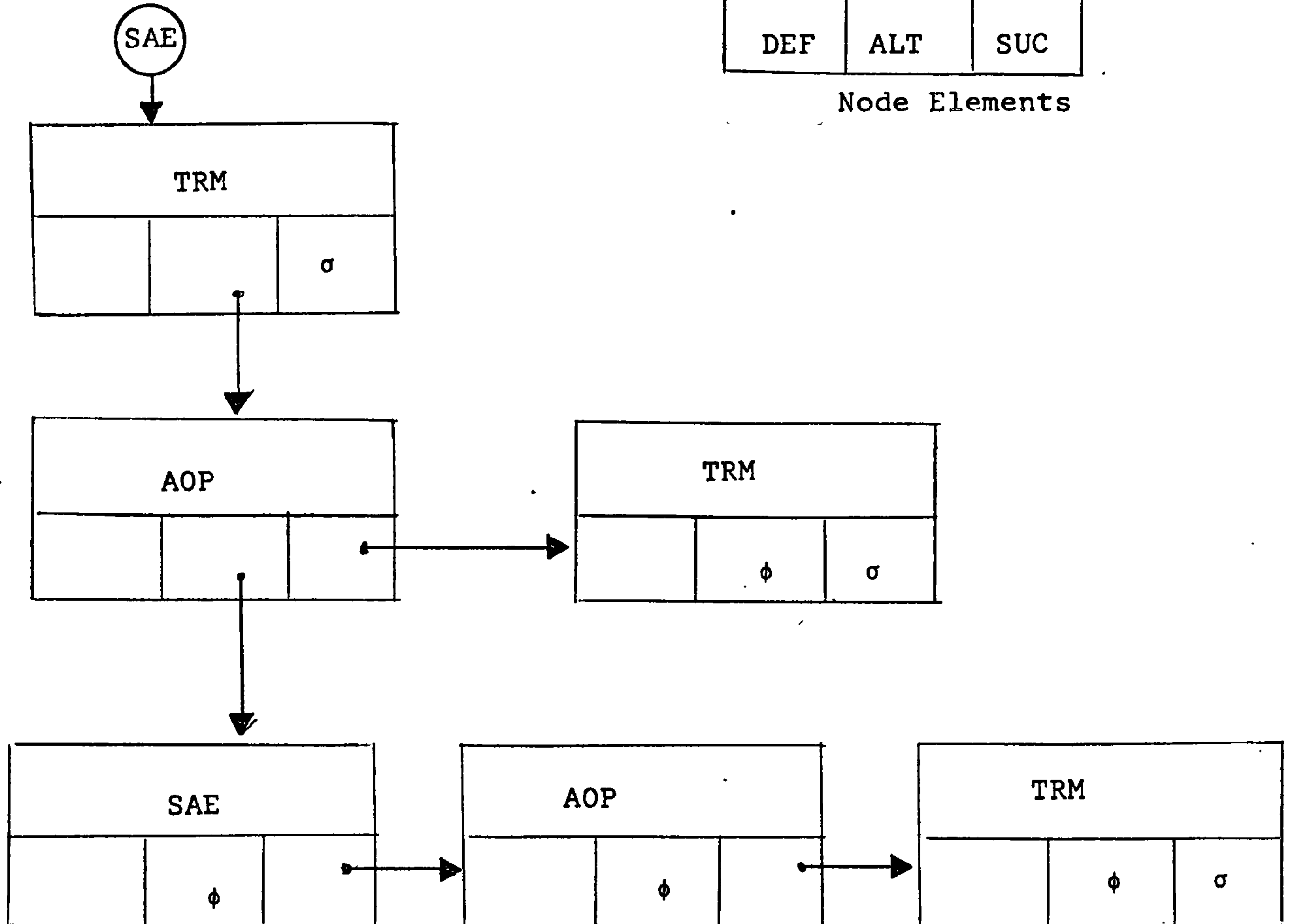
2) The elements of each production are linked through their SUC links in the order of occurrence in the production, from left to right. The last component of each production contains a special end-of-production symbol, σ, in its SUC link.

3) The different alternative productions in each expression, assumed to be in arbitrary order, are linked by means of the ALT link of the first element of each production in the expression.

4) A flag, bearing the name of the element on the left-hand side of the expression, is attached to the root (first node) of the tree.

In summary, the elements of productions are linked through their SUC links, with the symbol σ signalling the end of each production. A production is referenced by its first element, so that the different productions of an expression are connected through the ALT link of the first element of each. The end of any chain or list is signaled by a ϕ in the appropriate box. Figure 2.1 illustrates the expression tree for SAE.

Figure 2.1 Expression tree for the nonterminal Simple Arithmetic Expression, SAE
SAE→TRM/AOP TRM/SAE AOP TRM



To complete the construction of the syntax graph, the expression trees are connected by means of the DEF links. For a terminal node the symbol ϕ is inserted in the DEF field. For a nonterminal node the DEF link points to the root of the expression tree representing the

definition of the nonterminal. The construction of the syntax graph is completed by creating the root node of the graph. This node contains the grammar initial symbol, S, and it is linked to the expression tree defining it by its DEF link,while σ and ϕ are entered into its SUC and ALT fields.

All the information about the syntax of the language is now contained in the syntax graph. This syntax graph can, in general, be simplified. The form described so far is called the nonreduced form.

2.3.2 Representation

The syntax graph is represented in the program by the array SYNG (dimensioned [1:500, 1:5]). Each node of the graph is a row entry in SYNG. The first element of the row is a value for the symbol (assigned by GRAMPA), the second element is a pointer to the DEF row, the third is a pointer to the ALT row, and the fourth is a pointer to the SUC row. A fifth element exists which holds the value of the nonterminal symbol which is the left-hand side of the production. The meta-symbols, ϕ and σ, are both represented by 0. The end of a production is signalled when both SYNG [row, 3] and SYNG [row, 4] are zero.

For example, suppose the values 300, 301, and 302 had been assigned to the nonterminals SAE, TRM and AOP respectively, then the expression tree for Simple Arithmetic Expression would look as follows in the array SYNG:

SAE→TRM|AOP TRM|SAE AOP TRM

	Row	VAL	DEF	ALT	SUC	LHS
	1					
TRM	2	301	-	3	0	300
AOP	3	302	-	5	4	300
TRM	4	301	-	0	0	0
SAE	5	300	2	0	6	300
AOP	6	302	-	0	7	0
TRM	7	301	-	0	0	0
	8					

The blank boxes, holding the DEFinitions of AOP and TRM would be filled in later when those nonterminals are defined by expressions.

Before the programming mechanism is described for creating this table, it is necessary to describe how symbols are represented in the system.

2.4 Symbol Tables

Two tables are associated with both the terminal and nonterminal symbols of the grammar being processed. One table is used to store a copy of the symbol while the other is a cross-reference (hash) table to check whether the symbol has been met before while reading in, and to point to its position in the first table.

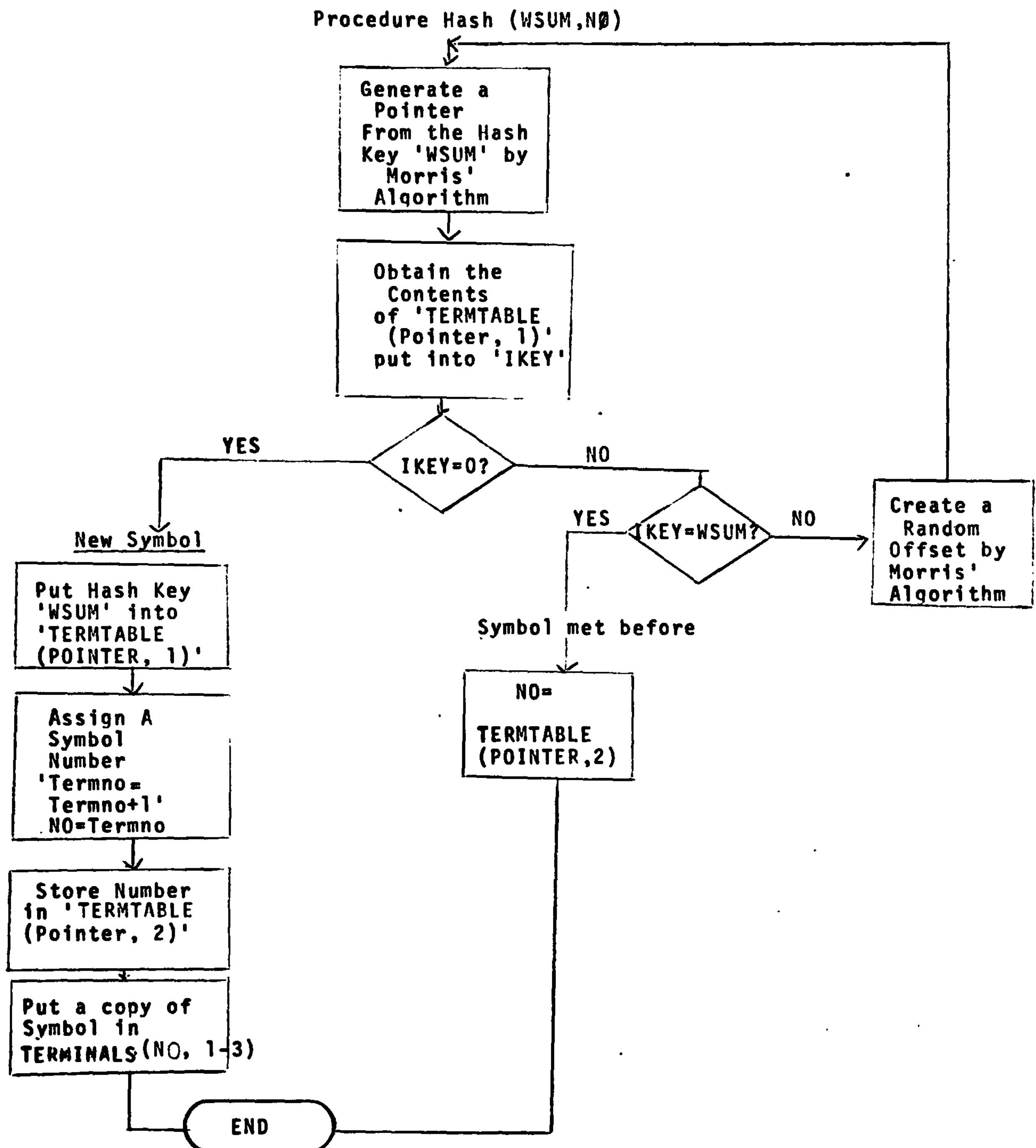
2.4.1 Terminal Symbols

Tables: TERMINALS [0:256, 1:3] ....store copy of symbol, and
TERMTABLE [1:256, 1:2] ....pointer table

As a terminal is read from cards, a copy of it is constructed in a three word array, WORD. The three words are then arithmetically added to form a hash key, WSUM, to check for previous occurrences of the symbol. The hash procedure used is the random probing technique of Morris[11], outlined in figure 2.2, (see also Appendix 3).

A pointer, POINTER, into the array TERMTABLE is first generated from the hash key. The contents of TERMTABLE [POINTER, 1] are compared to the key. If they are equal, then the symbol has been met before and is already stored at position TERMTABLE [POINTER,2] in the array TERMINALS. If the contents of TERMTABLE [POINTER, 1]

Figure 2.2 Outline of the Hashing Procedure for Terminal Symbols



are zero, then we have met the symbol for the first time. The hash key is stored in TERMTABLE [POINTER, 1] and a copy of the symbol is stored in TERMINALS with a pointer to it in TERMTABLE [POINTER, 2]. If the contents of TERMTABLE [POINTER, 1] are not zero and are not equal to the hash key, then we have a collision, and a random offset is created and added to POINTER (refer to figure 2.2). The procedure for checking is then repeated.

The terminal symbols of a grammar are numbered from 1 to 256 in GRAMPA. The value -1 is reserved for the empty word.

2.4.2 Nonterminal Symbols

Nonterminal symbols are handled in exactly the same way as terminals in GRAMPA. The two arrays used are:
NONTERMINALS [300:551, 1:4] for copies of the symbols, and
NONTERMTABLE [1:256, 1:2] for pointers and hash keys.

A fourth column is included in the array NONTERMINALS which is used during the construction of the syntax graph.

The system can easily determine whether a terminal or nonterminal symbol has been met by its position in context in a BNF expression.

2.5 Procedure INGRAMMAR

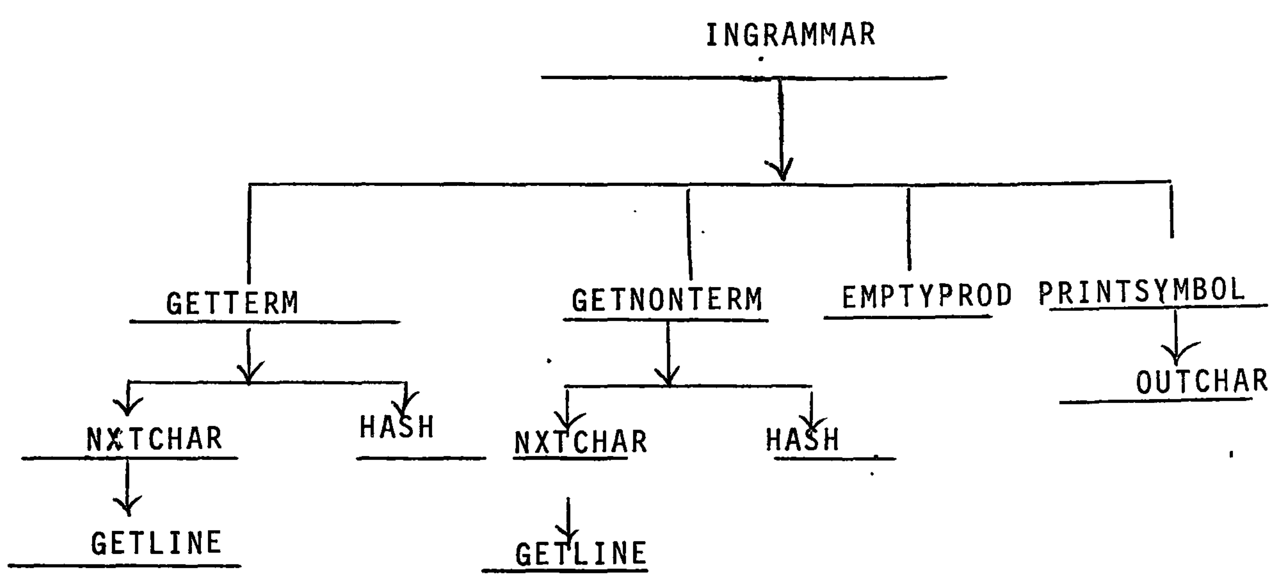
2.5.1 Introduction

The productions of a grammar are read into the program by the procedure INGRAMMAR. This single procedure also sets up all the tables described previously. The procedures called by INGRAMMAR are shown in figure 2.3. The reader is referred to the program listing for the detailed workings of these procedures. A brief summary is given below:

1) GETTERM: reads a terminal symbol, and puts it into the terminal tables.
2) GETNONTERM: Similar to GETTERM for the nonterminals,
3) EMPTYPROD: processes the empty statement when met by making appropriate flags in the syntax graph.
4) PRINTSYMBOL: prints a symbol, given its value.
5) HASH: Hashing routine to check for previous occurrence of a symbol or to put it into tables.
6) GETLINE: Reads an input card.
7) OUTCHAR: prints a character, given its internal value.

Before describing the logic used in the procedure INGRAMMAR, the characters and input conventions used with relation to 'inverse' BNF notation will be described.

Figure 2.3 Procedures Used by 'INGRAMMAR'



2.5.2 Characters Used and Their Internal Values

The characters allowed in a GRAMPA input deck are found in Table 1 together with their internal integer values assigned by the Algol system procedure IN CHARACTER.

2.5.3 'Inverse' BNF Notation

The inverse BNF notation[20] is essentially the same as BNF notation except that delimiters are placed around the *terminal* symbols instead of the nonterminals. For example, consider the following production for 'case sequence' in PL360[13]. <case seq>:= case< k reg> of begin| <case seq> <statement>

In GRAMPA this would be entered as:
case-seq→<case> k-reg <of> <begin>/case-seq statement; i.e. the terminals are delimited by the meta symbols <,>, and nonterminals are delimited by blanks - thus no blanks may occur *within* nonterminals. All nonterminal names longer than one word must be separated by a -(dash) symbol or concatenated, their total length should not exceed 13 characters. The meta symbols used in GRAMPA are listed below:

Symbols

<:>	delimit terminal symbols
/	delimit alternative right hand sides,
blank(s)	delimit nonterminals,

Table 1: Allowable Symbols and Their GRAMPA Values

Symbol	Value	Symbol	Value	Symbol	Value
0	1	L	22	.	43
1	2	M	23	'	44
2	3	N	24	↱	45
3	4	O	25	(blank)	46
4	5	P	26	*	47
5	6	Q	27	(	48
6	7	R	28	)	49
7	8	S	29	;	50
8	9	T	30	=	51
9	10	U	31	:	52
A	11	V	32	∧	53
B	12	W	33	∨	54
C	13	X	34	≤	55
D	14	Y	35	≥	56
E	15	Z	36	¬	57
F	16	-	37	≡	58
G	17	<	38	$	59
H	18	>	39	[	60
I	19	/	40	]	61
J	20	,	41	↑	62
K	21	+	42	↓	63

::= := or → "is defined by" -separator for left and right parts of a production.

; end of production

. end of grammar definition

\- used for concatenating nonterminal words.

When producing an input deck for GRAMPA, expressions may begin anywhere on a card and extend over any number of cards. However, terminal and nonterminal symbols may not be split over two cards. Examples of grammars punched on cards are given in chapter 3.

2.5.4 Logic of Procedure INGRAMMAR

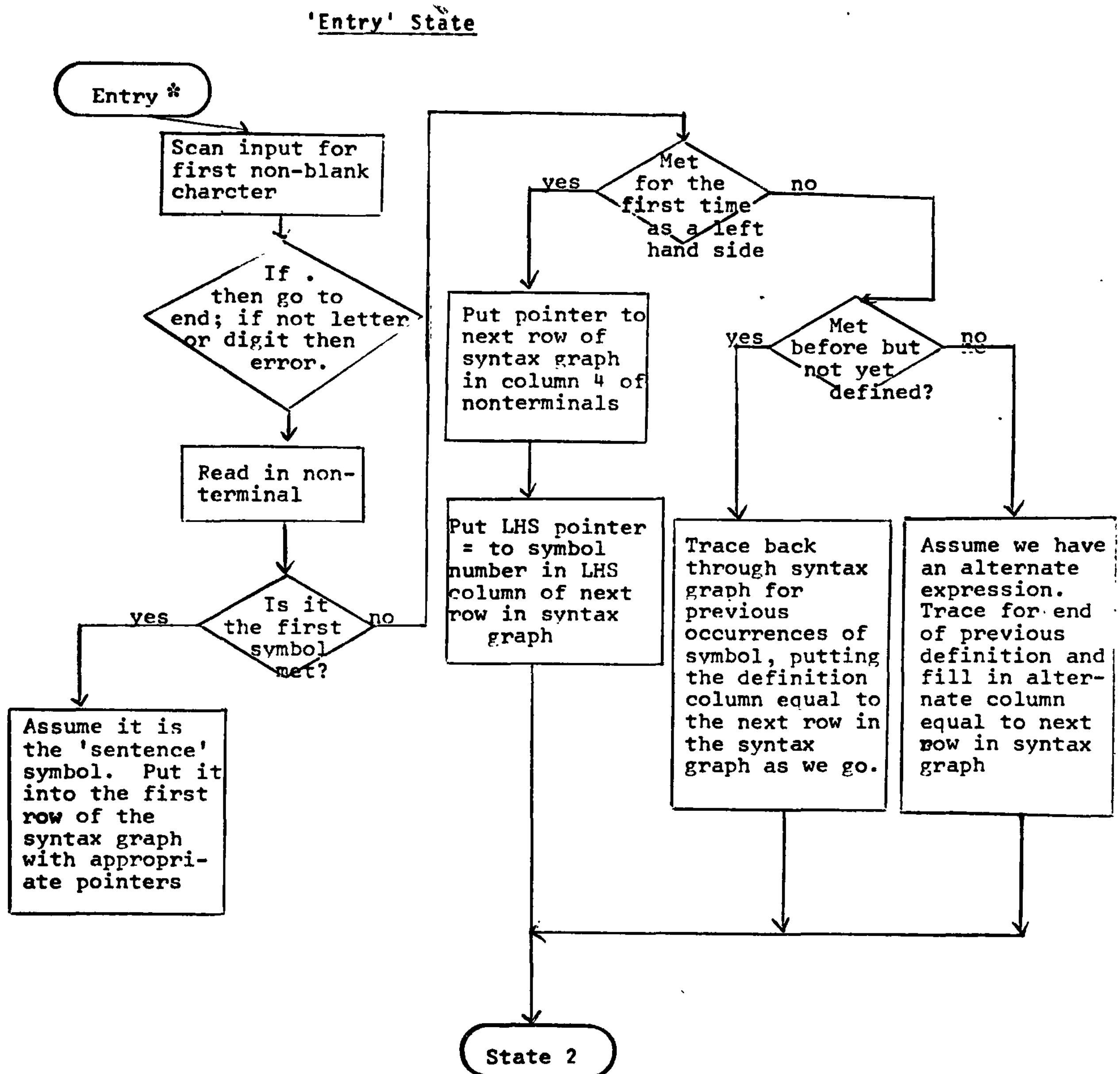
The procedure INGRAMMAR is basically divided into three parts:

1) initialize tables, counters, syntax graph,
2) read in grammar and set up tables, and
3) check for undefined nonterminals.

The first part of the procedure is fairly simple and involves mainly initialization of variables. The explanation of the working of the procedure will concentrate on the second part -- reading the grammar and constructing the tables.

The grammar is read in from cards using a very simple form of transition matrix. Three basic states are used:

Figure 2.4: Operation of 'INGRAMMAR'



*Entry, State 2 and State 3 are label identifiers in the program.

Figure 2.4 Continued Operation of 'INGRAMMAR'

State 2

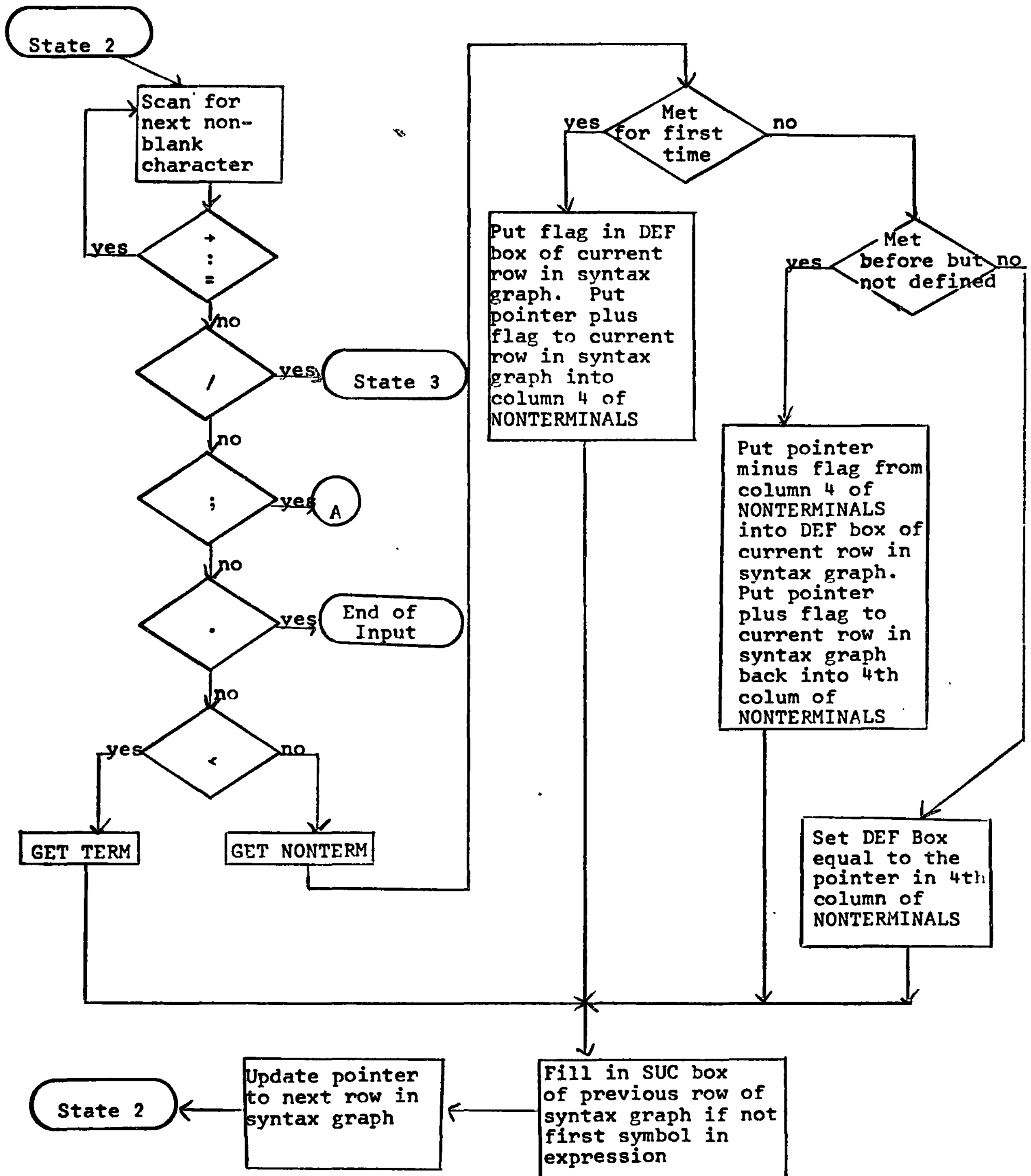
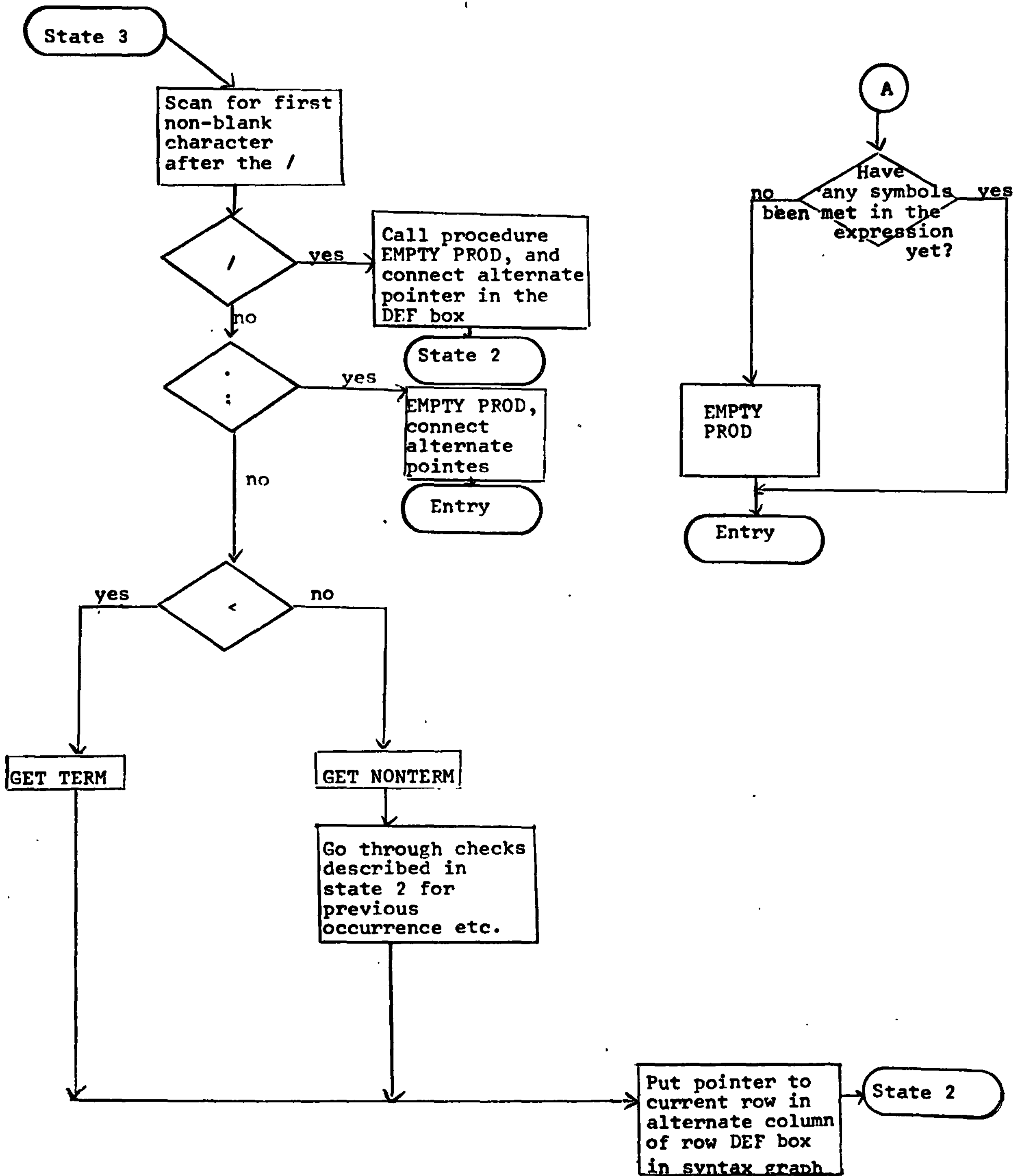


Figure 2.4 continued Operation of 'INGRAMMAR'
State 3



1) an ENTRY state for reading and processing the left-hand side of a production,
2) STATE 2 for handling the first complete expression, and any symbols after the first symbol after a / (alternate), and
3) STATE 3 for handling the first symbol after a /.

Flowcharts describing the operation of the three states are given in figure 2.4. The reader is also referred to the program listings. Key identifiers used in this portion of the program are explained below:

NO: value assigned to current symbol,

NXTBOX: next free row in syntax graph,

DEFBOX: row which contains the first symbol of current expression,

LHSBOX: value of nonterminal which is the left hand side of the current expression being processed,

RHSNO: counter for position of current symbol in expression,

PRODUCTIONS: count of the number of production met so far.

The detailed workings in each state will not be described, however, the pointer system maintained in the fourth column of the array NONTERMINALS is of interest. This pointer is used to tell us whether a nonterminal has been met before; if so, then whether it has been defined

or not. It must be remembered that the sentence symbol is the only nonterminal that goes straight into the syntax graph as a left-hand side, all the other nonterminals met as left-hand sides are not put into the syntax graph since they have occurred or will occur somewhere in a right-hand side.

If a nonterminal (not the sentence symbol) is met for the first time as a left hand side, then the position of the next free row in the syntax graph is put into NONTERMINALS [NO, 4], since this row will hold the first symbol of the first right-hand side expression. However, it is more likely that we meet a nonterminal within an expression and it has not been defined before. When this occurs, the position of the current row in the syntax graph holding the nonterminal (now in an expression) is stored in NONTERMINALS [NO ,4] added to the number 10,000 which serves as a flag. A flag of -1 is put into the definition box of the nonterminal's entry in the syntax graph. We may continue to meet the same nonterminal symbol in subsequent expressions until it is defined. In these cases, the pointer in NONTERMINALS [NO ,4] is updated to the current row in the syntax graph, while the definition box in the current row is set to the row holding the previous entry of the nonterminal. Thus a linked list is formed backwards through the syntax graph joining all occurrences

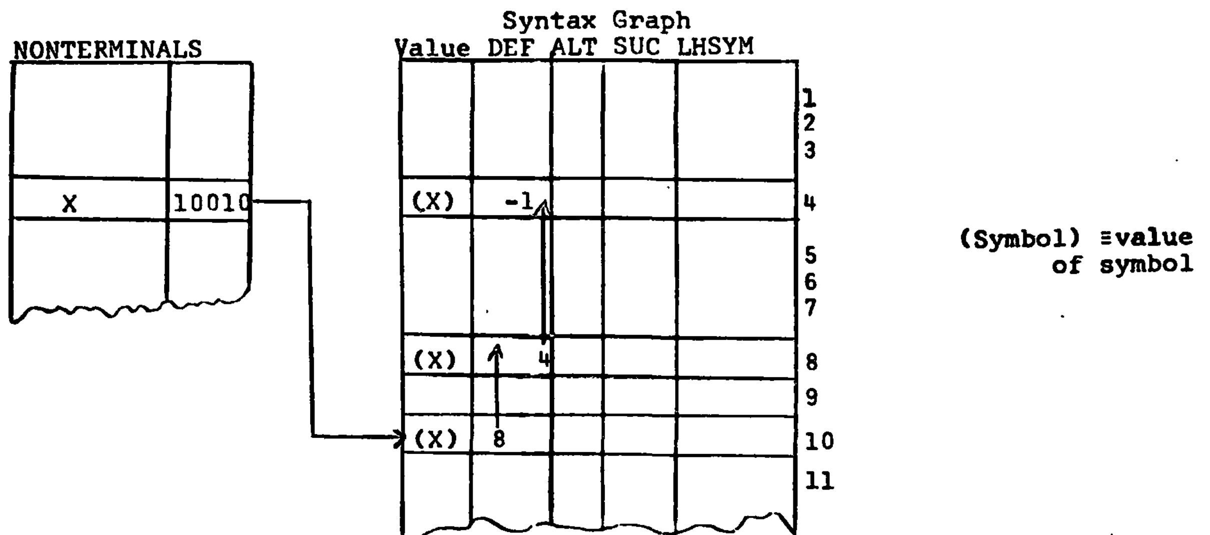
of the nonterminal. The head of the list is given in NONTERMINALS [NO, 4] and the tail is signified by -1 in the definition box of the row in the syntax graph holding the first occurrence of the nonterminal. When the symbol is finally met as a left-hand side, this list is traced through, and a pointer is inserted to the current row in the syntax graph holding the first symbol of the nonterminal's right-hand side. Figure 2.5 illustrates this mechanism for an arbitrary nonterminal, X.

The third way we can encounter a nonterminal is when it is a left-hand side but it has already been defined. In this case we have an alternate expression. The pointer to the head of the first right hand side for the nonterminal is retrieved from NONTERMINALS [NO,4]. Then the list of alternates (if any) for the nonterminal are traced through via the alternate pointers until a zero alternate pointer is met. The value of the next free row in the syntax graph is then inserted here and the right-hand side for the current occurrence of the nonterminal is processed as normal.

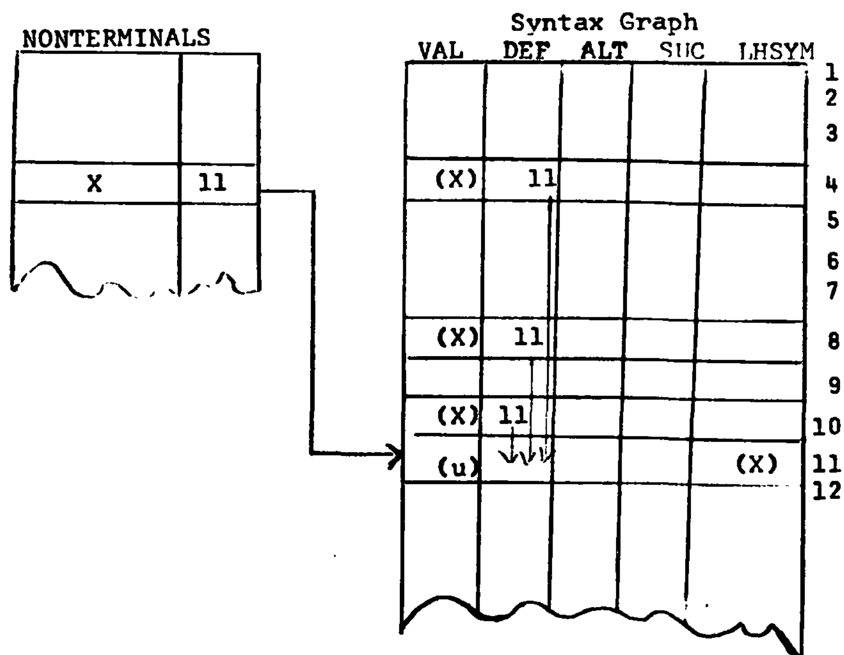
When all of the productions in the grammar have been processed, a scan is made of NONTERMINALS [NO, 4] to check for any values over 10,000 i.e. undefined nonterminals. If these are encountered, an appropriate

Figure 2.5: Illustration of Backward List Kept in the Syntax Graph

a) Nonterminal X met within expressions but not yet on a left hand side.



b) Nonterminal X now met as the left hand side of an expression X →u.



message is printed and the list through the syntax graph is cleared.

This list procedure was set up to enable the productions in the grammar to be entered in any order. However, the GRAMPA system expects the sentence or distinguished symbol to be the first nonterminal encountered.

2.5.5 The EMPTY Statement: (Procedure EMPTYPROD)

The empty statement is handled as if it were a terminal symbol, however, several flags are set to denote its occurrence. A value of -1 is assigned the symbol and is placed in the value box of the current row in the syntax graph, also the number 1 is put in TERMINALS [0,1] to signify its occurrence for printing purposes. Finally, the pointer in NONTERMINALS to the beginning of the expression containing the empty statement is negated.

2.5.6 Procedures GETTERM and GETNONTERM

These procedures read in a terminal or nonterminal symbol, form the hash key and call the procedure HASH to insert the symbol into tables. The operation of the procedures can be understood with reference to the program listing. The procedure GETTERM makes a special check for

the case when the meta symbol > is in fact a terminal symbol of the grammar, i.e. when the symbols <>> occur on a card.

2.6 Procedure PRINT TABLES

This procedure prints a neat copy of the symbols with their GRAMPA values, and prints the syntax graph. It can be called anywhere in GRAMPA after INGRAMMAR.

2.7 Procedure PRINTGRAMMAR

The procedure PRINTGRAMMAR is used to print a tidy version of the grammar after processing by INGRAMMAR. This is accomplished by stepping through the pointers in NONTERMINALS [NO, 4], and tracing the ensuing productions. If a nonterminal is undefined, an appropriate message is printed. This procedure is separate from INGRAMMAR and is very useful for locating input errors since the procedure prints the grammar as it was understood by INGRAMMAR. The procedure can be called anywhere in the GRAMPA system after INGRAMMAR.

PART II: GRAMMAR ANALYSIS AND MANIPULATION PROCEDURES

The second part of this chapter will describe the algorithms and procedures developed so far for GRAMPA which can be used for analyzing and manipulating context free grammars.

2.8 Admissibility of Grammars

The algorithms and examples given below which provide mechanical tests for the admissibility of a grammar are taken directly from Wood.[17]

2.8.1 Reachability Test

Let G = (V_N, V_T, S, P) be a grammar

Step 1: Let R={S} = L be two sets.

Step 2: Choose an X in L. Let L = L-{X}. For all Y in V such that X→uYv in P, for some u, v in V*, if Y not in R then (let R=R∪{Y} and if Y in V_N then let L= L ∪ {Y}).

Step 3: If L ≠ ϕ repeat steps 2 and 3, otherwise R holds the reachable symbols of G.

Now R is best represented as a table and L as a list. Consider the example grammar in section 1.3:

S→ aX|aYb

X→ aX|aaX

Y→ aYb|ε

Z→ aZ|b

Now:

R =

S	Y
X	N
Y	N
Z	N
a	N
b	N

and L = [S] initially. Y(=Yes) indicates that the adjoining symbol would be in the set R in Algorithm 1.

Step 2 gives

R =

S	Y
X	Y
Y	Y
Z	N
a	Y
b	Y

and L = [X,Y] since $S \rightarrow aX$ and $S \rightarrow aYb$.

as $L \neq \phi$ repeat steps 2 and 3 obtaining

R =

S	Y
X	Y
Y	Y
Z	N
a	Y
b	Y

and L = [Y], where X in L is chosen.

Again as $L \neq \phi$ we repeat steps 2 and 3, which give

R =

S	Y
X	Y
Y	Y
Z	N
a	Y
b	Y

and L = [ ].

Thus L = ϕ and R = (S,X,Y,a,b) is the set of reachable symbols of G, i.e. Z is unreachable.

We now turn to our second algorithm.

2.8.2 Usefulness Test

Let G = (V_N, V_T, S, P) be a grammar.

Step 1: Let U_0 = ϕ and i=0.

Step 2: Let U_{i+1} ={X: X→x in P, x in ($U_i \cup V_T$*}.

Step 3: Let i = i+1. If $U_i \neq U_{i-1}$ then repeat steps 2 and 3 otherwise U_i contains the useful nonterminals.

The first time step 2 is obeyed U_1 contains those nonterminals which have at least one associated rule whose right side is a terminal word. Therefore, for all X in U_1, X is useful. The second time step 2 is obeyed U_2 will consist of those nonterminals which have at least one associated rule whose right side is a word from $(V_T \cup U_1)^*$. Again such a nonterminal will be useful since

all members of U_1 are useful. The algorithm must halt since V_N is finite.

Using the example grammar (section 1.8):

Step 1: $U_0 = \phi$, $i = 0$.

Step 2: $U_1 = \{Y,Z\}$

Step 3: $i = 1$, as $U_1 \neq U_0$ repeat steps 2 and 3.

Step 2: $U_2 = \{S, Y, Z\}$

Step 3: $i = 2$, as $U_2 \neq U_1$ repeat step 2 and 3.

Step 2: $U_3 = \{S, Y, Z\}$

Step 3: $i = 3$, as $U_3 = U_2$ then U_3 is the set of useful nonterminals, therefore, X is a useless nonterminal.

Given any CFG,G, and using algorithms 2.8.1 and 2.8.2, there in fact exists an equivalent CFG G^1 such that G^1 is admissable.

2.8.3 Procedure ADMISSABLE

Tests for admissability are performed by the procedure ADMISSABLE in GRAMPA. This procedure first calls a procedure TERMINATE which constructs a Boolean table T to indicate which nonterminals lead to a terminal symbol. There is an entry in T for each nonterminal; T(nonterminal) is TRUE if the symbol is useful otherwise it is FALSE. The procedure TERMINATE uses the usefulness

algorithm of section 2.8.2.

When the useless nonterminals have been found, a procedure DELETE is called which deletes the productions containing the nonterminal from the grammar. Also the pointer in NONTERMINALS [useless nonterm, 4] is set to zero. The grammar is printed.

The checks for reachability are then performed within ADMISSABLE itself. The algorithm of section 2.8.1 is used. The Boolean array T represents the table R of section 2.8.1, and the array PD represents the set L. After the reachability test, the unreachable nonterminals are deleted and the grammar is printed. Symbols deleted for uselessness by the procedure TERMINATE also become unreachable because of the deletions. The admissibility procedures were initially written by Dr. D. Wood and modified slightly by the author.

2.9 Algorithms and Procedures for Simple Precedence Analysis

2.9.1 Check for the Empty Statement

Before any analysis for precedence is performed, a check is made by the procedure EMPTYCHECK for the existence of the empty statement. This is accomplished quite simply by scanning the 4th column of the array NONTERMINALS discussed in section 2.5.5. If any element

in the 4th column of this array is negative, then this is flag for the existence of the empty statement in the right hand side of the production. If this is encountered, an appropriate message is printed, and control jumps over all precedence procedures.

2.9.2 Left and Right Sets

The definition of the sets $\mathcal{L}(U)$ and $\mathcal{R}(U)$ in section 1.4 is such that an algorithm for generating the sets is evident. A symbol Y is a member of $\mathcal{L}(U)$, if (i) there exists a syntactic rule p : $U \to Yx$, for some x, or (ii) there exists a syntactic rule p : $U \to U_1 x$ and $Y \varepsilon \mathcal{L}(U_1)$; i.e. $\mathcal{L}(U)$ =

$$\{Y \mid \exists\ p:U \to Yx \vee \exists\ p:U \to U_1 x \wedge Y \varepsilon \mathcal{L}(U_1)\}.$$

Analogously:

$$\mathcal{R}(U) = \{Y \mid \exists p:U \to xY \vee \exists\ p:U \to xU_1 \wedge Y \varepsilon \mathcal{R}(U_1)\}.$$

The algorithm for finding $\mathcal{L}(U)$ and $\mathcal{R}(U)$ for all symbols $U \varepsilon V_N$ involves searching the list of productions P(p) for appropriate syntactic rules.

In GRAMPA, the left and right sets are found by the procedure LEFTRIGHT, which in turn uses the procedures LDEF, RDEF, and PATHCHK.

In the case of the left sets the procedure scans each nonterminal successively. The production generated

by a nonterminal starts at the row given in NONTERMINALS [i,4]. The recursive procedure LDEF is then called. This procedure traces through all the right sides for the nonterminal and puts the left-most symbols of the productions into row i of a two-dimensional array LR. If a nonterminal is one of these symbols, then LDEF recursively calls itself to trace through all of its productions. The procedure PATHCHK is used to ensure that we do not cycle. For example consider the productions:

$$X \rightarrow H$$
$$H \rightarrow Ha \mid b$$

The left part of X is H and b. While checking H we could continually cycle as H is in its own left set. The right sets are generated in a similar way using the recursive procedure RDEF.

When all of the nonterminals have been scanned, the left and right parts are printed by the procedure LEFTRIGHT.

2.9.3 Generating the Precedence Matrix

The simple precedence matrix can be generated after the left and right sets have been found using the

procedure WWPRECEDENCE. The algorithm used is that of Wirth-Weber.[14]

The precedence relations can be represented by matrix M with elements M_{ij} representing the relation between the ordered pair (X_i, X_j). The matrix has as many rows and columns as there are symbols in the vocabulary $V=V_N \cup V_T$.

Assuming that an arbitrary ordering of symbols of V has been made ($V=\{X_1, X_2, \ldots X_n\}$), the algorithm for the determination of the precedence matrix M is as follows: For every element $p \varepsilon P$ which is of the form

$$p: U \rightarrow X_1 X_2 \ldots X_m,$$

and for every pair X_i, X_{i+1} $(i=1,\ldots,m-1)$ assign:

a) $\doteq$ to $M_{i,i+1}$,

b) $<\cdot$ to all $M_{i,k}$ with column index k such that $X_k \varepsilon \mathcal{L}(X_{i+1})$,

c) $\cdot>$ to all $M_{k,i+1}$ with row index k such that $X_k \varepsilon \mathcal{R}(X_i)$,

d) $\cdot>$ to all $M_{k,\ell}$ with indices k,ℓ such that $X_k \varepsilon \mathcal{R}(X_i)$ and $X_\ell \varepsilon \mathcal{L}(X_{i+1})$.

Assignments under rule (b) occur only if $X_{i+1} \varepsilon V_N$, under (c) only if $X_i \varepsilon V_N$, and under (d) only if both X_i, $X_{i+1} \varepsilon V_N$, because $\mathcal{L}(X)$ and $\mathcal{R}(X)$ are empty

sets for all $X \varepsilon V_T$.

The procedure WWPRECEDENCE simply follows these four rules in order. For example in rule b, each entry is checked in the syntax graph, if the symbol has a successor that is a nonterminal symbol, then the relation <· holds between the orginal symbol and each symbol from the right part of the successor symbol.

The precedence matrix is stored in the array PMATRIX, and the precedence relations are represented by the following values:

Value	Precedence Symbol	Printed Symbol
1	no relation	(blank)
2	<·	<
3	·>	>
4	≐	=
5-9	conflict	C

When the precedence matrix has been determined, it is printed out with a maximum of 30 symbols across the top of a page. Thus several pages may be needed to print out the whole matrix.

2.9.4 Handling Precedence Conflicts

The procedure WWPRECEDENCE uses a procedure PUTM to place the precedence symbols in the matrix, PMATRIX. This procedure also checks for precedence conflicts, and maintains a list of these in the array CONFLICTABLE (which is global to the second part of the program, i.e. all precedence procedures), and a count in NCONFL. The array CONFLICTABLE is dimensioned [1:20, 1:5] i.e. there is room for 20 conflicts. The first column of the array holds a code for the conflict:

CODE	CONFLICT
5	$\lessdot$, $\gtrdot$
6	$\lessdot$, $\doteq$
7	$\gtrdot$, $\doteq$
9	$\lessdot$, $\doteq$, $\gtrdot$

The code is simply the sum of the GRAMPA values for the symbols given at the end of the preceeding section 2.7.3. The second column holds the left symbol of the conflict, and the third column holds the right symbol. The fourth and fifth columns hold the indices of the conflict in PMATRIX for bookkeeping purposes. This array is printed at the end of WWPRECEDENCE if it is nonempty.

After WWPRECEDENCE, the procedure REMOVE CONFLICTS can be called to remove any conflicts using the rules given in section 1.6, i.e. by using left and right restricted expansions. The procedure REMOVE CONFLICTS steps through the array CONFLICTABLE; if a type 6 conflict is met $(<,\doteq)$ then a procedure LRE(CONFLICTABLE [I,3]) is called, otherwise a procedure RRE (CONFLICTABLE [I,2]) is called. These two procedures produce left (right) restricted expansions according to the definitions in section 1.6.

For example, for LRE's, a new nonterminal is artificially produced named XXXXi (i=0-9) and is added to NONTERMINALS. A production XXXXi→A, where A is the argument of LRE, is added at the end of the syntax graph. All productions are then scanned and all occurrences of the old nonterminal A are replaced by XXXXi except where A is the left-most symbol of a right side. The procedure RRE is more or less the same except it does not replace A when it is the right-most symbol of a right side.

After the initial scan of CONFLICTABLE and the ensuing expansions, REMOVE CONFLICTS calls LEFTRIGHT and WWPRECEDENCE again, and the cycle is repeated. The procedure terminates when all conflicts have been removed

i.e. NCONFL=0.

2.9.5 Precedence Functions

Many algorithms exist in the literature for calculating precedence functions[1][2][9][15]. GRAMPA uses Wirth's algorithm[15], and the procedure PFUNCTIONS is a direct copy of the procedure given as Algorithm 265, C.ACM 8 (10), Oct. 1965 pp. 604-605. If the precedence functions exist, the procedure will produce a neat list of the symbols with the function values.

2.9.6 Parsing Using Simple Precedence

The process of parsing is very straightforward using precedence relations. In accordance with the definition of the canonical parse, a parsing algorithm must first detect the leftmost phrase of the sentence to which a reduction is applicable, i.e. the leftmost simple phrase. The reduction is then performed and the same principle is applied to the new sentence. The process of detecting the leftmost reducible phrase, after precedence relations have been determined, consists of scanning the sentence from left to right until the first symbol pair is found so that $X_i \cdot> X_{i+1}$, then to retreat back to the last symbol pair for which $X_{j-1} <\cdot X_j$ holds. $X_j \ldots X_i$ is the sought substring or phrase; it is replaced by the

symbol resulting from the reduction. The process is then repeated.

A description of an algorithm given in Wirth-Weber[14] is given below (in pseudo Pascal). The original sentence is denoted by $P_1 \ldots\ldots P_n$. k is the index of the last symbol scanned. All scanned symbols are copied and renamed $X_1 \ldots\ldots X_i$. The reducible substring therefore, will always be $X_j \ldots\ldots X_i$ for some j. Internal to the algorithm, there exists a symbol $\perp$ called an endmarker, initializing and terminating the process. To any symbol $X \varepsilon V$ it has the relations $\perp <\cdot X_1$ and $X_n \cdot> \perp$. Assume $P_0 = P_{n+1} = \perp$.

$X_0 := P_0$; $j := 0$; $k := 1$;
while $P_k \neq \perp$ *do*
begin $i := j := j+1$; $X_j := P_k$; $k := k+1$;
    while $X_i \cdot> P_k$ *do*
    begin while $X_{j-1} \doteq X_j$ *do* $j := j-1$;
    X_j = Left part $(X_j \ldots\ldots X_i)$; $i := j$
    end
end

A version of this algorithm allowing a heuristic form of error recovery can be found in Wirth.[13]

The heart of this algorithm is the procedure Left part ($X_j \ldots X_i$) which has to identify the reducible phrase to obtain the symbol resulting from the reduction. (If furthermore the parsed sentence is to be translated, then the semantic rule corresponding to the syntactic rule $U \rightarrow X_j \ldots\ldots X_i$ should be identified and executed).

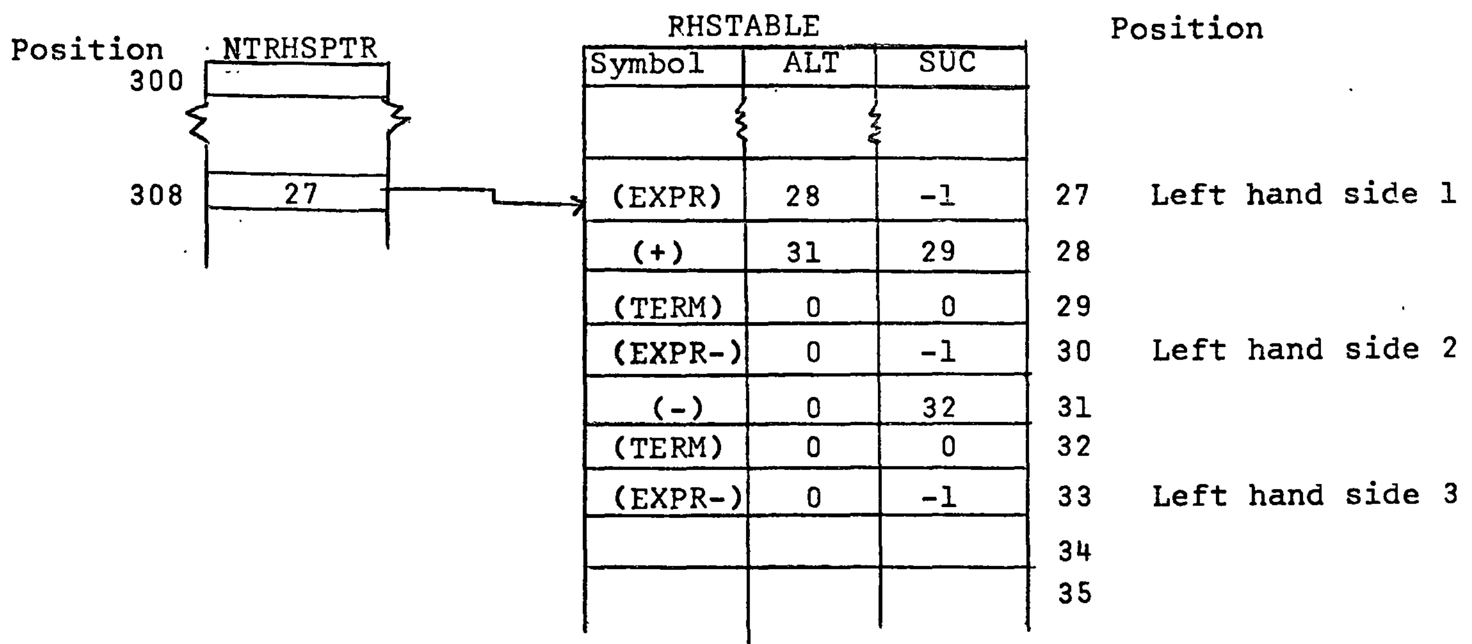
A procedure TRANSPOSE SYNTAX GRAPH has been written for GRAMPA which produces tables to facilitate the writing of a procedure similar to Left part. The procedure produces three tables - two pointer tables which use the first symbol X_j of the phrase to be reduced as an index, and a "right-hand-side" table. The index tables, called TRHSPTR (terminal right-hand-side pointer) and NTRHSPTR are single arrays. The symbol X_j is an index to a position in the table which contains a pointer into a larger table, RHSTABLE, which holds the rest of the production $X_{j+1} \ldots\ldots X_i$ in a list structure form (similar to the syntax graph). The table TRHSPTR is used if a terminal symbol starts the phrase (internal GRAMPA value 1-256), and NTRSPTR is used if a nonterminal symbol starts the phrase (value 300-557). The table, RHSTABLE, has three columns. The first holds the symbol value, the second an alternate pointer, and the third a successor pointer.

For example, consider the following productions from Wirth.[14]

EXPR ::= EXPR-

EXPR- ::= EXPR- <+> TERM | EXPR- <-> TERM | TERM,

and suppose the nonterminal EXPR has the value 307 and EXPR- = 308 in GRAMPA, then the following entries might be made in NTRHSPTR and RHSTABLE:



where (symbol) would be the GRAMPA value for the symbol. If the symbol EXPR- is encountered as the left symbol of a phrase, then the pointer is obtained from NTRHSPTR into RHSTABLE. Any entry in RHSTABLE flagged with a -1 means that it is a right hand side of a production. Thus immediately, EXPR is a right hand side. However, if

the phrase is longer than just EXPR-, then the alternate list starting at row 28 is scanned - namely <+> TERM. If this is matched, then the next row gives the right hand side i.e. EXPR-. If it is not matched, then the alternate starting at row 31 (indicated in the row <+>) is scanned etc. We thus have a list structure of alternates and successors very similar to the syntax graph.

2.10 Checks for Recursion

The concept of left, right and imbedded recursion was presented in section 1.4.2. The procedure RECURSIVECHECK produces lists of all three types of recursive symbols. The procedure uses three smaller procedures, LEFTRECURSIVE, RIGHTRECURSIVE and IMBED to locate the recursive symbols. The left and right recursive symbols are found by checking for the occurrence of a particular nonterminal within its own left or right parts. The imbedded recursion checks are somewhat more complicated and rely on recursive calls of the Boolean procedure IMBED. This procedure first checks whether a particular nonterminal, N, is imbedded in one of its own productions. If not, then the nonterminal, M^L, on the left of the production is checked--N can be imbedded or on the right of the production generated by M^L. Similarly

for a nonterminal M^R on the right of the production generated by N, N can occur imbedded or on the left of any production generated by M^R. These checks can go for many levels of recursion until a true value is returned or the whole procedure exits because it is looping (an array is maintained which holds the symbols checked by IMBED to prevent looping).

CHAPTER 3

USE OF GRAMPA WITH EXAMPLES

3.1 Programming Considerations

3.1.1 Overall Structure of GRAMPA

The overall structure of GRAMPA is shown in Figure 3.1. The first set of procedures reads in the grammar and sets up the tables, therefore, they must be included in any GRAMPA run. However, the procedures PRINTGRAMMAR and PRINT TABLES are optional. The whole second block of the program containing the precedence analysis procedures is optional, and can be omitted until the user is satisfied that he has the grammar punched correctly. In the second section, the procedures EMPTYCHECK and LEFTRIGHT must be included before WWPRECEDENCE and RECURSIVECHECK can be used, because both of the procedures use the left and right sets and assume an ε-free grammar.

The structure of any Algol program is flexible, the user can insert his own procedures easily. However, before attempting this, the user should familiarize himself with the global variables - described in Appendix 5.

Figure 3.1: Overall Structure of GRAMPA

```
begin
      -define tables and working variables for grammar input
      -procedures:  GET 4        \
                    PUT 4         |
                    GET 9         |
                    PUT 9         |
                    IABS          |
                    OUTCHAR       |
                    GETLINE       |
                    PRINTSYMBOL   |
                    NXCHAR         >      must always
                    GETNONTERM    |       be included
                    GETTERM       |
                    HASH          |
                    EMPTYPROD     |
                    PRINTABLES    |
                    INGRAMMAR    /
                    PRINTGRAMMAR \
                    TERMINATE     >       optional
                    DELETE        |
                    ADMISSIBLE   /
      (i) main program:  INGRAMMAR; PRINTGRAMMAR; PRINT TABLES;
                         ADMISSIBLE;

      begin
      -define tables for left and right parts and precedence matrix
      -procedures:  EMPTYCHECK    \
                    LEFTRIGHT      |                          \
                    proc:  RDEF    |     must be included if   |
                           LDEF     >    WWPRECEDENCE and/or   |
                           PATHCHK |     RECURSIVECHECK are    |
                                  /      used.                 |
                    WWPRECEDENCE                               |
                           PUTM                                |
                    REMOVECONFLICTS                             >optional
                           RRE                                 |
                           LRE                                 |
                    PFUNCTIONS                                 |
                    TRANSPOSE SYNTAX GRAPH                     |
                    RECURSIVECHECK                             |
                           LEFTRECURSIVE                       |
                           RIGHTRECURSIVE                      |
                           IMBED                              /
      (ii) main program:  EMPTYCHECK; LEFTRIGHT; WWPRECEDENCE;
                          REMOVECONFLICTS; PFUNCTIONS; TRANSPOSE
                          SYNTAX GRAPH; RECURSIVE CHECK
      end
end
```

3.1.2 The Program Deck

GRAMPA at time of writing is not stored on disc or tape on the CDC 6400 computer. Thus a user must obtain a copy of the deck in card form, (2100 cards).

3.1.3 Storage Requirements

The first major program block requires 35.7_8K words of storage to load,and the whole program requires 55_8K words to load. However, it must be remembered that many of the arrays used in the second major block are dynamic, and their size depends upon the grammar being processed. The largest arrays are LR(for the left and right sets), PMATRIX(for the precedence matrix) and RHSTABLE (generated by TRANSEPOSE SYNTAX GRAPH for left side lookup).

Compacting techniques are used in GRAMPA to reduce the storage requirements for these arrays. In the case of the array LR, five entries each of size nine bits are made into each word. Upon entry to the second block, the array is dimensioned:

[300:NTERMNO +20, 2x(SIZE+19)// 5+1)],

where SIZE is the size of the vocabulary, and NTERMNO-300 is the number of non-terminals. SIZE can be altered as

artificial nonterminals are created to remove precedence conflicts, hence an allowance for 20 extra symbols. Entries are made into the array by the procedure PUT9. For example, to place an item x in row i, column j of LR, the following statement is used:

PUT 9(LR,i,j,x).

Items are retrieved by the integer procedure GET 9. e.g. x= GET 9(LR,i,j).

The precedence matrix, PMATRIX, is compacted in a similar way to LR. Upon entry to the second major program block, the array is dimensioned:

[1:SIZE+20,1:(SIZE+19)// 12+1].

Twelve items of size four bits are packed into each word. The procedures PUT 4 and GET 4 are used to place and retrieve items.

The array RHSTABLE in TRANSPOSE SYNTAX GRAPH is dimensioned [1:NGRAPH+10, 1:3], where NGRAPH is the number of entries in the syntax graph. Thus the array will never exceed 1530 words in size.

When analyzing grammars, estimates should be made of the size of the arrays before using the program. In the case of small grammars (20 productions and symbols) about 60_8K of storage is sufficient. Larger grammars

of around 100 productions and symbols need over 100_8K of storage.

3.1.4 Run Time

Compilation time for the complete program is about 29 seconds, and about 1 1/2 seconds are needed for loading. A central processor time algorithm for the first major program block is given in Appendix 4. The execution time for the second block is difficult to estimate because it depends on the structure as well as the size of the grammar. Total run times are given in the next sections for the example grammars, and are summarized in table form in Appendix 4.

3.1.5 The Data Deck

The grammar to be analysed is punched in free format using the meta-symbols described in section 2.5.3. The first data card must be a title card for the grammar (1 to 80 characters). The end of data is a . (period) on the last card.

3.2 Example 1: Simple Phrase Structure Language

3.2.1 Correct Grammar

The first example is the simple phrase structure language defined in Wirth-Weber[14]. The punched deck (as listed by the program) is given below.

```
*INPUT*

BLOCK        →   <BEGIN> BODY <END> ;
BODY         →   BODY- ;
BODY-        →   DECL <;> BODY- /
                 STATLIST ;
STATLIST     →   STATLIST <,> STATEMENT /
                 STATEMENT ;
STATEMENT    →   VAR <:=> EXPR /
                 BLOCK ;
EXPR         →   EXPR- ;
EXPR-        →   EXPR- <+> TERM /
                 EXPR- <-> TERM /
                 <-> TERM / TERM ;
TERM         →   TERM- ;
TERM-        →   TERM- <*> FACTOR / TERM- </> FACTOR / FACTOR ;
FACTOR       →   VAR /
                 <(> EXPR <)> /
                 NUMBER ;
VAR          →   <L> ;
NUMBER       →   DIGIT / NUMBER DIGIT ;
DECL         →   <NEW> <L> ;
DIGIT        →   <D> .
```

The first part of the output is shown in Exhibit 3.1, the remainder is given in Appendix I. The total computer run time, including compilation, was 46.3 seconds.

Exhibit 3.1 is a neat listing of the grammar <u>as understood by GRAMPA</u>. If any nonterminals are found to be undefined, the message "(NO RIGHT SIDE DEFINED IN THE GRAMMAR)" is printed as the right side. Every alternate right side of a production starts on a separate line, and a line is skipped between productions.

The second section of output (shown in Appendix I) is the symbol tables and syntax graph. The GRAMPA value assigned to the symbol is also given. Seven columns of output appear for the syntax graph. The first is the row number, followed by the five columns of the syntax graph itself, described in section 2.3.1, then the name of the symbol in the row is given.

The output from the procedure LEFTRIGHT is next shown. The symbol numbers only are given in the listings of the sets. The precedence matrix comes next (photoreduced), followed by the precedence functions. Depending upon its size, the precedence matrix may be printed out in blocks since only 30 symbols are allowed across each page. The form of the precedence matrix output is self-explanatory. The precedence functions f and g (defined in section 1.7) are listed for each nonterminal columnwise (if they exist).

Next the lookup tables for a parser are shown (photoreduced). These tables are described in section 2.9.6. The recursive symbols found by RECURSIVECHECK come finally.

Exhibit 3.1 Output of Print Grammar

```
                                        SIMPLE PHRASE STRUCTURE LANGUAGE
BLOCK                 ::= <BEGIN>  BODY  <END>
BODY                  ::= BODY-
BODY-                 ::= DECL  <;>  BODY-
                          STATLIST
-------------------------------------------------------------------------
DECL                  ::= <NEW>  <L>
STATLIST              ::= STATLIST  <,>  STATEMENT
                          STATEMENT
STATEMENT             ::= VAR  <:=>  EXPR
                          BLOCK
-------------------------------------------------------------------------
VAR                   ::= <L>
EXPR                  ::= EXPR-
EXPR-                 ::= EXPR-  <+>  TERM
                          EXPR-  <->  TERM
                          <->  TERM
                          TERM
-------------------------------------------------------------------------
TERM                  ::= TERM-
TERM-                 ::= TERM-  <*>  FACTOR
                          TERM-  </>  FACTOR
                          FACTOR
-------------------------------------------------------------------------
FACTOR                ::= VAR
                          <(>  EXPR  <)>
                          NUMBER
NUMBER                ::= DIGIT
                          NUMBER  DIGIT
DIGIT                 ::= <D>
-------------------------------------------------------------------------
```

(refer also to Appendix I)

3.2.2 Precedence Conflicts

The simple phrase structure language described in the previous sub-section was modified to introduce precedence conflicts by removing the nonterminal EXPR- from the grammar. The modified grammar is shown in exhibit 3.2 below; the arrow indicates where the grammar has been modified.

Exhibit 3.2 Modified Grammar

```
                                SIMPLE PHRASE STRUCTURE LANGUAGE
BLOCK                 ::= <BEGIN>  BODY  <END>
BODY                  ::= BODY-
BODY-                 ::= DECL  <;>  BODY-
                          STATLIST
DECL                  ::= <NEW>  <L>
STATLIST              ::= STATLIST  <,>  STATEMENT
                          STATEMENT
STATEMENT             ::= VAR  <:=>  EXPR
                          BLOCK
VAR                   ::= <L>
EXPR                  ::= EXPR  <+>  TERM   <----
                          EXPR  <->  TERM
                          <->  TERM
                          TERM
TERM                  ::= TERM-
TERM-                 ::= TERM-  <*>  FACTOR
                          TERM-  </>  FACTOR
                          FACTOR
FACTOR                ::= VAR
                          <(>  EXPR  <)>
                          NUMBER
NUMBER                ::= DIGIT
                          NUMBER  DIGIT
DIGIT                 ::= <D>
```

Exhibit 3.3 shows the precedence matrix and conflicts. The syntax graph and left and right parts were essentially the same as before with the exception of references to EXPR. The nonterminal EXPR also became left recursive.

Procedure REMOVECONFLICTS removed the precedence conflicts, and the new grammar and matrix are shown in exhibit 3.4. The total processor time was 51.3 seconds.

Exhibit 3.3 Precedence Matrix and Conflicts

PRECEDENCE CONFLICTS

(<,=) BETWEEN := AND EXPR
(<,=) BETWEEN (AND EXPR

P R E C E D E N C E M A T R I X

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	300	301	302	303	304	305	306	307	308	309	310	311	312
1	<											<	<		<	=	<	<	<	<	<						
2		>		>																							
3	<											<	<		<		=	<	<	<	<						
4	<											<			<					=	<						
5							<			<		<		<							<	C	<	<	<	<	<
6										<		<		<							<		=	<	<	<	<
7										<		<		<							<		=	<	<	<	<
8										<		<		<							<				=	<	<
9										<		<		<							<				=	<	<
10							<			<		<		<							<	C	<	<	<	<	<
11		>		>		>	>	>	>		>																
12		>	>	>	>	>	>	>	>		>																
13												=															
14		>		>		>	>	>	>		>			>													>
300		>		>																							
301		=																									
302		>																									
303			=																								
304		>		=																							
305		>		>																							
306		>		>	=	>	>	>	>		>																
307		>		>		=	=				=																
308		>		>		>	>				>																
309		>		>		>	>	=	=		>																
310		>		>		>	>	>	>		>																
311		>		>		>	>	>	>		>			<													=
312		>		>		>	>	>	>		>			>													>

Exhibit 3.4 Modified Grammar After Removal of Conflicts

```
                                          SIMPLE PHRASE STRUCTURE LANGUAGE
BLOCK                  ::= <BEGIN>  BODY  <END>
BODY                   ::= BODY-
BODY-                  ::= DECL  <;>  BODY-
                           STATLIST
DECL                   ::= <NEW>  <L>
STATLIST               ::= STATLIST  <,>  STATEMENT
                           STATEMENT
STATEMENT              ::= VAR  <:=>  XXXX0                   ←
                           BLOCK
VAR                    ::= <L>
EXPR                   ::= EXPR  <+>  TERM
                           EXPR  <->  TERM
                           <->  TERM
                           TERM
TERM                   ::= TERM-
TERM-                  ::= TERM-  <*>  FACTOR
                           TERM-  </>  FACTOR
                           FACTOR
FACTOR                 ::= VAR
                           <(>  XXXX0  <)>                    ←
                           NUMBER
NUMBER                 ::= DIGIT
                           NUMBER  DIGIT
DIGIT                  ::= <D>
XXXX0                  ::= EXPR                               ←
```

Exhibit 3.4 cont'd. Precedence Matrix After Removal of Conflicts

PRECEDENCE MATRIX

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	300	301	302	303	304	305	306	307	308	309	310	311	312	313
1	<											<	<		<	■	<	<	<	<	<							
2		>		>																								
3	<											<	<		<		■	<	<	<	<							
4	<											<			<					■	<							
5																						<						■
6										<		<		<							<		■	<	<	<	<	
7										<		<		<							<		■	<	<	<	<	
8										<		<		<							<				■	<	<	
9										<		<		<							<				■	<	<	
10																						<						■
11		>		>		>	>	>	>																			
12		>	>	>	>	>	>	>	>																			
13												■																
14		>		>		>	>	>	>					>													>	
300		>		>																								
301		■																										
302		>																										
303			■																									
304		>		■																								
305		>		>																								
306		>		>	■	>	>	>	>																			
307						■	■				>																	
308		>		>		>	>																					
309		>		>		>	>	■	■																			
310		>		>		>	>	>	>																			
311		>		>		>	>	>	>					<													■	
312		>		>		>	>	>	>					>													>	
313		>		>							■																	

3.3 Example 2 Admissable Test

The grammar described in section 2.8.1 was used to test the admissability algorithms. The symbols and initial syntax graph for the grammar are given in exhibit 3.5, and the output from the admissability algorithms is shown in exhibit 3.6.

It should be noted that the $ sign next to the nonterminal Y is printed by PRINTGRAMMAR to indicate the existence of the empty statement as a right side, ie. $Y \rightarrow aYb|\varepsilon$.

The first grammar of exhibit 3.5 is the grammar before ADMISSABLE. Procedure ADMISSABLE first checks and deletes useless nonterminals. The symbol X is located and all right sides containing X are deleted. It can be seen that the production $S \rightarrow aX$ is deleted as well as the right sides of X itself. X remains in NONTERMINALS but becomes undefined. This in no way inhibits the operation of the precedence procedures later in the program.

ADMISSABLE next locates Z as being unreachable. This is deleted in the same way as X. It can be seen that because of the deletion of X previously, X also becomes unreachable. Thus all symbols found useless will become unreachable.

Exhibit 3.5

NUMBER OF PRODUCTIONS 8
NUMBER OF NON TERMINALS 4
NUMBER OF TERMINALS 2

THE SENTENCE SYMBOL FOR THE GRAMMAR IS: S

N O N T E R M I N A L S

NUMBER	SYMBOL
300	S
301	X
302	Y
303	Z

T E R M I N A L S

NUMBER	SYMBOL
-1	<EMPTY>
1	A
2	B

SYNTAX GRAPH

	SYMBOL NO.	DEFN.	ALT.	SUC.	LHSSYM	
1	300	2	0	0	0	S
2	1	0	4	3	300	A
3	301	7	0	0	0	X
4	1	0	0	5	300	A
5	302	12	0	6	0	Y
6	2	0	0	0	0	B
7	1	0	9	8	301	A
8	301	7	0	0	0	X
9	1	0	0	10	301	A
10	1	0	0	11	0	A
11	301	7	0	0	0	X
12	1	0	15	13	302	A
13	302	12	0	14	0	Y
14	2	0	0	0	0	B
15	-1	0	0	0	0	<EMPTY>
16	1	0	18	17	303	A
17	303	16	0	0	0	Z
18	2	0	0	0	303	B

Exhibit 3.6

```
                                                   EXAMPLE GRAMMAR 1
     S                    ::= <A>  X
                              <A>  Y   <B>
     X                    ::= <A>  X
                              <A>  <A>  X
  $  Y                    ::= <A>  Y   <B>
     Z                    ::= <A>  Z
                              <B>

     USELESS NON-TERMINALS
     ---------------------
X
```

```
                                                   EXAMPLE GRAMMAR 1
     S                    ::= <A>  Y   <B>
     X                    ::= (NO RIGHT SIDE DEFINED IN THE GRAMMAR)
  $  Y                    ::= <A>  Y   <B>
     Z                    ::= <A>  Z
                              <B>

     UNREACHABLE NON-TERMINALS
     -------------------------
XZ
```

```
                                                   EXAMPLE GRAMMAR 1
     S                    ::= <A>  Y   <B>
     X                    ::= (NO RIGHT SIDE DEFINED IN THE GRAMMAR)
  $  Y                    ::= <A>  Y   <B>
     Z                    ::= (NO RIGHT SIDE DEFINED IN THE GRAMMAR)
```

3.4 Example 2: Euler

Output from GRAMPA for the first part of EULER is shown in Appendix 2. The input on cards is shown below.

INPUT

```
PROGRAM→<.> BLOCK <.> ;
BLOCK→BLOKBODY STAT <END> ;
BLOKBODY→BLOKBODY STAT <;> /BLOKHEAD ;
BLOKHEAD→<BEGIN>/BLOKHEAD VARDECL <;> / BLOKHEAD LABDECL <;> ;
STAT→STAT- ;
STAT- → LABDEF STAT- /EXPR ;
LABDEF→<L> <:> ;
EXPR→EXPR- ;
EXPR-→CATENA/<OUT> EXPR-/<GOTO> PRIMARY/VAR<:=>EXPR-/
    IFCLAUSE TRUEPART EXPR-/BLOCK ;
IFCLAUSE→ <IF> EXPR <THEN> ;
TRUEPART→EXPR <ELSE> ;
CATENA→CATENA <AND> PRIMARY/DISJ ;
DISJ→DISJHEAD DISJ/CONJ ;
DISJHEAD→CONJ <∨> ;
CONJ→CONJ- ;
CONJ-→CONJHEAD CONJ-/NEGATION ;
CONJHEAD→NEGATION <∧> ;
NEGATION→ RELATION/ <¬> RELATION ;
RELATION→ CHOICE <>> CHOICE/ CHOICE <<> CHOICE/
          CHOICE<≥> CHOICE / CHOICE <≤> CHOICE /
          CHOICE <¬=> CHOICE / CHOICE <=> CHOICE/ CHOICE ;
CHOICE→CHOICE- ;
CHOICE-→ SUM/CHOICE- <MIN> SUM/CHOICE- <MAX> SUM ;
SUM→SUM-;
SUM-→TERM/<+> TERM/<-> TERM/SUM- <+> TERM/SUM- <-> TERM ;
TERM→TERM- ;
TERM-→FACTOR/TERM- <*> FACTOR/TERM- </> FACTOR/TERM- <//> FACTOR/
    TERM- <MOD> FACTOR ;
FACTOR→FACTOR- ;
FACTOR-→ PRIMARY/FACTOR- <↑> PRIMARY ;
* PRIMARY→ VAR/VAR LIST/LOGVAL NUMBER/ <S>/REFERENCE/LIST/
       <TAIL> PRIMARY/PROCDEF/ <NULL> /
       <[> EXPR <]> / <IN>/ <ISB> VAR/ <ISN> VAR/ <ISR> VAR/
       <ISL> VAR/ <ISLI> VAR/ <ISY> VAR/ <ISP> VAR/
       <ISU> VAR/ <ABS> PRIMARY/
       <LENGTH> VAR/ <INTEGER> PRIMARY/ <REAL> PRIMARY /
       <LOGICAL> PRIMARY / <LIST> PRIMARY ;
PROCDEF→PROCHEAD EXPR <≠)≠> ;
PROCHEAD→ <≠(≠> /PROCHEAD FORDECL <;> ;
LIST→LISTHEAD EXPR <)> /LISTHEAD <)> ;
LISTHEAD→ <(> /LISTHEAD EXPR <,> ;
REFERENCE→ <$> VAR ;
NUMBER→ REAL /REAL <≠> INTEGER/ REAL <≠≡> INTEGER / <≠> INTEGER /
   <≠≡> INTEGER ;
REAL→INTEGER <.> INTEGER/ INTEGER ;
INTEGER→ INTEGER- ;
INTEGER- → DIGIT /INTEGER- DIGIT ;
DIGIT→<1>/<2>/<3>/<4>/<5>/<6>/<7>/<8>/<9>/<0> ;
LOGVAL→ <TRUE>/<FALSE> ;
VAR→VAR- ;
VAR- → <L> /VAR- <[> EXPR <]> /VAR- <.> ;
LABDECL→ <LABEL> <L> ;
FORDECL→ <FORMAL> <L> ;
VARDECL→ <NEW> <L> ;
.
```

* See note on next page

The execution time to compile and execute the first major program block was 32.5 seconds. Compilation and execution of the complete program took 388 seconds.

Note: The production PRIMARY→LOGVAL NUMBER should read PRIMARY→LOGVAL|NUMBER. This error does not invalidate the example or introduce precedence conflicts.

CHAPTER 4

FUTURE DIRECTIONS

In this report, a set of Algol procedures has been described which can read a context-free grammar from cards and set it up in such a way that it can be easily studied and manipulated. Several procedures have been described which perform analyses of various kinds on grammars. Most of these procedures are geared to simple precedence.

The first improvement to the system would be a better form of representation for the large array holding the left and right sets. A good representation would be to use a binary table. Each nonterminal would have space for every symbol to be in its left and right set. This array would initially be set to zero; a binary 1 could then be entered to flag those symbols that are in the sets. This type of array would have 48 entries in one word; thus a grammar with a vocabulary of 120 symbols with 60 nonterminals would require only 360 words of storage for the sets instead of the current 3000 words. The array could be scanned and handled much faster by the precedence and recursion procedures than presently. Also, the bit manipulation routines used to pack the

left and right sets and the precedence matrix could be written in assembly language to further reduce the program execution time.

The first avenue of expansion for the system would be the automatic production of a simple precedence syntax analyser for the grammar. Most of the facilities for doing this already exist in the system e.g. precedence matrix (function) production, look-up tables. However, facilities for handling semantic rules and names should be included for producing a simple lexical analyser. This in turn would probably involve an extension of the meta language used in GRAMPA to handle such things as zero and one repeats etc.[16][18][19]. This would allow manipulations to be performed on grammars such as PASCAL[16] or BASIC[19] without altering the meta language used in their definitions.

A straightforward extension to the present GRAMPA system would be the inclusion of a procedure to remove the empty statement from the grammar (Wood[17]). Also procedures could be included to perform the more complex analyses for extended precedence using the rules of Graham[8].

The GRAMPA program will soon become excessively large as more procedures are added. Therefore, an overlay scheme should be devised to reduce storage, and a system of control cards could be designed to enable the user to select only those procedures that he needs. The control card system could in fact be designed as a language for grammatical analysis and manipulation. This would better serve the aims of developing a compiler laboratory which would also include, for example, routines for automatic compiler construction.

It is the opinion of the author that a fruitful first step has been taken into automatic grammar manipulation, and it is hoped that the program, even in its present form, can be successfully applied as an aid in programming language development, or for practical studies in formal language theory.

REFERENCES

1) Aho A.V., Ullman J.D.: *Linear Precedence Functions for Weak Precedence Grammars*, to appear (1973) in International Journal of Computer Mathematics.

2) Bell J.R.: *A New Method for Determining Linear Precedence Functions for Precedence Grammars*, C.ACM 12, (10) October 1969, pp 567-569.

3) Cheatham T.E., Sattley K: *Syntax Directed Compiling*, Proc. AFIPS 1964, Spring Joint Computer Conference, Vol 25, pp 31-57.

4) Cohen D.J., Gotlieb C.C.: *A List Structure Form of Grammars for Syntactic Analysis*, Computing Surveys, Vol 2 (1), March 1970, pp 65-82.

5) Feldman J., Gries D.: *Translator Writing Systems*, C.ACM 11 (2) February 1968, pp 77-113.

6) Floyd R.W.: *Syntactic Analysis and Operator Precedence*, J.ACM 10, July 1963, pp 316-333. (Not referred to explicitly in the report).

7) George J.E.: *"SIMPLE - A Simple Precedence Translator Writing System*, Stanford University, STAN-CS-71-226.

8) Graham S.: *Precedence Languages and Bounded Right Context Languages*, Ph.D. Thesis, Stanford University, 1971.

9) Martin D.F.: *A Boolean Matrix Method for the Computation of Linear Precedence Functions*, C.ACM 15 (6) June 1972, pp 448-454.

10) McKeeman W.M.: *An Approach to Computer Language Design*, Stanford University Technical Report #CS48, August 31, 1966.

11) Morris R.: *Scatter Storage Techniques*, C.ACM 11 (1), June 1968, pp 38-44.

12) Naur P. (Ed.): *Revised Report on the Algorithmic Language Algol 60*, C.ACM 6, June 1963, pp 1-17.

13) Wirth N.: *PL360 - A Programming Language for the 360 Computers*, J.ACM 15 (1), June 1968, pp 37-76.

14) -------, Weber H.: *EULER - A Generalization of Algol, and its Formal Definition, Parts I and II* C.ACM 9 (1), June 1966, pp 13-23, C.ACM 9 (2), Feb. 1966, pp 89-99.

15) -------: *Algorithm 265 - Find Precedence Functions,* C.ACM 8 (10), Oct. 1965, pp 604-605.

16) -------: *The Programming Language PASCAL,* Acta Informatica, 1971 pp 36-63.

17) Wood D.: *Introduction to Formal Language Theory,* Computer Science Technical Report 71/4, McMaster University, 1971.

18) -------: *Syntax Generated Interpretation of Programming Languages,* Revue Francaise d'Informatique et de Recherche Operationelle, 4, B-2, 1970, pp. 71-89.

19) Lee J.A.N.: *The Formal Definition of the BASIC Language,* The Computer Journal, 15 (1) 1972, pp 37-41.

20) Rohl J. S.: *A Note on Backus - Naur Form,* The Computer Journal, 10, 1968 pp 336-337.

APPENDIX I

OUTPUT FROM GRAMPA FOR SIMPLE PHRASE STRUCTURE LANGUAGE

SYNTAX GRAPH

	SYMBOL NO.	DEFN.	ALT.	SUC.	LHSSYM	
1	300	2	0	0	0	BLOCK
2	1	0	0	3	300	BEGIN
3	301	5	0	4	0	BODY
4	2	0	0	0	0	END
5	302	6	0	0	301	BODY-
6	303	45	9	7	302	DECL
7	3	0	0	8	0	;
8	302	6	0	0	0	BODY-
9	304	10	0	0	302	STATLIST
10	304	10	13	11	304	STATLIST
11	4	0	0	12	0	;
12	305	14	0	0	0	STATEMENT
13	305	14	0	0	304	STATEMENT
14	306	41	17	15	305	VAR
15	5	0	0	16	0	:=
16	307	18	0	0	0	EXPR
17	300	2	0	0	305	BLOCK
18	308	19	0	0	307	EXPR-
19	308	19	22	20	308	EXPR-
20	6	0	0	21	0	+
21	309	28	0	0	0	TERM
22	308	19	25	23	308	EXPR-
23	7	0	0	24	0	-
24	309	28	0	0	0	TERM
25	7	0	27	26	308	-
26	309	28	0	0	0	TERM
27	309	28	0	0	308	TERM
28	310	29	0	0	309	TERM-
29	310	29	32	30	310	TERM-
30	8	0	0	31	0	*
31	311	36	0	0	0	FACTOR
32	310	29	35	33	310	TERM-
33	9	0	0	34	0	/
34	311	36	0	0	0	FACTOR
35	311	36	0	0	310	FACTOR
36	306	41	37	0	311	VAR
37	10	0	40	38	311	(
38	307	18	0	39	0	EXPR
39	11	0	0	0	0	)
40	312	42	0	0	311	NUMBER
41	12	0	0	0	306	L
42	313	47	43	0	312	DIGIT
43	312	42	0	44	312	NUMBER
44	313	47	0	0	0	DIGIT
45	13	0	0	46	303	NEW
46	12	0	0	0	0	L
47	14	0	0	0	313	D

```
NUMBER OF PRODUCTIONS  25
NUMBER OF NON TERMINALS   14
NUMBER OF TERMINALS  14

THE SENTENCE SYMBOL FOR THE GRAMMAR IS: BLOCK
```

N O N   T E R M I N A L S

NUMBER	SYMBOL
300	BLOCK
301	BODY
302	BODY-
303	DECL
304	STATLIST
305	STATEMENT
306	VAR
307	EXPR
308	EXPR-
309	TERM
310	TERM-
311	FACTOR
312	NUMBER
313	DIGIT

T E R M I N A L S

NUMBER	SYMBOL
1	BEGIN
2	END
3	;
4	,
5	:=
6	+
7	-
8	*
9	/
10	(
11	)
12	L
13	NEW
14	D

L E F T P A R T S

SYMBOL NO.	LEFT PART										
300	1										
301	302	303	13	304	305	306	12	300	1		
302	303	13	304	305	306	12	300	1			
303	13										
304	304	305	306	12	300	1					
305	306	12	300	1							
306	12										
307	308	7	309	310	311	306	12	10	312	313	14
308	308	7	309	310	311	306	12	10	312	313	14
309	310	311	306	12	10	312	313	14			
310	310	311	306	12	10	312	313	14			
311	306	12	10	312	313	14					
312	313	14	312								
313	14										

R I G H T P A R T S

SYMBOL NO.	RIGHT PART															
300	2															
301	302	304	305	307	308	309	310	311	306	12	11	312	313	14	300	2
302	302	304	305	307	308	309	310	311	306	12	11	312	313	14	300	2
303	12															
304	305	307	308	309	310	311	306	12	11	312	313	14	300	2		
305	307	308	309	310	311	306	12	11	312	313	14	300	2			
306	12															
307	308	309	310	311	306	12	11	312	313	14						
308	309	310	311	306	12	11	312	313	14							
309	310	311	306	12	11	312	313	14								
310	311	306	12	11	312	313	14									
311	306	12	11	312	313	14										
312	313	14														
313	14															

PRECEDENCE MATRIX

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	300	301	302	303	304	305	306	307	308	309	310	311	312	313
1	<											<	<		<	=	<	<	<	<	<							
2		>		>																								
3	<											<	<		<		=	<	<	<	<							
4	<											<			<					=	<							
5							<			<		<		<							<	=	<	<	<	<	<	<
6										<		<		<							<			=	<	<	<	<
7										<		<		<							<			=	<	<	<	<
8										<		<		<							<					=	<	<
9										<		<		<							<					=	<	<
10							<			<		<		<							<	=	<	<	<	<	<	<
11		>		>		>	>	>	>		>																	
12		>	>	>	>	>	>	>	>		>																	
13												=																
14		>		>		>	>	>	>		>			>														>
300		>		>																								
301		=																										
302		>																										
303			=																									
304		>		=																								
305		>		>																								
306		>		>	=	>	>	>	>		>																	
307		>		>							=																	
308		>		>		=	=				>																	
309		>		>		>	>				>																	
310		>		>		>	>	=	=		>																	
311		>		>		>	>	>	>		>																	
312		>		>		>	>	>	>		>			<														=
313		>		>		>	>	>	>		>			>														>

PRECEDENCE FUNCTIONS

F	G	SYMBOL
1	4	BEGIN
3	1	END
2	1	;
3	2	,
1	6	:=
2	4	+
2	4	-
3	5	*
3	5	/
1	4	(
6	3	)
7	4	L
4	3	NEW
8	7	D
3	4	BLOCK
1	1	BODY
2	2	BODY-
1	3	DECL
2	3	STATLIST
3	3	STATEMENT
6	4	VAR
3	1	EXPR
4	2	EXPR-
5	2	TERM
5	3	TERM-
6	3	FACTOR
6	4	NUMBER
8	6	DIGIT

TRANSPOSE SYNTAX GRAPH FOR ANALYSER

NONTERMINALS INDEX-POINTER TABLE

SYMBOL NUMBER	POINTER TO RHSTABLE
300	13
301	0
302	14
303	15
304	18
305	22
306	23
307	0
308	27
309	34
310	35
311	42
312	43
313	46

TERMINALS INDEX-POINTER TABLE

SYMBOL NUMBER	POINTER TO RHSTABLE
-1	0
0	0
1	1
2	3
3	0
4	0
5	0
6	0
7	4
8	0
9	0
10	6
11	0
12	9
13	10
14	12

TRANSPOSED SYNTAX GRAPH FOR LHS LOOKUP

	SYMBOL NO.	ALT.	SUC.	
1	301	0	2	BODY
2	2	0	0	END
3	300	0	-1	BLOCK
4	309	0	0	TERM
5	308	0	-1	EXPR-
6	307	0	7	EXPR
7	11	0	0	)
8	311	0	-1	FACTOR
9	306	0	-1	VAR
10	12	0	0	L
11	303	0	-1	DECL
12	313	0	-1	DIGIT
13	305	0	-1	STATEMENT
14	301	0	-1	BODY
15	3	0	16	;
16	302	0	0	BODY-
17	302	0	-1	BODY-
18	302	19	-1	BODY-
19	4	0	20	;
20	305	0	0	STATEMENT
21	304	0	-1	STATLIST
22	304	0	-1	STATLIST
23	5	26	24	:=
24	307	0	0	EXPR
25	305	0	-1	STATEMENT
26	311	0	-1	FACTOR
27	307	28	-1	EXPR
28	6	31	29	+
29	309	0	0	TERM
30	308	0	-1	EXPR-
31	7	0	32	-
32	309	0	0	TERM
33	308	0	-1	EXPR-
34	308	0	-1	EXPR-
35	309	36	-1	TERM
36	8	39	37	*
37	311	0	0	FACTOR
38	310	0	-1	TERM-
39	9	0	40	/
40	311	0	0	FACTOR
41	310	0	-1	TERM-
42	310	0	-1	TERM-
43	311	44	-1	FACTOR
44	313	0	0	DIGIT
45	312	0	-1	NUMBER
46	312	0	-1	NUMBER

R E C U R S I O N   C H E C K

LEFT RECURSIVE SYMBOLS

STATLIST
EXPR-
TERM-
NUMBER

RIGHT RECURSIVE SYMBOLS

BODY-

IMBEDDED RECURSIVE SYMBOLS

BLOCK
BODY
BODY-
STATLIST
STATEMENT
EXPR
EXPR-
TERM
TERM-
FACTOR

APPENDIX 2

FIRST PART OF GRAMPA OUTPUT FOR EULER

THE GRAMMAR

```
                                                        E U L E R
PROGRAM         ::= <.>  BLOCK  <.>
BLOCK           ::= BLOKBODY  STAT  <END>
BLOKBODY        ::= BLOKBODY  STAT  <;>
                    BLOKHEAD
STAT            ::= STAT-
BLOKHEAD        ::= <BEGIN>
                    BLOKHEAD  VARDECL  <;>
                    BLOKHEAD  LABDECL  <;>
VARDECL         ::= <NEW>  <L>
LABDECL         ::= <LABEL>  <L>
STAT-           ::= LABDEF  STAT-
                    EXPR
LABDEF          ::= <L>  <:>
EXPR            ::= EXPR-
EXPR-           ::= CATENA
                    <OUT>  EXPR-
                    <GOTO>  PRIMARY
                    VAR  <:=>  EXPR-
                    IFCLAUSE  TRUEPART  EXPR-
                    BLOCK
CATENA          ::= CATENA  <AND>  PRIMARY
                    DISJ
PRIMARY         ::= VAR
                    VAR  LIST
                    LOGVAL  NUMBER
                    <S>
                    REFERENCE
                    LIST
                    <TAIL>  PRIMARY
                    PROCDEF
                    <NULL>
                    <[>  EXPR  <]>
                    <IN>
                    <ISB>  VAR
                    <ISN>  VAR
                    <ISR>  VAR
                    <ISL>  VAR
                    <ISLI>  VAR
                    <ISY>  VAR
                    <ISP>  VAR
                    <ISU>  VAR
                    <ABS>  PRIMARY
                    <LENGTH>  VAR
                    <INTEGER>  PRIMARY
                    <REAL>  PRIMARY
                    <LOGICAL>  PRIMARY
                    <LIST>  PRIMARY
VAR             ::= VAR-
IFCLAUSE        ::= <IF>  EXPR  <THEN>
TRUEPART        ::= EXPR  <ELSE>
DISJ            ::= DISJHEAD  DISJ
                    CONJ
DISJHEAD        ::= CONJ  <∨>
CONJ            ::= CONJ-
CONJ-           ::= CONJHEAD  CONJ-
                    NEGATION
CONJHEAD        ::= NEGATION  <∧>
NEGATION        ::= RELATION
                    <¬>  RELATION
RELATION        ::= CHOICE  <>>  CHOICE
                    CHOICE  <<>  CHOICE
                    CHOICE  <≥>  CHOICE
                    CHOICE  <≤>  CHOICE
                    CHOICE  <¬=>  CHOICE
                    CHOICE  <=>  CHOICE
                    CHOICE
CHOICE          ::= CHOICE-
CHOICE-         ::= SUM
                    CHOICE-  <MIN>  SUM
                    CHOICE-  <MAX>  SUM
SUM             ::= SUM-
SUM-            ::= TERM
                    <+>  TERM
                    <->  TERM
                    SUM-  <+>  TERM
                    SUM-  <->  TERM
TERM            ::= TERM-
TERM-           ::= FACTOR
                    TERM-  <*>  FACTOR
                    TERM-  </>  FACTOR
                    TERM-  <//>  FACTOR
                    TERM-  <MOD>  FACTOR
FACTOR          ::= FACTOR-
FACTOR-         ::= PRIMARY
                    FACTOR-  <*>  PRIMARY
```

THE GRAMMAR CONT'D

```
LIST          ::= LISTHEAD  EXPR  <)>
                  LISTHEAD  <)>
LOGVAL        ::= <TRUE>
                  <FALSE>
NUMBER        ::= REAL
                  REAL  <#>  INTEGER
                  REAL  <#->  INTEGER
                  <#>  INTEGER
                  <#->  INTEGER
REFERENCE     ::= <$>  VAR
PROCDEF       ::= PROCHEAD  EXPR  <#)#>
PROCHEAD      ::= <#(#>
                  PROCHEAD  FORDECL  <;>
FORDECL       ::= <FORMAL>  <L>
LISTHEAD      ::= <(>
                  LISTHEAD  EXPR  <,>
REAL          ::= INTEGER  <.>  INTEGER
                  INTEGER
INTEGER       ::= INTEGER-
INTEGER-      ::= DIGIT
                  INTEGER-  DIGIT
DIGIT         ::= <1>
                  <2>
                  <3>
                  <4>
                  <5>
                  <6>
                  <7>
                  <8>
                  <9>
                  <0>
VAR-          ::= <L>
                  VAR-  <(>  EXPR  <)>
                  VAR-  <.>
```

```
NUMBER OF PRODUCTIONS 119
NUMBER OF NON TERMINALS  44
NUMBER OF TERMINALS  74

THE SENTENCE SYMBOL FOR THE GRAMMAR IS: PROGRAM
```

N O N T E R M I N A L S

NUMBER	SYMBOL
300	PROGRAM
301	BLOCK
302	BLOKBODY
303	STAT
304	BLOKHEAD
305	VARDECL
306	LABDECL
307	STAT-
308	LABDEF
309	EXPR
310	EXPR-
311	CATENA
312	PRIMARY
313	VAR
314	IFCLAUSE
315	TRUEPART
316	DISJ
317	DISJHEAD
318	CONJ
319	CONJ-
320	CONJHEAD
321	NEGATION
322	RELATION
323	CHOICE
324	CHOICE-
325	SUM
326	SUM-
327	TERM
328	TERM-
329	FACTOR
330	FACTOR-
331	LIST
332	LOGVAL
333	NUMBER
334	REFERENCE
335	PROCDEF
336	PROCHEAD
337	FORDECL
338	LISTHEAD
339	REAL
340	INTEGER
341	INTEGER-
342	DIGIT
343	VAR-

T E R M I N A L S

NUMBER	SYMBOL
1	.

SYMBOLS CONT'D

No.	Symbol
2	END
3	;
4	BEGIN
5	(
6	)
7	OUT
8	GOTO
9	:=
10	IF
11	THEN
12	ELSE
13	AND
14	[illegible]
15	[illegible]
16	[illegible]
17	[illegible]
18	[illegible]
19	[illegible]
20	[illegible]
21	[illegible]
22	=
23	MIN
24	MAX
25	+
26	-
27	*
28	/
29	//
30	MOD
31	+
32	$
33	TAIL
34	NULL
35	[
36	]
37	IN
38	[illegible]SSB
39	[illegible]SSN
40	[illegible]SSR
41	[illegible]SSL
42	[illegible]SSLI
43	[illegible]SSY
44	[illegible]SSP
45	[illegible]SSU
46	ABS
47	LENGTH
48	INTEGER
49	REAL
50	LOGICAL
51	LIST
52	#)#
53	#(#
54	)
55	(
56	[illegible]
57	#
58	#
59	#B
60	1
61	2
62	3
63	4
64	5
65	6
66	7
67	8
68	9
69	0
70	TRUE
71	FALSE
72	LABEL
73	FORMAL
74	NEW

SYNTAX GRAPH

	SYMBOL NO.	DEFN.	ALT.	SUC.	LHSSYM	
1	300	2	0	0	0	PROGRAM
2	1	0	0	3	300	.
3	301	5	0	4	0	BLOCK
4	1	0	0	0	0	.
5	302	8	0	6	301	BLOKBODY
6	303	19	0	7	0	STAT
7	2	0	0	0	0	END
8	302	8	11	9	302	BLOKBODY
9	303	19	0	10	0	STAT
10	3	0	0	0	0	;
11	304	12	0	0	302	BLOKHEAD
12	4	0	13	0	304	BEGIN
13	304	12	16	14	304	BLOKHEAD
14	305	224	0	15	0	VARDECL
15	3	0	0	0	0	;
16	304	12	0	17	304	BLOKHEAD
17	306	229	0	18	0	LABDECL
18	3	0	0	0	0	;
19	307	21	0	0	303	STAT-
20	308	23	22	21	307	LABDEF
21	307	21	0	0	0	STAT-
22	309	25	0	0	307	EXPR
23	5	0	0	24	308	L
24	6	0	0	0	0	[illegible]
25	310	25	0	0	309	EXPR-
26	311	43	27	0	310	CATENA
27	7	0	29	28	310	OUT
28	310	26	0	0	0	EXPR-
29	8	0	31	30	310	GOTO
30	312	119	0	0	0	PRIMARY
31	313	212	34	32	310	VAR
32	9	0	0	33	0	:=
33	310	25	0	0	0	EXPR-
34	314	39	37	35	310	IFCLAUSE
35	315	41	0	36	0	TRUEPART
36	310	26	0	0	0	EXPR-
37	301	5	0	0	310	BLOCK
38	10	0	0	39	314	IF
39	309	25	0	40	0	EXPR
40	11	0	0	0	0	THEN
41	309	25	0	42	315	EXPR
42	12	0	0	0	0	ELSE
43	311	43	46	44	311	CATENA
44	13	0	0	45	0	AND
45	312	119	0	0	0	PRIMARY
46	316	47	0	0	311	DISJ
47	317	50	49	48	316	DISJHEAD
48	316	47	0	0	0	DISJ
49	318	52	0	0	316	CONJ
50	318	52	0	51	317	CONJ
51	14	0	0	0	0	[illegible]
52	319	53	0	0	318	CONJ-
53	320	56	55	54	319	CONJHEAD
54	319	53	0	0	0	CONJ-
55	321	58	0	0	319	NEGATION
56	321	58	0	57	320	NEGATION
57	15	0	0	0	0	¬
58	322	61	59	0	321	RELATION
59	15	3	0	60	321	¬
60	322	61	0	0	0	RELATION
61	323	80	64	62	322	CHOICE
62	17	0	0	63	0	>
63	323	87	0	0	0	CHOICE
64	323	80	67	65	322	CHOICE
65	18	0	0	66	0	<
66	323	80	0	0	0	CHOICE
67	323	80	70	68	322	CHOICE
68	19	0	0	69	0	≥
69	323	87	0	0	0	CHOICE
70	323	80	73	71	322	CHOICE
71	20	0	0	72	0	≤
72	323	87	0	0	0	CHOICE
73	323	80	76	74	322	CHOICE
74	21	0	0	75	0	¬=
75	323	87	0	0	0	CHOICE
76	323	80	79	77	322	CHOICE
77	22	0	0	78	0	=
78	323	87	0	0	0	CHOICE
79	323	80	0	0	322	CHOICE
80	324	81	0	0	323	CHOICE-
81	325	85	82	0	324	SUM
82	324	81	85	83	324	CHOICE-
83	23	0	0	84	0	MIN
84	325	85	0	0	0	SUM
85	324	81	0	86	324	CHOICE-
86	24	0	0	87	0	MAX
87	325	88	0	0	0	SUM
88	326	89	0	0	325	SUM-
89	327	100	90	0	326	TERM
90	25	0	92	91	326	+
91	327	100	0	0	0	TERM
92	26	0	94	93	326	-
93	327	100	0	0	0	TERM
94	326	89	97	95	326	SUM-
95	25	0	0	96	0	+
96	327	100	0	0	0	TERM
97	326	89	0	98	326	SUM-
98	26	0	0	99	0	-
99	327	100	0	0	0	TERM
100	328	101	0	0	327	TERM-
101	329	114	102	0	328	FACTOR
102	328	101	105	103	328	TERM-
103	27	0	0	104	0	*
104	329	114	0	0	0	FACTOR
105	328	101	108	106	328	TERM-
106	28	0	0	107	0	/
107	329	114	0	0	0	FACTOR
108	328	101	111	109	328	TERM-
109	29	0	0	110	0	//
110	329	114	0	0	0	FACTOR
111	328	101	0	112	328	TERM-
112	30	0	0	113	0	MOD
113	329	114	0	0	0	FACTOR
114	330	115	0	0	329	FACTOR-

115	312	119	116	0	330	PRIMARY
116	330	115	0	117	330	FACTOR-
117	31	0	0	118	0	*
118	312	119	0	0	0	PRIMARY
119	313	212	120	0	312	VAR
120	313	212	122	121	312	VAR
121	331	173	0	0	0	LIST
122	332	213	124	123	312	LOGVAL
123	333	181	0	0	0	NUMBER
124	32	0	125	0	312	S
125	334	179	126	0	312	REFERENCE
126	331	170	127	0	312	LIST
127	33	0	129	128	312	TAIL
128	312	119	0	0	0	PRIMARY
129	335	167	130	0	312	PROCDEF
130	34	0	131	0	312	NULL
131	35	0	134	132	312	[
132	309	25	0	133	0	EXPR
133	36	0	0	0	0	]
134	37	0	135	0	312	IN
135	38	0	137	136	312	ISB
136	313	212	0	0	0	VAR
137	39	0	139	138	312	ISN
138	313	212	0	0	0	VAR
139	40	0	141	140	312	ISR
140	313	212	0	0	0	VAR
141	41	0	143	142	312	ISL
142	313	212	0	0	0	VAR
143	42	0	145	144	312	ISLI
144	313	212	0	0	0	VAR
145	43	0	147	146	312	ISV
146	313	212	0	0	0	VAR
147	44	0	149	148	312	ISP
148	313	212	0	0	0	VAR
149	45	0	151	150	312	ISU
150	313	212	0	0	0	VAR
151	46	0	153	152	312	ABS
152	312	119	0	0	0	PRIMARY
153	47	0	155	154	312	LENGTH
154	313	212	0	0	0	VAR
155	48	0	157	156	312	INTEGER
156	312	119	0	0	0	PRIMARY
157	49	0	159	158	312	REAL
158	312	119	0	0	0	PRIMARY
159	50	0	161	160	312	LOGICAL
160	312	119	0	0	0	PRIMARY
161	51	0	0	162	312	LIST
162	312	119	0	0	0	PRIMARY
163	336	166	0	164	335	PROCHEAD
164	309	25	0	165	0	EXPR
165	52	0	0	0	0	#)#
166	53	0	167	0	336	#(#
167	336	165	0	168	336	PROCHEAD
168	337	222	0	169	0	FORDECL
169	3	0	0	0	0	$
170	338	175	173	171	331	LISTHEAD
171	309	25	0	172	0	EXPR
172	54	0	0	0	0	)
173	338	175	0	174	331	LISTHEAD
174	54	0	0	0	0	)

175	55	0	176	0	338	(
176	338	175	0	177	338	LISTHEAD
177	309	25	0	178	0	EXPR
178	56	0	0	0	0	)
179	57	0	0	180	334	$
180	313	212	0	0	0	VAR
181	339	192	182	0	333	REAL
182	339	192	185	183	333	REAL
183	58	0	0	184	0	#
184	340	196	0	0	0	INTEGER
185	339	192	188	186	333	REAL
186	59	0	0	187	0	#=
187	340	196	0	0	0	INTEGER
188	58	0	190	189	333	#
189	340	196	0	0	0	INTEGER
190	59	0	0	191	333	#=
191	340	196	0	0	0	INTEGER
192	340	196	195	193	339	INTEGER
193	1	0	0	194	0	.
194	340	196	0	0	0	INTEGER
195	340	196	0	0	339	INTEGER
196	341	197	0	0	340	INTEGER-
197	342	200	198	0	341	DIGIT
198	341	197	0	199	341	INTEGER-
199	342	200	0	0	0	DIGIT
200	60	0	201	0	342	1
201	61	0	202	0	342	2
202	62	0	203	0	342	3
203	63	0	204	0	342	4
204	64	0	205	0	342	5
205	65	0	206	0	342	6
206	66	0	207	0	342	7
207	67	0	208	0	342	8
208	68	0	209	0	342	9
209	69	0	0	0	342	0
210	70	0	211	0	332	TRUE
211	71	0	0	0	332	FALSE
212	343	213	0	0	313	VAR-
213	5	0	214	0	343	L
214	343	213	218	215	343	VAR-
215	35	0	0	216	0	[
216	309	25	0	217	0	EXPR
217	36	0	0	0	0	]
218	343	213	0	219	343	VAR-
219	7	0	0	0	0	.
220	72	0	0	221	306	LABEL
221	5	0	0	0	0	L
222	73	0	0	223	337	FORMAL
223	5	0	0	0	0	L
224	74	0	0	225	305	NEW
225	5	0	0	0	0	L

APPENDIX 3

HASH CODING TECHNIQUE OF MORRIS [11]

The hash coding technique of Morris is as follows:

1) Calculate an address ℓ in the table by using some transformation on the key as an index,
2) If the item is already at this address or if the place is empty, the job is done,
3) If some other key is there, call a pseudorandom number generator for an integer offset ρ. Make the next probe at $\ell + \rho$ and go to step 2.

If KEY is the key, then the following coding (in FORTRAN) calculates an index into the table (assumed to be of size 2^N):

```
      IHASH = 0
      KRAND = 1
      KEYA = IABS (KEY)
      DO 11 I = 1, WDSIZE, N
          IHASH = IHASH + KEYA/(2**(I - 1))
11    CONTINUE
      KPLACE = MOD (IHASH + KRAND/4, 2**N) +1
```

Where KPLACE is the index, and WDSIZE is the word

size in bits. KRAND is the random offset. To create the random offset, the simplest form of psendorandom number generator is used:

KRAND = MOD (5*KRAND, 2** (N + 2)).

The efficiency of this technique is best expressed in terms of the average number E of probes necessary to retrieve an item in the table. E depends on the fraction of the table that is full (if k items are in the table, size N, then $\alpha = k/N$).

The expected number of probes, A, to enter the (k +1) st. item, including the final probe is:

$$A = 1/(1 - \frac{k}{N + 1}).$$

For large N, $k/(N + 1) \approx k/N$ i.e. $A = 1/(1-\alpha)$.

If E is equal to the average of A for k = 0 to k - 1, then $E \approx \frac{N}{K}\int \frac{dx}{1-x} = -\frac{1}{\alpha}\log(1 - \alpha)$.

eg. some values of E are:

Load Factor	E
0.1	1.05
0.5	1.39
0.75	1.83
0.9	2.56

The procedure HASH in GRAMPA is based directly on this technique. The hash key is calculated by arithmetically adding the contents of the three words containing a symbol.

During a test using the BNF of Algol 60 with 118 nonterminals and 90 terminals, the following statistics were obtained:

	Degree of table fill:	Average number of probes, E
Nonterminals:	0.46	1.51
Terminals:	0.35	1.55

i.e. the observed behaviour was not quite as good as the theoretical predictions, but still acceptable.

APPENDIX 4

CENTRAL PROCESSOR TIME MODEL

For the first block of the program i.e. INGRAMMAR and PRINTGRAMMAR, the following regression equation was derived:

$$T = \text{CP time (secs.)} = 13.12 + 0.1075 \times P + 0.0492 \times V \text{......(i)}$$

where P = number of productions in grammar, and V is the total number of symbols in the vocabulary. Index of determination was 0.999 and F - ratio = 384. T includes compilation and load times which are of the order of 9 seconds.

Compilation and load times for the complete system (including all precedence analysis routines) are 29.8 and about 1.4 seconds respectively. Total CP time for the whole system is difficult to estimate. Some values obtained are summarized below in table A4-1.

Table A4-1: Some CP Times Obtained for Example Grammars

Grammar	No. of Productions	No. of Terminals	No. of Non-Terminals	Compile, Load Execute 1st Block (secs)	Compile, Load Execute Whole System (secs)
Simple Phrase	25	14	14	18.7	46.3
Euler	119	74	44	32.5	388.1
PL/360	154	62	65	40.9	254.2
Algol 60	300	90	117	56.9	-
BNF of BNF	44	28	14	17.8	-

APPENDIX 5

PROGRAM LISTING

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN


```
00**  GRAMPA
      ≠BEGIN≠
      ≠COMMENT≠ ***********************************************************

                G R A M M A T I C A L   A N A L Y S I S   P R O G R A M

                ***********************************************************

10**        -------------------------
               AUTHOR     K R BARNES
            LANGUAGE      ALGOL
            COMPUTER      CDC 6400
                DATE       JAN 1972
            -------------------------

              *    THIS PROGRAM COMPRISES A SET OF PROCEDURES WHICH      *
              *    CAN BE USED TO ANALYSE A GRAMMAR SUBMITTED IN         *
20**          *    REVERSE BACKUS NOTATION.                              *
              *                                                          *;

      ≠COMMENT≠  THE FOLLOWING DECLARATIONS ARE GLOBAL,THEIR USAGE IS AS
                 FOLLOWS,

                1. -I-,-J-,-T1-,-T2-,-T3-,-T4-,-N-  ...GENERAL PURPOSE INTEGER
                   IDENTIFIERS,
                2. -NGRAPH- ...  NUMBER OF ENTRIES IN SYNTAX GRAPH,
30**            3. -NTERM-  ...  NUMBER OF TERMINAL SYMBOLS IN THE GRAMMAR,
                4. -NNONTERM ...  NUMBER OF NONTERMINALS IN THE GRAMMAR,
                5. -TERMNO- ...  CURRENT NUMBER OF TERMINAL SYMBCLS MET WHILE
                   PROCESSING THE GRAMMAR,
                6, -NTERMNO- ...  NUMBER OF LATEST NONTERMINAL MET WHILE
                  PROCESSING THE  GRAMMAR -STARTS AT 300,
                7. -PRODUCTIONS- ...  A COUNT OF THE NUMBER OF PRODUCTIONS
                  PROCESSED,
                8. -SENTENCE- ... THE VALUE  OF THE SENTENCE SYMBOL,
                9. -DEFBOX- ...  USED IN THE FORMATION OF THE SYNTAX GRAPH TO
40**              STORE THE ROW NUMBER OF THE FIRST SYMBOL IN A PRODUCTION,
               10. -NXTBOX- ...  POINTER TO THE NEXT FREE ROW IN THE SYNTAX
                  GRAPH,
               11. -LHSBOX- ...  USED IN FORMATION OF SYNTAX GRAPH TO HOLD THE
                   VALUE OF THE LHS SYMBOL OF THE PRODUCTION BEING PROCESSED,
               12. -RHSNO- .USED TO KEEP A COUNT OF THE NUMBER OF ALTERNATES
                   OF A PRODUCTION,
               13. -CHAR- ... CURRENT CHARACTER BEING SCANNED,
               14. -NCHAR- ... NUMBER OF CHARACTERS IN A TERMINAL OR
                   NONTERMINAL -CALCULATED IN -PRINTSYMBOL- ROUTINE,
50**           15. -IC- ... POINTER TO POSITION OF CHARACTER BEING SCANNED ON
                   THE CARD,
               16. -J1-,-J2-,-J3- ...HOLD THE NUMBERS 64,2,4 RESPECTIVELY FOR
                   USE IN INTEGER-DIVIDE ARITHMETIC EXPRESSIONS,
               17. -IWNO- ... COUNTER USED IN GETTING A SYMBOL FROM CARD,
               18. -NO- ... RETURNED BY HASH PROCEDURE - VALUE OF SYMBOL JUST
                   LOCATED IN TABLES OR JUST INSERTED IN TABLES.
               19. -SIZE- ... NUMBER OF TERMINALS AND NONTERMINALS,
```

```
 60**        ARRAYS
                 1. -SYNG-  SYNTAX GRAPH,
                 2. -LINE-  INTEGER REPRESENTATIONS OF CHARACTERS ON CURRENT
                    DATA CARD,
                 3. -WORD-  USED IN FORMATION OF HASH CODES,
                 4. -TERMTABLE-  HOLDS HASH KEY AND POINTER TO -TERMIALS-
                    TO PROCESS TERMINALS,
                 5. -TERMINALS-  HOLDS IMAGES OF TERMINAL SYMBOLS,
                 6. -NONTERMTABLE-,NONTERMINALS- SAME AS ABOVE FOR NONTERMINALS;

 70**
        ≠INTEGER≠ I,J,NGRAPH,I1,I2,I3,I4,NTERM,NNONTERM,SIZE,
                  LCOUNT,DEFBOX,NXTBOX,TERMNO,NTERMNO,NO,N,IWNO,IC,
                  PRODUCTIONS,CHAR,J1,J2,J3,NCHAR,RHSNO,LHSBOX,SENTENCE;
        ≠INTEGER≠ CSIZE,ONEFILL4.,

        ≠INTEGER≠ ≠ARRAY≠  SYNG(/1..500,1..5/),
                           LINE(/1..80/),WORD(/1..3/),TITLE[1:80],
                           TERMTABLE(/1..256,1..2/),TERMINALS(/0..256,1..3/),
                           NONTERMTABLE[1:256,1:2],NONTERMINALS[300:557,1:4] ;
 80**


        ≠COMMENT≠ ***********************************************;

        ≠COMMENT≠ THE PROCEDURES GET4 AND PUT4  ALLOW ITEMS OF LENGTH 4 BITS
                  TO BE ENTERED AND RETRIEVED FROM AN ARRAY.THEY ARE
 90**             SPECIFICALLY USED FOR PACKING THE PRECEDENCE MATRIX;

        ≠PROCEDURE≠ PUT4(MAT,ROW,COL,VAL).,
        ≠VALUE≠ ROW,COL,VAL.,       ≠INTEGER≠ ROW,COL,VAL.,
        ≠INTEGER≠ ≠ARRAY≠ MAT.,
        ≠BEGIN≠ ≠INTEGER≠ I,WD1,WD2,POS,WORD,ACOL.,
            ACOL..=(COL-1)//12+1.,
            POS..=COL-(ACOL-1)*12.,
            I..=16**(12-POS).,
            WORD..=MAT(/ROW,ACOL/).,
100**       WD2..=WORD//I.,
            WD1..=WD2//15.,
            WD2..=WORD-WD2*I.,
            MAT(/ROW,ACOL/)..=WD2+(WD1*16+VAL)*I.,
        ≠END≠.,

        ≠INTEGER≠ ≠PROCEDURE≠ GET4(MAT,ROW,COL).,
        ≠VALUE≠ ROW,COL.,   ≠INTEGER≠ ROW,COL.,   ≠INTEGER≠ ≠ARRAY≠ MAT.,
        ≠BEGIN≠ ≠INTEGER≠ I,ACOL, POS.,
110**       ≠INTEGER≠ WD.,
            ACOL..=(COL-1)//12+1.,
            POS..=COL-(ACOL-1)*12.,
            I..=16**(12-POS).,
            WD..=MAT(/ROW,ACOL/)//I.,
            GET4..=WD-WD//15*16.,
```

```
        ≠END≠.,

        ≠COMMENT≠ *************************************************;
 120**
        ≠COMMENT≠ THE PROCEDURES GET9 AND PUT9 ARE SIMILAR TO GET4 AND
                  PUT4. THEY PACK ITEMS OF LENGTH 9 BITS INTO AN ARRAY.
                  THEY ARE USED FOR PACKING THE LEFT AND RIGHT SETS;

        ≠PROCEDURE≠ PUT9(MAT,ROW,COL,VAL).,
        ≠VALUE≠ ROW,COL,VAL.,      ≠INTEGER≠ ROW,COL,VAL.,
        ≠INTEGER≠ ≠ARRAY≠ MAT.,
        ≠BEGIN≠ ≠INTEGER≠ I,WD1,WD2,POS,WORD,ACOL.,
            ACOL..=(COL-1)// 5+1.,
 130**      POS..=COL-(ACOL-1)*5.,
            I..=512**(5-POS).,
            WORD..=MAT(/ROW,ACOL/).,
            WD2..=WORD//I.,
            WD1..=WD2//512.,
            WD2..=WORD-WD2*I.,
            MAT(/ROW,ACOL/)..=WD2+(WD1*512+VAL)*I.,
        ≠END≠.,

 140**  ≠INTEGER≠ ≠PROCEDURE≠ GET9(MAT,ROW,COL).,
        ≠VALUE≠ ROW,COL.,   ≠INTEGER≠ ROW,COL.,   ≠INTEGER≠ ≠ARRAY≠ MAT.,
        ≠BEGIN≠ ≠INTEGER≠ I,ACOL, POS.,
            ≠INTEGER≠ WD.,
            ACOL..=(COL-1)// 5+1.,
            POS..=COL-(ACOL-1)*5.,
            I..=512**(5-POS).,
            WD..=MAT(/ROW,ACOL/)//I.,
            GET9..=WD-WD//512*512.,
        ≠END≠.,
 150**

        ≠COMMENT≠ *************************************************;

        ≠COMMENT≠ THIS PROCEDURE RETURNS INTEGER  ABSOLUTE VALUE;
        ≠INTEGER≠ ≠PROCEDURE≠ IABS(ARG);
        ≠VALUE≠ ARG; ≠INTEGER≠ ARG;
        IABS:=ABS(ARG);

 160**

        ≠COMMENT≠ *************************************************;

        ≠PROCEDURE≠ OUTCHAR(C).,
        ≠INTEGER≠ C.,

        ≠COMMENT≠ PRINTS SYMBOL WITH INTEGER REPRESENTATION -C-;

        OUT CHARACTER(61,≠(≠0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ-<>/,+.≠↱ *();=:
 170**  ∧∨≤≥¬≡$[]↑↓≠)≠,C);
```

```
        ≠COMMENT≠ ************************************************;

        ≠PROCEDURE≠ GETLINE.,
        ≠COMMENT≠ READS A DATA CARD INTO -LINE- AND ECHOES IT ON THE PRINTER;
  180**
            ≠BEGIN≠
            ≠INTEGER≠ J;
            ≠FOR≠ J:=1 ≠STEP≠ 1 ≠UNTIL≠ 80 ≠DO≠
                ≠BEGIN≠

         IN CHARACTER(60,≠(≠0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ-<>/,+.≠↱ *();=:
        ∧∨≤≥¬≡?[]↑↓≠)≠,LINE[J]);

                OUTCHAR(LINE(/J/)).,
  190**
                ≠END≠;

            OUTPUT  (61,≠(≠/≠)≠).,

            IC.=0.,

            ≠END≠ OF PROCEDURE GETLINE;

  200**
        ≠COMMENT≠ ************************************************;

        ≠PROCEDURE≠ PRINTSYMBOL(POS,COUNT);

        ≠INTEGER≠ POS,COUNT;

        ≠COMMENT≠ PRINTS A SYMBOL OF THE GRAMMAR WITH VALUE -POS-,RETURNS A
                  COUNT OF THE NUMBER OF CHARACTERS IN THE SYMBOL;

  210**     ≠BEGIN≠

            ≠INTEGER≠ J,N,T,I2,I3;

            ≠COMMENT≠ CHECK FOR EMPTY SYMBOL;
            ≠IF≠ POS=0∨POS=-1 ≠THEN≠
                ≠BEGIN≠
                ≠IF≠ TERMINALS[0,1] >0 ≠THEN≠ OUTPUT(61,≠(≠≠(≠<EMPTY>≠)≠≠)≠);
                ≠GOTO≠ LL2;
                ≠END≠;
  220**     COUNT:=0;

            ≠COMMENT≠ DECODE SYMBOL;
            ≠FOR≠ J.=1 ≠STEP≠ 1  ≠UNTIL≠ 3 ≠DO≠

                ≠BEGIN≠

                N.=6.,

                ≠COMMENT≠ CHECK IF TERMINAL OR NONTERMINAL;
  230**         T:=≠IF≠ POS>299 ≠THEN≠ NONTERMINALS[POS,J]
                   ≠ELSE≠ TERMINALS[POS,J];
```

```
                ≠IF≠ T=0 ≠THEN≠ ≠GOTO≠ LL1;
                ≠COMMENT≠UNPACK SYMBOL AT A TIME SCANNING FROM LEFT TO RIGHT;
            LL: I3:=J1**N;  I2:=T//I3;

                ≠IF≠ I2>0 ≠THEN≠
                     ≠BEGIN≠
                     OUTCHAR(I2);
                     COUNT:=COUNT+1;
 240**               ≠END≠;

                T.=T-I2*(64**N)., N.=N-1.,
                ≠GOTO≠ ≠IF≠ N<0 ≠THEN≠ LL1 ≠ELSE≠ LL;

           LL1: ≠END≠;

        LL2:≠END≠ OF PROCEDURE PRINTSYMBOL;

 250**
        ≠COMMENT≠ ************************************************;

        ≠INTEGER≠ ≠PROCEDURE≠ NXTCHAR.,

        ≠COMMENT≠ OBTAINS NEXT CHARACTER FROM -LINE-,IF PAST 80 THEN GETS THE
                  NEXT CARD AND RETURNS FIRST SYMBOL;

            ≠BEGIN≠

 260**  L: IC:=IC+1;

            ≠IF≠ IC>80 ≠THEN≠
                ≠BEGIN≠
                GETLINE;
                ≠GOTO≠ L;
                ≠END≠ ≠ELSE≠
            NXTCHAR:=LINE[IC];

            ≠END≠ OF PROCEDURE NXTCHAR;
 270**

        ≠COMMENT≠ ************************************************;

        ≠PROCEDURE≠ GETNONTERM.,

        ≠COMMENT≠ READS NONTERMINAL FROM INPUT,PACKS IT INTO 3 WORDS;

            ≠BEGIN≠
 280**      ≠INTEGER≠ I,WSUM;

            ≠COMMENT≠ INITTIALISE;
            WORD(/3/).=WORD(/2/).=0., IWNO.=LCOUNT.=1.,

            ≠COMMENT≠ PUT FIRST CHARACTER INTO-WORD[1]-;
            WORD(/1/).=CHAR.,

            ≠COMMENT≠ SCAN INPUT UNTIL NON-ALLOWABLE CHARACTER MET;
            ≠FOR≠ I.=NXTCHAR ≠WHILE≠ I <  38 ≠DO≠
```

```
 290**
                ≠BEGIN≠

                ≠COMMENT≠ INCREMENT CHARACTER COUNT;
                LCOUNT.=LCOUNT+1.,

                ≠COMMENT≠ INCREMENT WORD COUNT IF NECESSARY;
                ≠IF≠ LCOUNT >  7 ≠THEN≠ IWNO.=2
                ≠ELSE≠≠IF≠ LCOUNT >  14≠THEN≠ IWNO.=3.,

 300**          ≠COMMENT≠ SHIFT CONTENTS OF CURRENT WORD 1 CHARACTER TO LEFT
                          AND ADD NEW CHARACTER;
                WORD(/IWNO/).=WORD(/IWNO/)*64+I.,

                ≠END≠ ;

            ≠COMMENT≠ FORM HASH CODE -SUM OF 3 WORDS INTO WHICH SYMBOL IS
                      PACKED;
            CHAR.=I.,  WSUM.=WORD(/1/)+WORD(/2/)+WORD(/3/).,
            IC.=IC-1;
 310**
            ≠COMMENT≠ CALL HASH PROCEDURE TO EITHER PUT SYMBOL IN TABLES OR
                      CHECK IF ITS ALDEADY THERE -RETURN SYMBOL VALUE IN -NO-;
            HASH(WSUM,NO,1).,

            ≠END≠ OF PROCEDURE GETNONTERM;


        ≠COMMENT≠ ********************************************************;
 320**
        ≠PROCEDURE≠ GETTERM.,

        ≠COMMENT≠ READS A TERMINAL FROM INPUT AND PACKS INTO 3 WORDS;

            ≠BEGIN≠
            ≠INTEGER≠ I,J,WSUM;

           ≠COMMENT≠ INITIALISE;
           WORD(/1/).=WORD(/2/).=WORD(/3/).=LCOUNT.=0., IWNO.=1.,
 330**
           ≠COMMENT≠ SCAN CHARACTERS UNTIL A > IS MET;
           ≠FOR≠ I.=NXTCHAR ≠WHILE≠ I ¬= 39 ≠DO≠

                ≠BEGIN≠

                ≠COMMENT≠ INCREMENT CHARACTER COUNT;
                LCOUNT.=LCOUNT+1.,

                ≠COMMENT≠ INCREMENT WORD COUNT IF NECESSARY;
 340**          ≠IF≠ LCOUNT >  7 ≠THEN≠ IWNO.=2
                ≠ELSE≠≠IF≠ LCOUNT >  14≠THEN≠ IWNO.=3.,

                ≠COMMENT≠ SHIFT CONTENTS OF CURRENT WORD 1 CHARACTER TO LEFT
                          AND ADD NEW CHARACTER;
                WORD(/IWNO/).=WORD(/IWNO/)*64+I.,

                ≠END≠;
```

```
            ≠COMMENT≠  CHECK FOR SPECIAL CASE OF > BEING THE TERMINAL;
  350**     J!=NXTCHAR;
            ≠IF≠ J=39 ≠THEN≠
                ≠BEGIN≠

                WORD(/IWNO/).=WORD(/IWNO/)*64+J.,

                CHAR.=NXTCHAR ;
                IC!=IC-1;

                ≠END≠ ≠ELSE≠
  360**         ≠BEGIN≠
                CHAR!=J;  IC!=IC-1;
                ≠END≠;

            ≠COMMENT≠  FORM HASH CODE;
            WSUM.=WORD(/1/)+WORD(/2/)+WORD(/3/).,

        ≠COMMENT≠  CALL HASH PROCEDURE TO OBTAIN SYMBOL VALUE;
            HASH(WSUM,NO,0).,

  370**     ≠END≠ OF PROCEDURE GETTERM;

        ≠COMMENT≠ ***********************************************************;

        ≠PROCEDURE≠ HASH(WSUM,NO,S).,
        ≠VALUE≠ S., ≠INTEGER≠ WSUM,NO,S.,

        ≠COMMENT≠ THIS PROCEDURE HASHES A KEY STORED IN -WSUM- AND EITHER
  380**            PUTS SYMBOL INTO TABLES OR LOCATES THE SYMBOL ALREADY IN
                   TABLES, -S- IS A FLAG   1 FOR NONTERMIALS, 0 FOR TERMINALS,
                   THE SYMBOL VALUE IS RETURND IN -NO-;

        ≠COMMENT≠ HASH SCHEME USED GIVEN IN
                       R.MORRIS  SCATTER STORAGE TECHNIQUES,
                       CACM 11 (1) JAN,1968 PP 38-44 RANDOM PROBING
                       TECHNIQUE;

            ≠BEGIN≠
  390**     ≠INTEGER≠ N1,N2,NPROBE,KRAND,IHASH,I1,I2,IPTR,IKEY;

            ≠COMMENT≠  SET TABLE SIZE   2**N;
            N1!=8;
            N2!=J2**N1;
            NPROBE.=KRAND.=1.,

            ≠COMMENT≠  CALCULATE HASH KEY FROM -WSUM-;
           IHASH.=WSUM+WSUM//N2;
  400**
            ≠COMMENT≠  CALCULATE POINTER INTO TABLE
                       =MOD(IHASH+KRAND/4,2**N)+1;
        L1! I1!=IHASH+KRAND//J3;
            I2!=2**N1;
            IPTR.=(I1-(I1//I2)*I2)+1.,
```

```
            ≠COMMENT≠  CHECK IF PROCESSING TERMINAL OR NONTERMINAL;
            ≠IF≠ S =   1 ≠THEN≠ ≠GOTO≠ LAB1.,
 410**
            ≠COMMENT≠        TERMINAL ;

            ≠COMMENT≠ OBTAIN KEY ALREADY STORED AT POSITION -IPTR-;
            IKEY.=TERMTABLE(/IPTR,1/).,

            ≠COMMENT≠ COMPARE WITH -WSUM-;
            ≠IF≠ IKEY =  WSUM ≠THEN≠
                ≠BEGIN≠
 420**
                ≠COMMENT≠ ENTRY ALREADY IB TABLE -RETURN SYMBOL VALUE;
                NO.=TERMTABLE(/IPTR,2/).,
                ≠GOTO≠ ENL.,

                ≠END≠ ≠ELSE≠
            ≠IF≠ IKEY=0 ≠THEN≠

                ≠BEGIN≠
 430**          ≠COMMENT≠ ENTRY NOT IN TABLE, UPDATE -TERMNO-,PUT KEY AND
                          POINTER INTO -TERMTABLE-, PUT COPY OF SYMBOL INTO
                          -TERMINALS- ,FINISH;
                TERMNO.=TERMNO+1.,
                TERMTABLE(/IPTR,1/).=WSUM.,
                TERMTABLE(/IPTR,2/).=TERMNO.,
                NO.=TERMNO.,
                ≠FOR≠ I.=1≠STEP≠1≠UNTIL≠ 3 ≠DO≠
                TERMINALS(/TERMNO,I/).=WORD(/I/).,
                ≠GOTO≠ ENL;
 440**
                ≠END≠ ≠ELSE≠
                ≠BEGIN≠

                ≠COMMENT≠ COLLISION, CALCULATE RANDOM OFFSET
                           KRAND=MOD(5*KRAND,2**(N+2));
                I1.=2**(N1+2).,
                I2.=5*KRAND.,
                KRAND.= I2-(I2//I1)*I1.,
 450**          ≠COMMENT≠ GO TO RECALCULATE POINTER;
               ≠GOTO≠ L1.,
                ≠END≠ ;

            ≠COMMENT≠   NONTERMINAL,

                        GO THROUGH EXACTLY THE SAME PROCEDURE AS FOR TERMINALS;
 460** LAB1..   IKEY.=NONTERMTABLE(/IPTR,1/).,
            ≠IF≠ IKEY =  WSUM ≠THEN≠
```

```
         ≠BEGIN≠
         ≠COMMENT≠ PRINT OUT NON-TERMINALS.,
         OUTPUT  (61,≠(≠/,/,54B,≠(≠N O N  T E R M I N A L S≠)≠/,/,58B,
                             ≠(≠NUMBER       SYMBOL≠)≠≠)≠).,
         ≠FOR≠ I.=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
             ≠BEGIN≠
 530**       OUTPUT   (61,≠(≠/,60BZZZD,5B≠)≠,I).,
             PRINTSYMBOL(I,NCHAR);
             ≠END≠;
         ≠COMMENT≠ PRINT OUT TERMINALS;
         OUTPUT  (61,≠(≠/,/,57B,≠(≠T E R M I N A L S≠)≠/,/,58B,
                             ≠(≠NUMBER       SYMBOL≠)≠≠)≠).,
         ≠IF≠ TERMINALS[0,1]>0 ≠THEN≠
             ≠BEGIN≠
 540**       OUTPUT(61,≠(≠/,63B,≠(≠-1≠)≠,5B≠)≠);
             PRINTSYMBOL(0,NCHAR);
             ≠END≠;
         ≠FOR≠ I.=1 ≠STEP≠ 1 ≠UNTIL≠ NTERM ≠DO≠
             ≠BEGIN≠
             OUTPUT   (61,≠(≠/,60BZZZD,5B≠)≠,I);
             PRINTSYMBOL(I,NCHAR);
             ≠END≠;
 550**   ≠COMMENT≠ PRINT SYNTAX GRAPH;
         OUTPUT  (61,≠(≠↑16B,≠(≠ SYNTAX GRAPH≠)≠≠)≠).,
         OUTPUT(61,≠(≠/,17B,≠(≠------------≠)≠/,/≠)≠);
         OUTPUT(61,≠(≠6B,≠(≠ SYMBOL  DEFN. ALT. SUC. LHSSYM≠)≠,/,
               6B,     ≠(≠   NO.≠)≠,/≠)≠);
         NGRAPH.=NXTBOX-1.,
         ≠FOR≠ I.=1 ≠STEP≠ 1 ≠UNTIL≠ NGRAPH ≠DO≠
             ≠BEGIN≠
 560**       OUTPUT(61,≠(≠/,ZZD,3B≠)≠,I);
             OUTPUT(61,≠(≠ B-ZZZD,4B,-ZZD,BZZZD,BZZZD,3B,ZZZD≠)≠,SYNG[I,1],
                     SYNG(/I,2/),SYNG(/I,3/),SYNG(/I,4/),SYNG[I,5]);
             OUTPUT(61,≠(≠10B≠)≠); PRINTSYMBOL(SYNG[I,1],NCHAR);
             ≠END≠;
         ≠END≠ OF PRINT TABLES;
 570**
      ≠COMMENT≠ ***********************************************;
      ≠PROCEDURE≠ INGRAMMAR;
      ≠COMMENT≠ THIS PROCEDURE READS IN THE GRAMMAR IN FREE FORMAT, REVERSE BNF
                FROM CARDS, FORMS SYNTAX GRAPH AND SYMBOL TABLES AND PRINTS
```

```
 580**           SYMBOLS AND SYNTAX GRAPH;
            ≠BEGIN≠
            ≠COMMENT≠ INTIALISE ALL SYMBOL TABLES AND SYNTAX GRAPH TO 0;
            ≠FOR≠ I.=1 ≠STEP≠ 1 ≠UNTIL≠ 256 ≠DO≠
           ≠FOR≠ J.=1 ≠STEP≠ 1 ≠UNTIL≠ 2 ≠DO≠
            NONTERMTABLE(/I,J/).=TERMTABLE(/I,J/).=0.,
            TERMINALS[0,1]:=0;
 590**
            ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ 557 ≠DO≠
            NONTERMINALS[I,4]:=0;

            ≠FOR≠ I.=1 ≠STEP≠ 1 ≠UNTIL≠ 500 ≠DO≠
            ≠FOR≠ J.=1 ≠STEP≠ 1 ≠UNTIL≠ 5 ≠DO≠
            SYNG(/I,J/).=0.,

            PRODUCTIONS.=0.,
 600**      ≠COMMENT≠ INITIALISE POINTERS AND COUNTERS;
            NXTBOX.=1.,NTERMNO.=299; TERMNO:=0;

            OUTPUT(61,≠(≠20B,≠(≠*INPUT*≠)≠,/,/,/≠)≠);

            ≠COMMENT≠ GET FIRST LINE OF INPUT;
            GETLINE.,

            ≠COMMENT≠ THIS CODING PROCESSES LHS,
 610**                START BY SCANNING FOR LETTER;
         ENTRY:
            ≠FOR≠ I:=NXTCHAR ≠WHILE≠ I=46 ≠DO≠: CHAR:=I;
            ≠COMMENT≠ CHECK FOR ENDING . OR ERROR;
            ≠IF≠ I=43 ≠THEN≠ ≠GOTO≠ FIN ≠ELSE≠
            ≠IF≠ CHAR>37 ≠THEN≠
                ≠BEGIN≠
               OUTPUT(61,≠(≠5B,≠(≠*ERROR*≠)≠,/≠)≠);
 620**          ≠GOTO≠ ENTRY;
                ≠END≠;

            GETNONTERM:
            PRODUCTIONS:=PRODUCTIONS+1; RHSNO:=0;
            ≠COMMENT≠ IF FIRST PRODUCTION THEN FILL IN FIRST BOX OF GRAPH
                   ELSE CHECK BACK FOR PREVIOUS OCCURENCE OF LHS TO FILL IN
                   DEFINITION;
            ≠IF≠ PRODUCTIONS=1 ≠THEN≠
                ≠BEGIN≠
 630**          SENTENCE:=300;
                SYNG[1,1]:=NO;
                DEFBOX:=NXTBOX:=SYNG[1,2]:=2;
                NONTERMINALS[NO,4]:=2;
                LHSBOX:=300;
                ≠END≠ ≠ELSE≠
                ≠BEGIN≠
                ≠COMMENT≠ PERFORM CHECKS TO SEE WHETHER NONTERM
```

```
                              HAS BEEN MET BEFORE AND/OR DEFINED ;
                    ≠IF≠ NONTERMINALS[NO,4]=0 ≠THEN≠
  640**                   ≠BEGIN≠
                          ≠COMMENT≠ MET FOR FIRST TIME AS A LHS;
                          NONTERMINALS[NO,4]:=NXTBOX;
                          SYNG[NXTBOX,5]:=NO;
                          ≠END≠ ≠ELSE≠
                          ≠BEGIN≠
                          ≠IF≠ NONTERMINALS[NO,4]>10000 ≠THEN≠
                              ≠BEGIN≠
                              ≠COMMENT≠ MET BEFORE IN A RHS BUT NOT DEFINED ;
                              I:=NONTERMINALS[NO,4]-10000;
  650**                       NONTERMINALS[NO,4]:=NXTBOX;
            TRACEBACK:        J:=SYNG[I,2];
                              SYNG[I,2]:=NXTBOX;
                              I:=J; ≠IF≠ J¬=-1 ≠THEN≠ ≠GOTO≠ TRACEBACK;
                              SYNG[NXTBOX,5]:=NO;
                              ≠END≠ ≠ELSE≠
                              ≠BEGIN≠
                              ≠COMMENT≠ MET AND DEFINED BEFORE,NOW MET AGAIN
                                        AS AN ALTERNATE ;
                              J:=IABS(NONTERMINALS[NO,4]);
  660**     TRACEALT:         I:=J; J:=SYNG[I,3];
                              ≠IF≠ J¬=0 ≠THEN≠ ≠GOTO≠ TRACEALT;
                              SYNG[I,3]:=NXTBOX;
                              SYNG[NXTBOX,5]:=NO;
                              ≠END≠
                          ≠END≠
                    ≠END≠;
              LHSBOX:=NO;
              DEFBOX:=NXTBOX;
              ≠COMMENT≠ STATE2 TAKES IN FIRST COMPLETE RHS AND ALL SYMBOLS
  670**                 AFTER A FIRST ALTERNATE SYMBOL;

          STATE2:

              ≠FOR≠ I:=NXTCHAR ≠WHILE≠ I=46 ≠DO≠; CHAR:=I;
              ≠GOTO≠ ≠IF≠ I=45∨I=52∨I=51 ≠THEN≠ STATE2
                      ≠ELSE≠ ≠IF≠ I=40 ≠THEN≠ STATE3
                      ≠ELSE≠ ≠IF≠ I=50 ≠THEN≠ ENDCHECK
                      ≠ELSE≠ ≠IF≠ I=43 ≠THEN≠ FIN
                      ≠ELSE≠ GOON1;
  680**
          GOON1: ≠IF≠ I=38 ≠THEN≠ GETTERM ≠ELSE≠ GETNONTERM;
              RHSNO:=RHSNO+1;
              ≠COMMENT≠ CHECK BACK FOR PREVIOUS OCCURENCES OF SYMBOL IF ITS
                        A NON-TERMINAL TO FILL IN ITS DEF. POINTER;
              ≠IF≠ NXTBOX>2∧NO>299 ≠THEN≠
                  ≠BEGIN≠
                  ≠IF≠ NONTERMINALS[NO,4]=0 ≠THEN≠
                      ≠BEGIN≠
                      SYNG[NXTBOX,2]:=-1 ;
  690**               NONTERMINALS[NO,4]:=10000+NXTBOX;
                      ≠END≠ ≠ELSE≠
                      ≠BEGIN≠
                      ≠IF≠ NONTERMINALS[NO,4]>10000 ≠THEN≠
                          ≠BEGIN≠
                          SYNG[NXTBOX,2]:=NONTERMINALS[NO,4]-10000;
```

```
                          NONTERMINALS[NO,4]:=NXTBOX+10000;
                          ≠END≠ ≠ELSE≠
                      SYNG[NXTBOX,2]:=IABS(NONTERMINALS[NO,4]);
                      ≠END≠;
 700**            ≠END≠;
              SYNG[NXTBOX,1]:=NO;
              ≠COMMENT≠ FILL IN SUCCESSOR PTR. OF PREVIOUS BOX;
              ≠IF≠ NXTBOX>DEFBOX ≠THEN≠ SYNG[NXTBOX-1,4]:=NXTBOX;
              ≠COMMENT≠ UPDATE FREE BOX NO. AND GET NEXT SYMBOL;
              NXTBOX:=NXTBOX+1;
              ≠GOTO≠ STATE2;

              ≠COMMENT≠ ENDCHECK CHECKS FOR THE EMPTY STATEMENT AND LINKS THE
                         RHS PTR. BOX;
 710**
          ENDCHECK:
              ≠IF≠ RHSNO=0 ≠THEN≠ EMPTYPROD;
              SYNG[DEFBOX,5]:=LHSBOX;
              ≠GOTO≠ ENTRY;


              ≠COMMENT≠ STATE3 PROCESSES SYMBOL AFTER A / ;

         STATE3:
 720**
              SYNG[DEFBOX   ,5]:=LHSBOX;
              ≠FOR≠ I:=NXTCHAR ≠WHILE≠ I=46 ≠DO≠; CHAR:=I;
              ≠COMMENT≠ CHECK FOR EMPTY PROD / / ;
              ≠IF≠ I=40 ≠THEN≠
                  ≠BEGIN≠
                  EMPTYPROD;
                  SYNG[DEFBOX,3]:=NXTBOX-1;
                  DEFBOX:=NXTBOX-1;
                  ≠GOTO≠ STATE2;
 730**            ≠END≠ ≠ELSE≠
                ≠IF≠ I=43vI=50 ≠THEN≠
                  ≠BEGIN≠
                  EMPTYPROD;
                  SYNG[DEFBOX,3]:=NXTBOX-1;
                  ≠GOTO ≠ ENTRY;
                  ≠END≠ ≠ELSE≠
              ≠IF ≠ I=38 ≠THEN≠ GETTERM ≠ELSE≠ GETNONTERM;

              RHSNO:=1;
 740**        PRODUCTIONS:=PRODUCTIONS+1;
              ≠COMMENT≠ FILL IN DEF. BOX BY CHECKING BACK FOR PREVIOUS OCCURENCE;
              ≠IF≠ NXTBOX>2^NO>299 ≠THEN≠
                  ≠BEGIN≠
                  ≠IF≠ NONTERMINALS[NO,4]=0 ≠THEN≠
                      ≠BEGIN≠
                      SYNG[NXTBOX,2]:=-1 ;
                      NONTERMINALS[NO,4]:=10000+NXTBOX;
                      ≠END≠ ≠ELSE≠
                      ≠BEGIN≠
 750**                ≠IF≠ NONTERMINALS[NO,4]>10000 ≠THEN≠
                          ≠BEGIN≠
                          SYNG[NXTBOX,2]:=NONTERMINALS[NO,4]-10000;
                          NONTERMINALS[NO,4]:=NXTBOX+10000;
```

```
                        ≠END≠ ≠ELSE≠
                    SYNG[NXTBOX,2]:=IABS(NONTERMINALS[NO,4]);
                    ≠END≠;
                ≠END≠;
            ≠COMMENT≠ FILL IN OTHER POINTERS;
            SYNG[NXTBOX,1]:=NO;
  760**     SYNG[DEFBOX,3]:=NXTBOX;
            DEFBOX:=NXTBOX;
            NXTBOX:=NXTBOX+1;
            ≠GOTO≠ STATE2;

            ≠COMMENT≠ END OF INPUT;

         FIN..

  770**     SYNG[DEFBOX  ,5]:=LHSBOX;

            ≠COMMENT≠ PRINT NUMBER OF SYMBOLS,PRODUCTIONS ETC;
            OUTPUT(61,≠(≠↑≠)≠);
            OUTPUT  (61,≠(≠/,/,10B,≠(≠NUMBER OF PRODUCTIONS ≠)≠,ZZD,/,
                               10B,≠(≠NUMBER OF NON TERMINALS ≠)≠,ZZD,/,
                               10B,≠(≠NUMBER OF TERMINALS ≠)≠,ZZD≠)≠,
                                        PRODUCTIONS,NTERMNO-299,TERMNO);
            OUTPUT(61,≠(≠/,/,10B,≠(≠THE SENTENCE SYMBOL FOR THE GRAMMAR IS: ≠)≠
  780**  ≠)≠);
            PRINTSYMBOL(SENTENCE,NCHAR);

             NTERM:=TERMNO;
            NNONTERM.=NTERMNO-299;
            SIZE.=NTERM+NNONTERM.,
                CSIZE.=(SIZE-1)//12+1.,

            ≠COMMENT≠ CLEAR LISTS OF UNDEFINED NONTERMINALS-SET TO ZERO;
  790**     ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
            ≠IF≠ NONTERMINALS[I,4]>10000 ≠THEN≠
                ≠BEGIN≠
                OUTPUT(61,≠(≠/,5B,≠(≠THE FOLLOWING NONTERMINAL IS UNDEFINED
         ≠)≠≠)≠);
                PRINTSYMBOL(I,NCHAR);
                J:=NONTERMINALS[I,4]-10000;
                NONTERMINALS[I,4]:=0;
        TRACE: I1:=SYNG[J,2];
                SYNG[J,2]:=1;
  800**         J:=I1;
                ≠IF≠ J¬=-1 ≠THEN≠ ≠GOTO≠ TRACE ;
                ≠END≠;


            ≠END≠ OF PROCEDURE INGRAMMAR;

  810**
        ≠COMMENT≠ ****************************************************;
```

```
          ≠PROCEDURE≠ PRINTGRAMMAR;
              ≠BEGIN≠
              ≠INTEGER≠ I,RHSPTR,J,J1,JSPACE,NCHAR,K;
   820**      OUTPUT(61,≠(≠↑≠)≠);  OUTPUT(61,≠(≠/,/,28B≠)≠);
               ≠FOR≠ I:=1 ≠STEP≠ 1 ≠UNTIL≠ 80 ≠DO≠ OUTCHAR(TITLE[I]);
              ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
                  ≠BEGIN≠
                  OUTPUT(61,≠(≠/,/≠)≠);
                  RHSPTR:=NONTERMINALS[I,4];
                  ≠IF≠ RHSPTR<0 ≠THEN≠ OUTPUT(61,≠(≠2B,≠(≠$≠)≠,2B≠)≠) ≠ELSE≠
                  OUTPUT(61,≠(≠5B≠)≠);
                  PRINTSYMBOL(I,NCHAR);
   830**          JSPACE:=21-NCHAR;
                  ≠FOR≠ K:=1 ≠STEP≠ 1 ≠UNTIL≠ JSPACE ≠DO≠ OUTPUT(61,≠(≠B≠)≠);
                  OUTPUT(61,≠(≠≠(≠ ::= ≠)≠≠)≠);
                  J:=IABS(RHSPTR); J1:=J;
                  ≠IF≠ J=0 ≠THEN≠
                      ≠BEGIN≠
                    OUTPUT(61,≠(≠≠(≠(NO RIGHT SIDE DEFINED IN THE GRAMMAR)≠)≠≠)≠);
                      ≠GOTO≠ NEXTI;
                      ≠END≠;
   840**     RHSP: ≠IF≠ SYNG[J,1]=-1 ≠THEN≠ ≠GOTO≠ TRACE ≠ELSE≠
                  ≠IF≠ SYNG[J,2]=0 ≠AND≠ SYNG[J,1]<300 ≠THEN≠
                      ≠BEGIN≠
                      OUTPUT(61,≠(≠≠(≠<≠)≠≠)≠);
                      PRINTSYMBOL(SYNG[J,1],NCHAR);
                      OUTPUT(61,≠(≠≠(≠>≠)≠,2B≠)≠);
                      ≠END≠  ≠ELSE≠
                      ≠BEGIN≠
                      PRINTSYMBOL(SYNG[J,1],NCHAR);
                      OUTPUT(61,≠(≠2B≠)≠);
   850**              ≠END≠;
             TRACE: J:=SYNG[J,4];
                  ≠IF≠ J¬=0 ≠THEN≠ ≠GOTO≠ RHSP;
                  J:=SYNG[J1,3]; J1:=J;
                  ≠IF≠ J¬=0 ≠THEN≠
                      ≠BEGIN≠
                      OUTPUT(61,≠(≠/,31B≠)≠);
                      ≠GOTO≠ RHSP;
                      ≠END≠;
   860**
            NEXTI:
                  ≠END≠ OF MAIN LOOP;
              ≠END≠ OF PRINTGRAMMAR;
```

```
  870** ≠COMMENT≠*******************************************************;

        ≠PROCEDURE≠ TERMINATE(SYNG,NON,N,S);
        ≠VALUE≠ N;       ≠INTEGER≠ ≠ARRAY≠ SYNG,NON;
        ≠BOOLEAN≠ ≠ARRAY≠ S; ≠INTEGER≠ N;
        ≠BEGIN≠ ≠BOOLEAN≠ E;  ≠INTEGER≠ I,J,K;
            E:=≠FALSE≠;
        INITIAL:
            ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ N ≠DO≠
  880**     ≠BEGIN≠ J:=NON[I,4];
                ≠IF≠ J<0 ≠THEN≠ S[I]:=E:=≠TRUE≠ ≠ELSE≠ S[I]:=≠FALSE≠
            ≠END≠;
            ≠IF≠ E ≠THEN≠
        ITERATE:
            ≠BEGIN≠ E:=≠FALSE≠;
                ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ N ≠DO≠
                ≠IF≠ ¬S[I] ≠THEN≠
                ≠BEGIN≠  ≠COMMENT≠ SEARCH RULES OF THIS NT;
                    K:=J:=NON[I,4];
  890**             ≠IF≠ J<0 ≠THEN≠ E:=S[I]:=≠TRUE≠ ≠ELSE≠
                    ≠IF≠ J¬=0 ≠THEN≠
                    ≠BEGIN≠ ≠COMMENT≠ J NOW POINTS TO FIRST SYMBOL OF FIRST
                                   RHS OF THE RULES OF I;
                     SUC:≠IF≠ SYNG[K,2]¬=0 ≠THEN≠
                        ≠BEGIN≠ ≠IF≠ S[SYNG[K,1]] ≠THEN≠
                    SUC1:
                            ≠BEGIN≠ K:=SYNG[K,4];
                                ≠IF≠ K¬=0 ≠THEN≠ ≠GOTO≠ SUC ≠ELSE≠
                                ≠BEGIN≠ E:=S[I]:=≠TRUE≠; ≠GOTO≠ FORFIN; ≠END≠;
  900**                     ≠END≠;
                        ≠END≠ ≠ELSE≠ ≠GOTO≠ SUC1;
                        ≠COMMENT≠ MOVE TO NEXT ALTERNATIVE;
                        K:=J:=SYNG[J,3];
                        ≠IF≠ J¬=0 ≠THEN≠ ≠GOTO≠ SUC;
                    ≠END≠;
          FORFIN:
                ≠END≠ FOR LOOP;
                ≠IF≠ E ≠THEN≠ ≠GOTO≠ ITERATE;
            ≠END≠;
  910** ≠END≠ OF TERMINATE;

        ≠PROCEDURE≠ DELETE(SYNG,NON,N,SYM);
        ≠VALUE≠ N,SYM;  ≠INTEGER≠ N,SYM;
        ≠INTEGER≠ ≠ARRAY≠ SYNG,NON;
  920** ≠BEGIN≠ ≠INTEGER≠ I,J,K,L,P;  ≠BOOLEAN≠ FIRST;
            ≠IF≠ SYM<300 ≠THEN≠ ≠GOTO≠ DEL;
            P:=IABS(NON[SYM,4]);
            ≠IF≠ P¬=0 ≠THEN≠
            ≠BEGIN≠ NON[SYM,4]:=0;
                ≠COMMENT≠ NOW DELETE ALL RHSS THAT CONTAIN SYM;
          DEL:  ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ N ≠DO≠
                ≠BEGIN≠  K:=J:=IABS(NON[I,4]);  FIRST:=≠TRUE≠;
```

```
                    ≠IF≠ J¬=0 ≠THEN≠
         SUC:       ≠BEGIN≠ ≠IF≠ SYNG[K,1]=SYM ≠THEN≠
  930**                   ≠BEGIN≠  ≠COMMENT≠ DELETE THIS ALT;
                               K:=J:=SYNG[J,3];
                               ≠IF≠ FIRST ≠THEN≠
                               ≠BEGIN≠
                                   ≠IF≠ J=0 ≠THEN≠ DELETE(SYNG,NON,N,I) ≠ELSE≠
                                       NON[I,4]:=J*SIGN(NON[I,4]);
                               ≠END≠ ≠ELSE≠ SYNG[L,3]:=J;
                               ≠GOTO≠ ≠IF≠ J¬=0 ≠THEN≠ SUC ≠ELSE≠ FORFIN;
                          ≠END≠ ≠ELSE≠
                          ≠BEGIN≠  K:=SYNG[K,4];
  940**                        ≠IF≠ K¬=0 ≠THEN≠ ≠GOTO≠ SUC;
                          ≠END≠ TRY NEXT ALT;
                          FIRST:=≠FALSE≠;  L:=J;
                          K:=J:=SYNG[J,3];
                          ≠IF≠ J¬=0 ≠THEN≠ ≠GOTO≠ SUC;
                    ≠END≠ OF J TEST;
     FORFIN: ≠END≠;
         ≠END≠ P TEST;
     ≠END≠ OF DELETE;

  950**
     ≠PROCEDURE≠ ADMISSABLE(OCH,SYNG,NON,N,SENTENCE);
     ≠VALUE≠ OCH,N,SENTENCE;  ≠INTEGER≠ OCH,N,SENTENCE;
     ≠INTEGER≠ ≠ARRAY≠ SYNG,NON;
     ≠BEGIN≠  ≠BOOLEAN≠ ≠ARRAY≠ T[300:N];
     ≠INTEGER≠ I,J,K,PDPT,SK1,M;
     ≠INTEGER≠ ≠ARRAY≠ PD[300:N+1];
         TERMINATE(SYNG,NON,N,T);
         ≠COMMENT≠ NOW DELETE THOSE NONTERMINALS AND RULES THAT CANNOT
         GENERATE A TERMINAL WORD;
  960**  OUTPUT(OCH,≠(≠/,/,/,5B,≠(≠USELESS NON-TERMINALS≠)≠,/,5B,
         21(≠(≠-≠)≠),/≠)≠);
         ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠  N ≠DO≠
         ≠IF≠ ¬T[I] ≠THEN≠ ≠BEGIN≠
                                OUTPUT(OCH,≠(≠5B≠)≠);
                                PRINTSYMBOL(I,NCHAR);
                                DELETE(SYNG,NON,N,I);
                                ≠END≠;
         PRINTGRAMMAR;
         ≠COMMENT≠ NOW CHECK NONTERMINALS FOR REACHABILITY;
  970** INIT:
         PDPT:=300;    PD[PDPT]:=SENTENCE;
         ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ N ≠DO≠
             ≠BEGIN≠ T[I]:=≠FALSE≠;  PD[I+1]:=0; ≠END≠;
         T[SENTENCE]:=≠TRUE≠;
         ≠FOR≠ M:=300 ≠STEP≠ 1 ≠UNTIL≠ PDPT ≠DO≠
         ≠BEGIN≠  I:=PD[M];  K:=J:=IABS(NON[I,4]);
             ≠IF≠ K¬=0 ≠THEN≠
        SUC:
             ≠BEGIN≠  ≠IF≠ SYNG[K,2]¬=0 ≠THEN≠
  980**            ≠BEGIN≠  SK1:=SYNG[K,1];
                       ≠IF≠¬T[SK1] ≠THEN≠
                       ≠BEGIN≠  ≠COMMENT≠ ADD THIS NONTERMINAL TO PD LIST;
                           PDPT:=PDPT+1;  T[SK1]:=≠TRUE≠;
                           PD[PDPT]:=SK1;
                       ≠END≠;
```

```
                          ≠END≠;
                              K:=SYNG[K,4];
                              ≠IF≠ K¬=0 ≠THEN≠ ≠GOTO≠ SUC ≠ELSE≠
                              ≠BEGIN≠ K:=J:=SYNG[J,3];
  990**                            ≠IF≠ J¬=0 ≠THEN≠ ≠GOTO≠ SUC;
                              ≠END≠;
                    ≠END≠;
               ≠END≠ OF THIS NONTERMINAL;
               ≠COMMENT≠ T CONTAINS THOSE NONTERMINALS THAT ARE REACHABLE.  NOW
               DELETE THE UNREACHABLE NONTERMINALS:
               OUTPUT(OCH,≠(≠/,/,/,5B,≠(≠UNREACHABLE NON-TERMINALS≠)≠,/,5B,
               25(≠(≠-≠)≠),/≠)≠);
               ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ N ≠DO≠
               ≠IF≠ ¬T[I] ≠THEN≠ ≠BEGIN≠
 1000**                                OUTPUT(OCH,≠(≠5B≠)≠);
                                       PRINTSYMBOL(I,NCHAR);
                                       DELETE(SYNG,NON,N,I);
                                       ≠END≠;
               PRINTGRAMMAR;
               ≠IF≠ ¬T[SENTENCE] ≠THEN≠ OUTPUT(OCH,
                    ≠(≠*,//,≠(≠///// SENTENCE DELETED/////≠)≠≠)≠);
          ≠END≠ OF ADMISSABLE:

 1010** ≠COMMENT≠*************************************************;



          ≠COMMENT≠**************************************************************
 1020**
                                  M A I N   P R O G R A M
                        ***************************************************************;

          ≠COMMENT≠ PRINT TITLE;

          ≠FOR≠ I:=1 ≠STEP≠ 1 ≠UNTIL≠ 10 ≠DO≠ OUTPUT(61,≠(≠/≠)≠);
          OUTPUT(61,≠(≠28B≠)≠);
 1030** GETLINE;  ≠FOR≠ I:=1 ≠STEP≠ 1 ≠UNTIL≠ 80 ≠DO≠ TITLE[I]:=LINE[I];
          OUTPUT(61,≠(≠28B≠)≠);
          ≠FOR≠ I:=1 ≠STEP≠ 1 ≠UNTIL≠ 80 ≠DO≠ OUTPUT(61,≠(≠≠(≠-≠)≠≠)≠);

          OUTPUT(61,≠(≠↑≠)≠);

          ≠COMMENT≠ SET CONSTANTS USED IN PROGRAM;
          J1.=64; J2.=2;  J3.=4;
               ONEFILL4.=0.,
 1040**        ≠FOR≠ I.=1 ≠STEP≠ 1 ≠UNTIL≠ 12 ≠DO≠
                    ONEFILL4.=ONEFILL4*16+1.,
```

```
         ≠COMMENT≠  ************  CALL PROCEDURES TO BE USED  *******************;
         ≠COMMENT≠              *************************;
                                  INGRAMMAR;
 1050**  ≠COMMENT≠              *************************;
                                  PRINT TABLES;
         ≠COMMENT≠              *************************;
                                  PRINTGRAMMAR;
         ≠COMMENT≠              *************************;
                                  ADMISSABLE(61,SYNG,NONTERMINALS,NTERMNO,300) ;
 1060**
         ≠COMMENT≠              *************************;
         ≠COMMENT≠              *************************;


         ≠COMMENT≠***********************************************************
                  ***********************************************************;
 1070**
             ≠COMMENT≠ THIS IS THE SECOND MAIN BLOCK OF GRAMPA CONTAINING
                      THE PROCEDURES FOR PRECEDENCE ANALYSIS ETC;

         ≠COMMENT≠***********************************************************
                  ***********************************************************;

             ≠BEGIN≠
 1080**      ≠COMMENT≠ IDENTIFIERS USED
                   - PMATRIX -   SQUARE ARRAY HOLDING THE PRECEDENCE MATRIX,
                   - LR -        ARRAY FOR THE LEFT AND RIGHT SETS,
                   - CONFLICTABLE -  HOLDS THE TYPE AND SYMBOLS INVOLVED IN
                                     PRECEDENCE CONFLICTS,
                   - NCONFL -    A COUNT OF THE NUMBER OF CONFLICTS;

             ≠INTEGER≠ ≠ARRAY≠ PMATRIX(/1..SIZE+20,1..(SIZE+19)//12+1/),
                           LR(/300..NTERMNO+20,1..2*((SIZE+19)//5+1)/),
 1090**                          CONFLICTABLE[1:20,1:5];
             ≠INTEGER≠ NCONFL;

             ≠COMMENT≠*******************************************;

             ≠PROCEDURE≠ EMPTYCHECK(EXITLABEL);
             ≠LABEL≠ EXITLABEL;
             ≠COMMENT≠ THIS PROCEDURE SCANS THE 4TH COLUMN OF THE ARRAY
                      -NONTERMINALS- TO CHECK FOR -VE VALUES INDICATING
 1100**               THAT THE EMPTY STATEMENT EXISTS IN THE LHS;
```

```
            ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
                ≠IF≠ NONTERMINALS[I,4]<0 ≠THEN≠
                  ≠BEGIN≠
                  OUTPUT(61,≠(≠/,/,/,/,≠(≠ THE EMPTY PRODUCTION EXISTS IN THE GR
         AMMAR...≠)≠,/,≠(≠   NO LEFT AND RIGHT PARTS,RECURSION OR PRECEDENCE WILL
          BE COMPUTED≠)≠≠)≠);
                   ≠GOTO≠ EXITLABEL;
                   ≠END≠ OF CHECK AND PROCEDURE EMPTY CHECK;
 1110**

            ≠COMMENT≠*******************************************************;

            ≠PROCEDURE≠ LEFTRIGHT.,

            ≠COMMENT≠ THIS PROCEDURE FINDS THE LEFT AND RIGHT SETS OF THE
                      NONTERMINAL SYMBOLS;
 1120**         ≠BEGIN≠

                ≠INTEGER≠ ≠ARRAY≠ PATH[1..2,1..NGRAPH],INO[1..2];
                ≠INTEGER≠ I,J,DEF,NL,NR,PC,LC.,

                ≠COMMENT≠*************************************************;

                ≠PROCEDURE≠ LDEF(DEF)., ≠INTEGER≠ DEF.,
 1130**         ≠COMMENT≠ THIS PROCEDURE FINDS THE LEFT SET OF A NON-TERM
                          WHOSE RHS STARTS AT ROW -DEF-;

                    ≠BEGIN≠

                    ≠INTEGER≠ I,DEF1;

                    ≠COMMENT≠ GET VALUE OF SYMBOL AT ROW -DEF- ;
                    LC.=SYNG[DEF,1];
                    ≠COMMENT≠ IF THE SET L IS EMPTY,THEN INSERT SYMBOL;
 1140**             ≠COMMENT≠ IF THE SYMBOL IS ALREADY IN L,THEN CARRY ON AND
                              CHECK ITS RHS;
                    ≠FOR≠ I.=1 ≠STEP≠ 1 ≠UNTIL≠ NL≠DO≠
                    ≠IF≠ GET9(LR,PC,I)=LC ≠THEN≠ ≠GOTO≠ L3.,
                    ≠COMMENT≠ INSERT SYMBOL INTO L;
                NL.=NL+1.,
                PUT9(LR,PC,NL,LC).,

                    ≠COMMENT≠  CHECK THAT SYMBOL AT ROW -DEF- IS A NON-TERM;
 1150**         L3: ≠IF≠ SYNG[DEF,2]¬=0 ≠THEN≠

                        ≠BEGIN≠
                        ≠COMMENT≠ GET FIRST SYMBOL OF RHS OF NON-TERM IN
                                  ROW -DEF-;
                        DEF1.=SYNG(/DEF,2/).,
                        ≠COMMENT≠ CHECK IF THIS SYMBOL HAS ALREDY BEEN
                                  SCANNED WHILE FINDING THE LEFT SET OF THE
                                  ORIGINAL SYMBOL;
                        PATHCHK(DEF1,L4,1).,
```

```
 1160**                   ≠COMMENT≠ ADD SYMBOL TO PATH;
                          INO(/1/).=INO(/1/)+1., PATH(/1,INO(/1/)/).=DEF1.,
                          ≠COMMENT≠ FIND ITS LEFT SET TO ADD TO THE LEFT SET OF
                                    THE SYMBOL THAT ORIGINALLY CALLED LDEF;
                          LDEF(DEF1).,
                          ≠END≠;

                      ≠COMMENT≠ CHECK IF THE ORIGINAL SYMBOL HAS AN ALTERNATE RHS;
                  L4: ≠IF≠ SYNG[DEF,3]¬=0 ≠THEN≠
 1170**
                          ≠BEGIN≠
                          ≠COMMENT≠ GO THROUGH A SIMILAR PROCEDURE TO THE ABOVE
                                    LOOP TO FIND THE LEFT SET OF THE FIRST
                                    SYMBOL OF THE ALTERNATE RHS;
                          DEF1.=SYNG(/DEF,3/).,
                          PATHCHK(DEF1,L5,1).,
                          INO(/1/).=INO(/1/)+1., PATH(/1,INO(/1/)/).=DEF1.,
                          LDEF(DEF1).,
                          ≠END≠;
 1180**

                  L5: ≠END≠ OF PROCEDURE LDEF;

                  ≠COMMENT≠*****************************************;

                  ≠PROCEDURE≠ RDEF(DEF)., ≠INTEGER≠ DEF.,

                  ≠COMMENT≠ THIS PROCEDURE FINDS THE RIGHT SET OF A SYMBOL
                            WHOSE LHS STARTS AT ROW -DEF-;
 1190**
                      ≠BEGIN≠

                      ≠INTEGER≠ I,DEF1;

                      ≠COMMENT≠ STORE -DEF- IN -DEF1-;
                      DEF1:=DEF;
                      ≠COMMENT≠ CHECK FOR A SUCCESSOR SYMBOL,IF NON THEN WE
                                ARE AT THE RIGHT HAND END OF THE EXPRESSION;
                  LL: ≠IF≠ SYNG[DEF1,4]=0 ≠THEN≠
 1200**
                          ≠BEGIN≠
                          ≠COMMENT≠ STORE VALUE OF SUCCESSOR SYMBOL;
                          LC.=SYNG(/DEF1,1/).,
                          ≠COMMENT≠ IF R IS EMPTY THEN INSERT SYMBOL;
                          ≠COMMENT≠ CHECK IF THIS SYMBOL ALREADY IN R;
                          ≠FOR≠ I.=1 ≠STEP≠ 1 ≠UNTIL≠NR≠DO≠
                          ≠IF≠ GET9(LR,PC,I+SIZE)=LC ≠THEN≠ ≠GOTO≠ L2.,
                      ≠COMMENT≠ INSERT SYMBOL INTO R;
                      L1: NR:=NR+1;
 1210**               PUT9(LR,PC,SIZE+NR,LC).,

                          ≠END≠ ≠ELSE≠

                          ≠BEGIN≠
                          ≠COMMENT≠ UPDATE SUCCESSOR POINTER AND REPEAT CHECK;
                          DEF1.=SYNG(/DEF1,4/)., ≠GOTO≠ LL
                          ≠END≠;
```

```
1220**          ≠COMMENT≠ SEE IF THE LAST SYMBOL SCANNED IS A NON-TERM,
                          IF SO THEN FIND ITS RIGHT SET AND ADD TO THE SET
                          OF THE ORIGINAL SYMBOL;
            L2: ≠IF≠ SYNG[DEF1,2]¬=0 ≠THEN≠
                    ≠BEGIN≠
                    DEF1.=SYNG(/DEF1,2/).,
                    PATHCHK(DEF1,L3,2).,
                    INO(/2/).=INO(/2/)+1., PATH(/2,INO(/2/)/).=DEF1.,
                    RDEF(DEF1).,
1230**
                    ≠END≠;

                ≠COMMENT≠ CHECK IF THE ORIGINAL CALLING SYMBOL HAS AN
                          ALTERNATE RHS,IF SO THEN FIND ALL THE RIGHT SETS:
            L3: ≠IF≠ SYNG[DEF,3]¬=0 ≠THEN≠

                    ≠BEGIN≠
                    DEF1.=SYNG(/DEF,3/).,
1240**              PATHCHK(DEF1,L4,2).,
                    INO(/2/).=INO(/2/)+1., PATH(/2,INO(/2/)/).=DEF1.,
                    RDEF(DEF1).,

                    ≠END≠;

            L4: ≠END≠ OF PROCEDURE RDEF;


            ≠COMMENT≠******************************************;
1250**
            ≠PROCEDURE≠ PATHCHK(K,L,J).,≠VALUE≠ J.,≠INTEGER≠ K,J.,
            ≠LABEL≠ L;

                ≠BEGIN≠

                ≠INTEGER≠ P;
                ≠COMMENT≠ THIS PROCEDURE CHECKS THE PATH TRACED THROUGH
                        THE SYNTAX GRAPH BY *LDEF* AND *RDEF* TO CHECK
                        FOR CYCLING.,
1260**
                ≠FOR≠ P.=1 ≠STEP≠ 1 ≠UNTIL≠ INO(/J/)≠DO≠
                    ≠IF≠ K =    PATH(/J,P/) ≠THEN≠ ≠GOTO≠ L.,

                ≠END≠ OF PROCEDURE PATHCHECK;
        ≠INTEGER≠ SIZE5:
            SIZE5:=(SIZE-1)//5+1;


1270**     ≠COMMENT≠
            BODY OF *LEFTRIGHT*.,

            ≠COMMENT≠ INITIALISE THE SETS;
            ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
                ≠FOR≠ J.=1 ≠STEP≠ 1 ≠UNTIL≠ 2*SIZE5 ≠DO≠
```

```
                        LR[I,J]:=0;
           ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
  1280**    ≠BEGIN≠
              NL.=NR.=0., INO(/1/).=INO(/2/).=1.,
              ≠COMMENT≠ CHECK IF SYMBOL ALREADY SCANNED.,
             PC:=I;
            ≠IF≠ GET9(LR,PC,1) ≠NOTEQUAL≠ 0 ≠THEN≠ ≠GOTO≠ L1.,
              ≠COMMENT≠ GET POINTER TO START OF LHS;
              DEF:=IABS(NONTERMINALS[I,4]);
              ≠COMMENT≠ CHECK FOR UNDEFINED NON TERM;
              ≠IF≠ DEF=0 ≠THEN≠ ≠GOTO≠ L1;
              PATH[1,1]:=PATH[2,1]:=DEF;
  1290**      ≠COMMENT≠ GET LEFT AND RIGHT SETS;
              LDEF(DEF).,RDEF(DEF).,

        L1:   ≠END≠ OF LOOP TO OBTAIN L AND R;

         ≠COMMENT≠ PRINT OUT THE SETS;

         OUTPUT(61,≠(≠↑≠)≠);
  1300**
         OUTPUT(61,≠(≠20B,≠(≠L E F T  P A R T S≠)≠,/,/,
              ≠(≠ SYMBOL≠)≠,20B,≠(≠LEFT PART≠)≠,/,≠(≠  NO.≠)≠,/≠)≠);
         ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
          ≠BEGIN≠
              OUTPUT   (61,≠(≠/≠)≠).,
              OUTPUT   (61,≠(≠BBZZD,BBBB≠)≠,I).,
             J.=0.,
            ≠FOR≠ J.=J+1 ≠WHILE≠  J< SIZE ∧GET9(LR,I,J)¬=0 ≠DO≠
                OUTPUT(61,≠(≠BZZD≠)≠,GET9(LR,I,J)).,
  1310**   ≠END≠ OF LEFT SET PRINT;

              OUTPUT(61,≠(≠/,/,/,/,≠)≠);

         OUTPUT(61,≠(≠20B,≠(≠R I G H T  P A R T S≠)≠,/,/,
              ≠(≠ SYMBOL≠)≠,20B,≠(≠RIGHT PART≠)≠,/,≠(≠  NO.≠)≠,/≠)≠);
         ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
          ≠BEGIN≠
              OUTPUT(61,≠(≠/≠)≠);
              OUTPUT(61,≠(≠BBZZD,BBBB≠)≠,I);
  1320**     J.=0.,
            ≠FOR≠ J.=J+1 ≠WHILE≠  J< SIZE ∧GET9(LR,I,J+SIZE)¬=0 ≠DO≠
                OUTPUT(61,≠(≠BZZD≠)≠,GET9(LR,I,J+SIZE)).,
          ≠END≠ OF RIGHT SET PRINT;

        ≠END≠ OF PROCEDURE LEFTRIGHT;

  1330**  ≠COMMENT≠*****************************************************;
```

```
              ≠PROCEDURE≠ WWPRECEDENCE;
              ≠COMMENT≠ THIS PROCEDURE CALCULATES THE PRECEDENCE MATRIX BY THE
                        RULES OF WIRTH AND WEBER,
                        SEE C.ACM 9 (1),JUNE 1966;
 1340**             ≠BEGIN≠
                  ≠INTEGER≠ SUC;
                    ≠COMMENT≠ -----PROCEDURE *PUTM* -----;
                    ≠PROCEDURE≠ PUTM(II1,II2,I3);
                    ≠VALUE≠ I3;≠INTEGER≠ I3,II1,II2;
                        ≠BEGIN≠
                        ≠INTEGER≠ I1,I2,S1,S2,CLASH,I;
 1350**                 I1:=II1; I2:=II2;
                        ≠COMMENT≠ CALCULATE INDICES INTO -PMATRIX-;
                        ≠IF≠ I1>299 ≠THEN≠ I1.=I1+NTERM-299 ;
                        ≠IF≠ I2>299 ≠THEN≠ I2:=I2+NTERM-299;

                        ≠COMMENT≠ CHECK IF WE NEED TO INSERT SYMBOL;
                       ≠IF≠ GET4(PMATRIX,I1,I2)=1 ≠OR≠ GET4(PMATRIX,I1,I2)=I3
                       ≠THEN≠ PUT4(PMATRIX,I1,I2,I3)
 1360**                 ≠ELSE≠≠BEGIN≠
                              ≠COMMENT≠ PROCESS COLLISION;
                              ≠COMMENT≠ CALCULATE COLLISION TYPE;
                           S1..=GET4(PMATRIX,I1,I2)..  S2..=I3..
                              ≠IF≠ S1=9 ≠THEN≠ ≠GOTO≠ CHKEND;
                              CLASH:=S1+S2;
                              ≠COMMENT≠ CHECK IF WE ALREADY HAVE A DIFFERENT
                                        TYPE OF COLLISION BETWEEN THESE TWO
 1370**                                 SYMBOLS;
                              ≠IF≠ NCONFL=0 ≠THEN≠ ≠GOTO≠ PUTL;
                              ≠FOR≠ I:=1 ≠STEP≠ 1 ≠UNTIL≠ NCONFL ≠DO≠
                                ≠IF≠ CONFLICTABLE[I,2]=II1≠AND≠
                                     CONFLICTABLE[I,3]=II2
                                ≠THEN≠ ≠BEGIN≠
                                       ≠IF≠ CONFLICTABLE[I,1]=CLASH
                                       ≠THEN≠ ≠GOTO≠ CHKEND ≠ELSE≠
                                           ≠BEGIN≠
                                        CONFLICTABLE[I,1]:=9;
 1380**                            PUT4(PMATRIX,I1,I2,9)..
                                         ≠GOTO≠ CHKEND;
                                         ≠END≠;
                                       ≠END≠;

                              ≠COMMENT≠ UPDATE NO. OF CONFLICTS;
                        PUTL: NCONFL:=NCONFL+1;
                              I:=NCONFL;
                              ≠COMMENT≠ INSERT CONFLICT INFORMATION INTO
                                        -CONFLICTABLE- ;
 1390**                       CONFLICTABLE[I,1]:=CLASH;
                              CCNFLICTABLE[I,2]:=II1;
```

```
                              CONFLICTABLE[I,3]:=II2;
                              CCNFLICTABLE[I,4]:=I1;
                              CCNFLICTABLE[I,5]:=I2;
              CHKEND:
                              ≠END≠  PROCESS CONFLICT;
1400**                  ≠END≠ OF PUTM;

                  ≠COMMENT≠ INITIALISE PRECEDENCE MATRIX.,
                   ≠FOR≠ I..=1≠STEP≠ 1≠UNTIL≠ SIZE ≠DO≠
                   ≠FOR≠ J.=1 ≠STEP≠ 1 ≠UNTIL≠ CSIZE ≠DO≠
                     PMATRIX(/I,J/).=ONEFILL4.,
1410**
                  NCONFL:=0;

                  ≠COMMENT≠
              ----- RULE 1 CHECK FOR = ;

                  ≠COMMENT≠ THIS RULE INVOLVES SCANNING DOWN THE SYNTAX
                            GRAPH FOR SYMBOLS LINKED BY A SUCCESSOR POINTER;
1420**            ≠FOR≠ I.=1 ≠STEP≠ 1 ≠UNTIL≠ NGRAPH ≠DO≠

                      ≠BEGIN≠
                      ≠IF≠ SYNG(/I,4/) =  0 ≠THEN≠≠GOTO≠ L10.,
                      I1.=SYNG(/I,1/)., I2.=SYNG(/SYNG(/I,4/),1/).,
                      PUTM(I1,I2,4).,
             L10:     ≠END≠ OF RULE ONE;

                  ≠COMMENT≠
1430**        ----- RULE 2  CHECK FOR < ;
                  ≠FOR≠ I.=1 ≠STEP≠ 1 ≠UNTIL≠ NGRAPH ≠DO≠

                      ≠BEGIN≠
                      ≠IF≠ SYNG(/I,4/) =  0 ≠THEN≠ ≠GOTO≠ L11.,
                      SUC.=SYNG(/I,4/).,
                      ≠IF≠ SYNG(/SUC,2/)  = 0 ≠THEN≠ ≠GOTO≠ L11

                      ≠ELSE≠ ≠BEGIN≠
1440**                        J.=SYNG(/SUC,1/).,
                              I2.=1.,
                L12.. I1.=GET9(LR,J,I2).,
                              I3.=SYNG(/I,1/)., PUTM(I3,I1,2).,
                     I2.=I2+1.,
                     ≠IF≠ I2>SIZE∨GET9(LR,J,I2)=0 ≠THEN≠
                                    ≠GOTO≠ L11 ≠ELSE≠ ≠GOTO≠ L12.,
                              ≠END≠;
             L11:     ≠END≠ OF RULE TWO ;
```

```
 1450**
              ≠COMMENT≠
          ----- RULE 3  CHECK FOR > ;

              ≠FOR≠ I.=1 ≠STEP≠ 1 ≠UNTIL≠ NGRAPH  ≠DO≠

                  ≠BEGIN≠
                  ≠IF≠ SYNG(/I,4/) =  0 ≠THEN≠ ≠GOTO≠ L13.,
 1460**           ≠IF≠ SYNG(/I,2/) =  0 ≠THEN≠ ≠GOTO≠ L13.,
                  SUC.=SYNG(/SYNG(/I,4/),1/).,
                  I2.=1.,
         L14! I1!=GET9(LR,SYNG[I,1],I2+SIZE).,
                  PUTM(I1,SUC,3);
         I2!=I2+1;
         ≠IF≠ I2>SIZE∨GET9(LR,SYNG[I,1],I2+SIZE)=0 ≠THEN≠
                      ≠GOTO≠ L13 ≠ELSE≠ ≠GOTO≠ L14.,
           L13!   ≠END≠ OF RULE THREE ;

 1470**
              ≠COMMENT≠
          ----- RULE 4  FURTHER CHECK FOR > ;

              ≠FOR≠ I.=1 ≠STEP≠ 1 ≠UNTIL≠ NGRAPH ≠DO≠

                  ≠BEGIN≠
                  ≠IF≠ SYNG(/I,4/) =  0 ≠THEN≠ ≠GOTO≠ L15.,
                  ≠IF≠ SYNG(/I,2/) =  0  ∨   SYNG(/SYNG(/I,4/),2/) =  0
                  ≠THEN≠ ≠GOTO≠ L15.,
 1480**
                  SUC.=SYNG(/I,4/).,
                  I1.=1.,
         L16! I3!=GET9(LR,SYNG[I,1],I1+SIZE);
                  I2.=1.,
         L17! I4!=GET9(LR,SYNG[SUC,1],I2).,
             PUTM(I3,I4,3);
             I2!=I2+1;
             ≠IF≠ I2>SIZE ∨GET9(LR,SYNG[SUC,1],I2)=0
                    ≠THEN≠ ≠GOTO≠ L18 ≠ELSE≠ ≠GOTO≠ L17.,
 1490**  L18! I1!=I1+1;
             ≠IF≠ I1>SIZE ∨GET9(LR,SYNG[I,1],I1+SIZE)=0 ≠THEN≠
                            ≠GOTO≠ L15 ≠ELSE≠ ≠GOTO≠ L16.,
           L15!   ≠END≠ OF RULE FOUR ;

              ≠COMMENT≠ THIS BLOCK PRINTS THE PRECEDENCE MATRIX NEATLY;

                  ≠BEGIN≠
 1500**           ≠INTEGER≠ LL,UL;

         ≠COMMENT≠ PRINT PRECEDENCE CONFLICTS;
         ≠IF≠ NCONFL=0 ≠THEN≠
             OUTPUT(61,≠(≠/,/,/,/,10B,≠(≠*NO PRECEDENCE CONFLICTS*≠)≠≠)≠)
         ≠ELSE≠
             ≠BEGIN≠
             OUTPUT(61,≠(≠/,/,/,10B,≠(≠PRECEDENCE CONFLICTS≠)≠,/≠)≠);
```

```
                                  ≠FOR≠ I:= 1 ≠STEP≠ 1 ≠UNTIL≠ NCONFL ≠DO≠
                                          ≠BEGIN≠
 1510**                                   I1:=CONFLICTABLE[I,4]; I2:=CONFLICTABLE[I,5];
                                          PUT4(PMATRIX,I1,I2,CONFLICTABLE(/I,1/)).,
                                          J:=CONFLICTABLE[I,1];
                                          ≠IF≠ J=5 ≠THEN≠ OUTPUT(61,≠(≠/,5B,≠(≠(<,>) BETWEEN  ≠)≠≠)≠)
                                          ≠ELSE≠ ≠IF≠ J=6 ≠THEN≠ OUTPUT(61,≠(≠/,5B,≠(≠(<,=) BETWEEN
                ≠)≠≠)≠)
                                          ≠ELSE≠ ≠IF≠ J=7 ≠THEN≠ OUTPUT(61,≠(≠/,5B,≠(≠(>,=) BETWEEN
                ≠)≠≠)≠)
                                          ≠ELSE≠ OUTPUT(61,≠(≠/,5B,≠(≠(<,=,>) BETWEEN   ≠)≠≠)≠);
                                          PRINTSYMBOL(CONFLICTABLE[I,2],NCHAR);
 1520**                                   OUTPUT(61,≠(≠≠(≠   AND   ≠)≠≠)≠);
                                          PRINTSYMBOL(CONFLICTABLE[I,3],NCHAR);
                                          ≠END≠;
                                           ≠END≠ ;

                                            OUTPUT(61,≠(≠*≠)≠);
                            OUTPUT          (61,≠(≠/50B,≠(≠P R E C E D E N C E  M A T R I X≠)≠/,/≠)≠).,
                                            LL:=1;  UL:=≠IF≠ SIZE>30 ≠THEN≠ 30 ≠ELSE≠ SIZE;
 1530**        PRINTIT:
                                            OUTPUT(61,≠(≠/,8B≠)≠);

                                            ≠FOR≠  I:=LL  ≠STEP≠ 1 ≠UNTIL≠ UL  ≠DO≠
                                                   ≠BEGIN≠
                                                   J:=≠IF≠ I>NTERM ≠THEN≠ I-NTERM+299 ≠ELSE≠ I;
                                                   OUTPUT(61,≠(≠-ZZD≠)≠,J);
                                                   ≠END≠;

                                            ≠FOR≠  I:=1  ≠STEP≠ 1 ≠UNTIL≠ SIZE ≠DO≠
 1540**                                            ≠BEGIN≠
                                                   J:=≠IF≠ I>NTERM ≠THEN≠ I-NTERM+299 ≠ELSE≠ I;
                                                   OUTPUT(61,≠(≠/,/,-ZZD,6B≠)≠,J);
                                                   ≠FOR≠  J:=LL  ≠STEP≠ 1 ≠UNTIL≠ UL  ≠DO≠
                                                          ≠BEGIN≠
                                                            OUTCHARACTER(61,≠(≠ <>=CCCCC≠)≠,GET4(PMATRIX,I,J
                                                                                              )).,
                                                          OUTPUT(61,≠(≠3B≠)≠);
                                                          ≠END≠ ;
                                                   ≠END≠ ;
 1550**
                                           ≠IF≠ SIZE>UL ≠THEN≠

                                                   ≠BEGIN≠
                                                   LL:=UL+1;  UL:=UL+30;
                                                   ≠IF≠ UL>SIZE ≠THEN≠ UL:=SIZE;
                                                   OUTPUT(61,≠(≠/,/,/,/≠)≠);
                                                   ≠GOTO≠ PRINTIT;
                                                   ≠END≠;

 1560**
                                  ≠END≠;
```

```
                ≠END≠ OF PROCEDURE WW PRECEDENCE ;

 1570**       ≠COMMENT≠*****************************************;

              ≠PROCEDURE≠ REMOVE CONFLICTS;
              ≠COMMENT≠ THIS PROCEDURE REMOVES PRECEDENCE CONFLICTS BY
                        LEFT AND RIGHT RESTRICTED EXPANSIONS,
                        SEE GEORGE J.E  SIMPLE - A SIMPLE PRECEDENCE
                        TRANSLATOR WRITING SYSTEM,  STANFORD U STAN-CS-71-226 ;
 1580**           ≠BEGIN≠
                  ≠INTEGER≠ WSUM,SYM;
                  ≠INTEGER≠ ≠ARRAY≠ LRETABLE[1:20],RRETABLE[1:20];

                  ≠PROCEDURE≠ RRE(OLD);
                      ≠VALUE≠ OLD.,
                  ≠COMMENT≠ THIS PROCEDURE PERFORMS A RIGHT RESTRICTED
                            EXPANSION  OF THE SYMBOL WHOSE VALUE IS -OLD- ;
                  ≠INTEGER≠ OLD;
 1590**               ≠BEGIN≠
                      ≠INTEGER≠ I,J,M,JSTART,NEW;
                      ≠COMMENT≠ HASH IN THE NEXT DUMMY SYMBOL,I.E. THE SYMBOL
                                THAT IS REPLACING -OLD- ;
                      HASH(WSUM,NEW,1);
                      ≠COMMENT≠ UPDATE DUMMY SYMBOL NO. FOR NEXT CALL OF RRE
                                OR LRE ;
                      SIZE:=SIZE+1;
                      WSUM:=WORD[1]:=WSUM+1;
                      ≠COMMENT≠ EXTEND SYNTAX GRAPH AND GET POINTER ;
 1600**               NXTBOX:=NGRAPH:=NGRAPH+1;

                      ≠COMMENT≠ INSERT NEW ARTIFICIAL PRODUCTION INTO SYNTAX
                                GRAPH AND UPDATE NO. OF PRODUCTIONS ;
                      NONTERMINALS[NEW,4]:=NXTBOX;
                  SYNG[NXTBOX,2]:=IABS(NONTERMINALS[OLD,4]);
                      SYNG[NXTBOX,1]:=OLD; SYNG[NXTBOX,5]:=NEW;
                      PRODUCTIONS:=PRODUCTIONS+1;

 1610**               ≠COMMENT≠ SCAN THE PRODUCTIONS,REPLACING ALL  OCCURENCES
                                OF -OLD- BY DUMMY SYMBOL EXCEPT WHERE -OLD-
                                IS THE RIGHT-MOST SYMBOL;
                      ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
                          ≠BEGIN≠
                          JSTART:=J:=IABS(NONTERMINALS[I,4]);
                     SUC: ≠IF≠ SYNG[J,1]=OLD∧SYNG[J,4]¬=0 ≠THEN≠ SYNG[J,1]:=NEW;
                          ≠COMMENT≠ TRACE THROUGH SUCCESSORS ;
                          J:=SYNG[J,4];
                          ≠IF≠ J¬=0 ≠THEN≠ ≠GOTO≠ SUC ≠ELSE≠
 1620**
                              ≠BEGIN≠
                              ≠COMMENT≠ TRACE ALTERNATE PRODUCTIONS ;
                              JSTART:=J:=SYNG[JSTART,3];
```

```
                   ≠IF≠ J¬=0 ≠THEN≠ ≠GOTO≠ SUC;
                   ≠END≠;
               ≠END≠;
           ≠END≠ OF RRE;

1630**
       ≠COMMENT≠ THIS PROCEDURE PERFORMS LEFT RESTRICTED
                 EXPANSION OF -OLD- ;
       ≠PROCEDURE≠ LRE(OLD);
           ≠VALUE≠ OLD.,
       ≠INTEGER≠ OLD;
           ≠BEGIN≠
           ≠INTEGER≠ I,J,M,JSTART,NEW;

           ≠COMMENT≠ SIMILAR OPERATION TO RRE FOR CREATING DUMMY
1640**               SYMBOL AND NEW ARTIFICIAL PRODUCTION ;
           HASH(WSUM,NEW,1);
           SIZE:=SIZE+1:
           WSUM:=WORD[1]:=WSUM+1;
           NXTBOX:=NGRAPH:=NGRAPH+1;
           PRODUCTIONS:=PRODUCTIONS+1;
           NONTERMINALS[NEW,4]:=NXTBOX:
           SYNG[NXTBOX,1]:=OLD; SYNG[NXTBOX,5]:=NEW;

           ≠COMMENT≠ SCAN THROUGH PRODUCTIONS,REPLACING -OLD- BY
1650**               DUMMY SYMBOL EXCEPT WHERE -OLD- IS THE LEFT-
                     MOST SYMBOL. COUNTER -M- IS USED TO SEE  WHERE
                     -OLD- LIES IN THE PRODUCTION ;
           ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
               ≠BEGIN≠
               M:=1; JSTART:=J:=IABS(NONTERMINALS[I,4]);
         SUC:  ≠IF≠ M¬=1^SYNG[J,1]=OLD ≠THEN≠ SYNG[J,1]:=NEW;
               ≠COMMENT≠ TRACE SUCCESSORS ;
               M:=M+1; J:=SYNG[J,4];
               ≠IF≠ J¬=0 ≠THEN≠ ≠GOTO≠ SUC ≠ELSE≠
1660**             ≠BEGIN≠
                   ≠COMMENT≠ TRACE ALTERNATES ;
                   M:=1; JSTART:=J:=SYNG[JSTART,3];
                   ≠IF≠ J¬=0 ≠THEN≠ ≠GOTO≠ SUC;
                   ≠END≠;
               ≠END≠;
           ≠END≠ OF LRE;

1670**     ≠FOR≠ I:=1 ≠STEP≠ 1 ≠UNTIL≠ 20 ≠DO≠
           LRETABLE[I]:=RRETABLE[I]:=0;
           ≠COMMENT≠ INITIALISE DUMMY SYMBOL TO XXXX0 ;
           WORD[2]:=WORD[3]:=0:
           WORD[1]:=(((34*64+34)*64+34)*64+34)*64+1;
       WSUM:=WORD[1];

       ≠COMMENT≠ IF NO CONFLICTS THEN SKIP PROCEDURE ;

     MAIN:
1680** ≠IF≠ NCONFL=0 ≠THEN≠ ≠GOTO≠ ENDRC;
```

```
                ≠COMMENT≠ TRACE THROUGH THE CONFLICTS CALLING EITHER LEFT
                          OR RIGHT RESTRICTED EXPANSIONS ACCORDING TO CONFLICT
                          TYPE ;
                ≠FOR≠ I:=1 ≠STEP≠ 1 ≠UNTIL≠ NCONFL ≠DO≠
                    ≠BEGIN≠
                    J:=CONFLICTABLE[I,1];
                    ≠IF≠ J=6 ≠THEN≠
                      ≠BEGIN≠
 1690**               SYM:=CONFLICTABLE[I,3];
                      ≠FOR≠ I1:=1 ≠STEP≠ 1 ≠UNTIL≠ I-1 ≠DO≠
                      ≠IF≠ LRETABLE[I1]=SYM ≠THEN≠ ≠GOTO≠ ENDRL;
                      LRETABLE[I]:=SYM;
                      LRE(SYM)
                      ≠END≠ ≠ELSE≠
                      ≠BEGIN≠
                      SYM:=CONFLICTABLE[I,2];
                      ≠FOR≠ I1:=1 ≠STEP≠ 1 ≠UNTIL≠ I-1 ≠DO≠
                      ≠IF≠ RRETABLE[I1]=SYM ≠THEN≠ ≠GOTO≠ ENDRL;
 1700**               RRETABLE[I]:=SYM;
                      RRE(SYM)
                      ≠END≠;
            ENDRL:  ≠END≠ OF REMOVE LOOP;

                ≠COMMENT≠ PRINT THE GRAMMAR AFTER MODIFICATION ;
                OUTPUT(61,≠(≠↑≠)≠); OUTPUT(61,≠(≠/,5B,≠(≠MODIFIED GRAMMAR AFTER
        REMOVAL OF CONFLICTS ≠)≠≠)≠);
 1710**         PRINTGRAMMAR;

                ≠COMMENT≠ CALL PRECEDENCE PROCEDURES AGAIN TO SEE IF NEW
                          CONFLICTS HAVE BEEN INTRODUCED ;
            CSIZE.=(SIZE-1)//12+1.,
                LEFTRIGHT;
                WWPRECEDENCE;
                ≠GOTO≠ MAIN;

 1720**     ENDRC:
                ≠END≠ OF REMOVE CONFLICTS;


            ≠COMMENT≠**********************************************;
 1730**     ≠PROCEDURE≠ PFUNCTIONS(M,N);
            ≠COMMENT≠ THIS PROCEDURE CALCULATES THE PRECEDENCE FUNCTIONS
                      BY WIRTHS ALGORITHM 265 C.ACM ;
            ≠VALUE≠ N; ≠INTEGER≠ N; ≠INTEGER≠ ≠ARRAY≠ M;
            ≠BEGIN≠
                ≠INTEGER≠ I,J,K,K1,FMIN,GMIN,LS,EQ,GR,NIL;
                ≠INTEGER≠ ≠ARRAY≠ F[1:N],G[1:N];
                ≠PROCEDURE≠ FIXROW(I,L,X); ≠VALUE≠ I,L,X; ≠INTEGER≠ I,L,X;
                ≠BEGIN≠
                    ≠INTEGER≠ J;
```

```
1740**          F[I]:=G[L]+X;
                ≠IF≠ K=K1 ≠THEN≠
                ≠BEGIN≠
                    ≠IF≠ GET4(M,I,K)=LS∧F[I]≥G[K] ≠THEN≠ ≠GOTO≠ FAIL ≠ELSE≠
                    ≠IF≠ GET4(M,I,K)=EQ∧F[I]¬=G[K] ≠THEN≠ ≠GOTO≠ FAIL
                ≠END≠;
                ≠FOR≠ J:=K1 ≠STEP≠ -1 ≠UNTIL≠ 1 ≠DO≠
                ≠IF≠ GET4(M,I,J)=LS∧F[I]≥G[J] ≠THEN≠ FIXCOL(I,J,1) ≠ELSE≠
                ≠IF≠ GET4(M,I,J)=EQ∧F[I]¬=G[J] ≠THEN≠ FIXCOL(I,J,0)
            ≠END≠  FIXROW;
1750**      ≠PROCEDURE≠ FIXCOL(L,J,X); ≠VALUE≠ L,J,X;≠INTEGER≠ L,J,X;
            ≠BEGIN≠
                ≠INTEGER≠ I;
                G[J]:=F[L]+X;
                ≠IF≠ K¬=K1 ≠THEN≠
                    ≠BEGIN≠
                    ≠IF≠ GET4(M,K,J)=GR∧F[K]≤G[J] ≠THEN≠ ≠GOTO≠ FAIL ≠ELSE≠
                    ≠IF≠ GET4(M,K,J)=EQ∧F[K]¬=G[J] ≠THEN≠ ≠GOTO≠ FAIL
                ≠END≠;
                ≠FOR≠ I:=K ≠STEP≠ -1 ≠UNTIL≠ 1 ≠DO≠
1760**          ≠IF≠ GET4(M,I,J)=GR∧F[I]≤G[J] ≠THEN≠ FIXROW(I,J,1) ≠ELSE≠
                ≠IF≠ GET4(M,I,J)=EQ∧F[I]¬=G[J] ≠THEN≠ FIXROW(I,J,0)
            ≠END≠  FIXCOL;
            LS:=2; GR:=3 ; EQ:=4; NIL:=1 ;
            K1:=0;
            ≠FOR≠ K:=1 ≠STEP≠ 1 ≠UNTIL≠ N ≠DO≠
                ≠BEGIN≠
                FMIN:=1;
                ≠FOR≠ J:=1 ≠STEP≠ 1 ≠UNTIL≠ K1 ≠DO≠
                ≠IF≠ GET4(M,K,J)=GR∧FMIN≤G[J] ≠THEN≠ FMIN:=G[J]+1 ≠ELSE≠
1770**          ≠IF≠ GET4(M,K,J)=EQ∧FMIN<G[J] ≠THEN≠ FMIN :=G[J];
                F[K]:=FMIN;
                ≠FOR≠ J:=K1 ≠STEP≠ -1 ≠UNTIL≠ 1 ≠DO≠
                ≠IF≠ GET4(M,K,J)=LS ∧FMIN≥G[J] ≠THEN≠ FIXCOL(K,J,1) ≠ELSE≠
                ≠IF≠ GET4(M,K,J)=EQ∧FMIN>G[J] ≠THEN≠ FIXCOL(K,J,0);
                K1:=K1+1;
                GMIN:=1;
                ≠FOR≠ I:=1 ≠STEP≠ 1 ≠UNTIL≠ K ≠DO≠
                ≠IF≠ GET4(M,I,K)=LS∧F[I]≥GMIN ≠THEN≠ GMIN:=F[I]+1 ≠ELSE≠
                ≠IF≠ GET4(M,I,K)=EQ∧F[I]>GMIN ≠THEN≠ GMIN:=F[I];
1780**          G[K]:=GMIN;
                ≠FOR≠ I:=K ≠STEP≠ -1 ≠UNTIL≠ 1 ≠DO≠
                ≠IF≠ GET4(M,I,K)=GR∧F[I]≤GMIN ≠THEN≠ FIXROW(I,K,1) ≠ELSE≠
                ≠IF≠ GET4(M,I,K)=EQ∧F[I]<GMIN ≠THEN≠ FIXROW(I,K,0)
                ≠END≠ K;

        ≠COMMENT≠ PRINT THE FUNCTIONS ;
        OUTPUT(61,≠(≠/,/,/,20B,≠(≠PRECEDENCE FUNCTIONS≠)≠≠)≠);
        OUTPUT(61,≠(≠/,10B,≠(≠F≠)≠,5B,≠(≠G≠)≠,10B,≠(≠SYMBOL≠)≠≠)≠);
1790**  OUTPUT(61,≠(≠/,/≠)≠);
        ≠FOR≠ I:=1 ≠STEP≠ 1 ≠UNTIL≠ N ≠DO≠
            ≠BEGIN≠
            OUTPUT(61,≠(≠/,7B,-ZZD,2B,-ZZD,10B≠)≠,F[I],G[I]);
            J:=I; ≠IF≠ J>NTERM ≠THEN≠ J:=299+I-NTERM;
            PRINTSYMBOL(J,NCHAR);
            ≠END≠ OF PRINT FUNCTIONS;
        ≠GOTO≠ ENDP;
```

```
         FAIL: OUTPUT(61,≠(≠/,/,/,15B,≠(≠CAN NOT FIND PRECEDENCE FUNCTIONS≠)≠
          ≠)≠);
1800**   ENDP: ≠END≠ OF PFUNCTIONS;

     ≠COMMENT≠************************************************************;

      ≠PROCEDURE≠ TRANSPOSESYNTAX GRAPH;

1810**
     ≠COMMENT≠ THIS PROCEDUPE FORMS LOOKUP TABLES FOR USE IN A PARSER
               FOR A SIMPLE PRECEDENCE GRAMMAR;

         ≠BEGIN≠
         ≠INTEGER≠ SYMBOLNO,JSCAN,NXTBOX,STARTBOX,I,K;
         ≠INTEGER≠ ≠ARRAY≠ NTRHSPTR[300:NTERMNO],TRHSPTR[-1:NTERM],
                           RHSTABLE[1:NGRAPH+10,1:3];

          ≠COMMENT≠ INITIALISE TABLES;
1820**    ≠FOR≠ I:=-1 ≠STEP≠ 1 ≠UNTIL≠ NTERM ≠DO≠ TRHSPTR[I]:=0;
          ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠ NTRHSPTR[I]:=0;
          ≠FOR≠ I:=1 ≠STEP≠ 1 ≠UNTIL≠ NGRAPH+10 ≠DO≠
           ≠FOR≠ K:=1 ≠STEP≠ 1 ≠UNTIL≠ 3 ≠DO≠ RHSTABLE[I,K]:=0;

          ≠COMMENT≠ SET FIPST SYMBOL NO. TO BE SCANNED;
          SYMBOLNO:=-1;  NXTBOX:=1;

       LOOP1:
1830**    JSCAN:=1;

       SYMBOLCHECK:

          ≠IF≠ SYNG[JSCAN,1]=SYMBOLNO ≠THEN≠
              ≠BEGIN≠
              I:=JSCAN;
              ≠IF≠ SYNG[JSCAN,5]=0 ≠THEN≠ ≠GOTO≠ UPDATEJ;
              ≠IF≠ SYMBOLNO≥300 ≠THEN≠
                 ≠BEGIN≠
1840**           ≠IF≠ NTRHSPTR[SYMBOLNO]¬=0 ≠THEN≠
                   RHSTABLE[STARTBOX,2]:=NXTBOX
                 ≠ELSE≠ NTRHSPTR[SYMBOLNO]:=NXTBOX;
                 ≠END≠ ≠ELSE≠

                 ≠BEGIN≠
                 ≠IF≠ TRHSPTR[SYMBOLNO]¬=0 ≠THEN≠
                   RHSTABLE[STARTBOX,2]:=NXTBOX
                 ≠ELSE≠ TRHSPTR[SYMBOLNO]:=NXTBOX;
                 ≠END≠;
1850**
             STARTBOX:=NXTBOX;

       SUCCHECK:

             ≠IF≠ SYNG[I,4]=0 ≠THEN≠
```

```
                    ≠BEGIN≠
                    RHSTABLE[NXTBOX,3]:=-1;
                    RHSTABLE[NXTBOX,1]:=SYNG[JSCAN,5];
                    NXTBOX:=NXTBOX+1;
1860**              ≠GOTO≠ UPDATEJ;
                    ≠END≠ ≠ELSE≠

                    ≠BEGIN≠
                    I:=SYNG[I,4];
                    RHSTABLE[NXTBOX,1]:=SYNG[I,1];
                    ≠IF≠ NXTBOX>STARTBOX ≠THEN≠
                      RHSTABLE[NXTBOX-1,3]:=NXTBOX;
                    NXTBOX:=NXTBOX+1;
                    ≠GOTO≠ SUCCHECK;
1870**              ≠END≠;

                ≠END≠ OF SYMBOL CHECK ;

        UPDATEJ:

                JSCAN:=JSCAN+1;
                ≠IF≠ JSCAN≤NGRAPH ≠THEN≠ ≠GOTO≠ SYMBOLCHECK;
                SYMBOLNO:=SYMBOLNO+1;
1880**          ≠IF≠ SYMBOLNO>NTERMNO ≠THEN≠ ≠GOTO≠ ENDLABEL;
                ≠IF≠ SYMBOLNO>NTERM∧SYMBOLNO<300 ≠THEN≠ SYMBOLNO:=300;
                ≠GOTO≠ LOOP1;


           ≠COMMENT≠ PRINT OUT TABLES;

        ENDLABEL:

            OUTPUT(61,≠(≠↑,/,/,20B,≠(≠ TRANSPOSE SYNTAX GRAPH FOR ANALYSER≠)≠,
1890**      /≠)≠);
            OUTPUT(61,≠(≠/,5B,≠(≠NONTERMINALS INDEX-POINTER TABLE≠)≠,/,
            ≠(≠SYMBOL NUMBER    POINTER TO RHSTABLE≠)≠,/≠)≠);
            ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
                OUTPUT(61,≠(≠5B,ZZD,16B,ZZD,/≠)≠,I,NTRHSPTR[I]);


            OUTPUT(61,≠(≠/,5B,≠(≠TERMINALS INDEX-POINTER TABLE≠)≠,/,
            ≠(≠SYMBOL NUMBER    POINTER TO RHSTABLE≠)≠,/≠)≠);
            ≠FOR≠ I:=-1 ≠STEP≠ 1 ≠UNTIL≠ NTERM ≠DO≠
1900**          OUTPUT(61,≠(≠5B,-ZZD,16B,ZZD,/≠)≠,I,TRHSPTR[I]);

          OUTPUT(61,≠(≠/,/,/,5B,≠(≠TRANSPOSED SYNTAX GRAPH FOR LHS LOOKUP≠)≠,/
            ≠)≠);
          OUTPUT(61,≠(≠/,9B,≠(≠     SYMBOL NO.≠)≠,4B,≠(≠ALT.≠)≠,4B,≠(≠SUC.≠)≠,/
                  ≠)≠);
            ≠FOR≠ I:=1 ≠STEP≠ 1 ≠UNTIL≠ NXTBOX-1 ≠DO≠
                ≠BEGIN≠
                OUTPUT(61,≠(≠/,ZZD,10B,-ZZD,9B,ZZZD,4B,-ZZD,7B≠)≠,I,
                           RHSTABLE[I,1],RHSTABLE[I,2],RHSTABLE[I,3]);
1910**          PRINTSYMBOL(RHSTABLE[I,1],JSCAN);
                ≠END≠ OF PRINT TRANSPOSED GRAPH;
```

```
                ≠END≠ OF TRANSPOSE SYNTAX GRAPH;

                ≠COMMENT≠*********************************************;
 1920**
               ≠PROCEDURE≠ RECURSIVECHECK;

                ≠COMMENT≠ THIS PROCEDURE FINDS THE LEFT,RIGHT AND IMBEDDED
                         RECURSIVE SYMBOLS OF THE GRAMMAR ;

                     ≠BEGIN≠
                     ≠INTEGER≠ ≠ARRAY≠ PATH[1:2,1:NGRAPH],INO[1:2];
                     ≠INTEGER≠ I,IPTR,PC;NCHAR;
 1930**

                     ≠BOOLEAN≠ ≠PROCEDURE≠ LEFTRECURSIVE(I,J);
                     ≠VALUE≠ I,J; ≠INTEGER≠ I,J;
                     ≠COMMENT≠ THIS PROCEDURE CHECKS WHETHER THE SYMBOL -J- IS
                               IS IN THE LEFT SET OF -I- ;

                         ≠BEGIN≠
                         ≠INTEGER≠ K;
                         ≠COMMENT≠ INITIALISE ;
 1940**                  LEFTRECURSIVE:=≠FALSE≠;  K:=0;

                         ≠COMMENT≠ SCAN THROUGH THE LEFT SET OF -I- ;
                         ≠FOR≠ K:=K+1 ≠WHILE≠ K<SIZE+1   ^ GET9(LR,I,K)¬=0 ≠DO≠
                          ≠IF≠ GET9(LR,I,K)=J ≠THEN≠
                               ≠BEGIN≠
                               LEFTRECURSIVE:=≠TRUE≠;  ≠GOTO≠ ENDL;
                               ≠END≠ OF CHECK ;

                  ENDL:  ≠END≠ OF LEFTRECURSIVE;
 1950**


                     ≠BOOLEAN≠ ≠PROCEDURE≠ RIGHTRECURSIVE(I,J);
                     ≠VALUE≠ I,J; ≠INTEGER≠ I,J;
                     ≠COMMENT≠ THIS PROCEDURE CHECKS IF SYMBOL -J- IS IN THE
                               RIGHT SET OF -I- ;

                         ≠BEGIN≠
 1960**                  ≠INTEGER≠ K;
                         ≠COMMENT≠ INITIALISE ;
                         RIGHTRECURSIVE:=≠FALSE≠;   K:=0;

                         ≠COMMENT≠ SCAN THROUGH THE RIGHT SET OF -I- ;
                         ≠FOR≠ K:=K+1 ≠WHILE≠ K<SIZE+1   ^ GET9(LR,I,K+SIZE)¬=0 ≠DO≠
                          ≠IF≠ GET9(LR,I,K+SIZE)=J ≠THEN≠
                               ≠BEGIN≠
                               RIGHTRECURSIVE:=≠TRUE≠;  ≠GOTO≠ ENDL;
                               ≠END≠ OF CHECK ;
 1970**
                  ENDL: ≠END≠ OF RIGHTRECURSIVE;
```

```
                ≠BOOLEAN≠ ≠PROCEDURE≠ IMBED(IPTR);
                ≠VALUE≠ IPTR; ≠INTEGER≠ IPTR;

                ≠COMMENT≠ THIS PROCEDURE FINDS  IF NON-TERMINAL X IS IMBEDDED
                          RECURSIVE,WHERE THE LHS OF X STARTS AT ROW -IPTR- IN
1980**                    THE SYNTAX GRAPH ;

                    ≠BEGIN≠
                    ≠INTEGER≠  J,JK,K,SUC,DEF; ≠BOOLEAN≠ IM;
                    ≠COMMENT≠ INITIALISE ;
                    IMBED:=IM:=≠FALSE≠;

                    ≠COMMENT≠ CHECK PATH TRACED THRO GRAPH FOR CYCLING;
                    INO[1]:=INO[1]+1;  PATH[1,INO[1]]:=IPTR;
                    ≠FOR≠ K:=1 ≠STEP≠ 1 ≠UNTIL≠ INO[1]-1 ≠DO≠
1990**             ≠IF≠ PATH[1,K]=IPTR ≠THEN≠ ≠GOTO≠ ENDL;

                    ≠COMMENT≠ SET POINTERS ;
                    K:=1;  J:=IPTR;  JK:=J;

               L1.. SUC:=SYNG[J,4];
                    ≠IF≠ K>1 ≠AND≠ SYNG[J,1]=PC ≠THEN≠

                         ≠BEGIN≠
                         IM   :=≠TRUE≠;  ≠GOTO≠ ENDL;
2000**                   ≠END≠ ;

                    DEF:=SYNG[J,2];

                    ≠IF≠ DEF>0 ≠AND≠ SUC>0 ≠THEN≠ IM   :=IMBED(DEF) ≠OR≠
                                  RIGHTRECURSIVE(SYNG[J,1],PC)

                    ≠ELSE≠ ≠IF≠ DEF>0 ≠AND≠ SUC=0 ≠THEN≠ IM   :=IMBED(DEF) ≠OR≠
                                  LEFTRECURSIVE(SYNG[J,1],PC)

2010**              ≠ELSE≠ ≠IF≠ DEF=0≠AND≠SUC=0 ≠THEN≠ ≠GOTO≠ UPDATEJ;

                    ≠IF≠ ¬IM    ≠THEN≠

                         ≠BEGIN≠
                         J:=SYNG[J,4]; K:=K+1; ≠IF≠ J>0 ≠THEN≠ ≠GOTO≠ L1;
                         ≠END≠  ≠ELSE≠ ≠GOTO≠ ENDL;

2020**        UPDATEJ: ≠IF≠ ¬IM     ≠THEN≠

                         ≠BEGIN≠
                         J:=SYNG[JK,3]; JK:=J; K:=1;
                         ≠IF≠ J>0 ≠THEN≠ ≠GOTO≠ L1;
                         ≠END≠ ;

              ENDL: IMBED:=IM;
```

```
 2030**
                   ≠END≠ OF IMBED ;

          OUTPUT(61,≠(≠↑,/,7B,≠(≠R E C U R S I O N  C H E C K≠)≠,/≠)≠);
                   OUTPUT(61,≠(≠/,5B,≠(≠LEFT RECURSIVE SYMBOLS≠)≠,/≠)≠);
                   ≠COMMENT≠ FIND LEFT RECURSIVE SYMBOLS ;
                   ≠FOR≠ I:=300 ≠STEP≠ 1≠UNTIL≠ NTERMNO ≠DO≠
 2040**            ≠IF≠ LEFTRECURSIVE(I,I) ≠THEN≠
                        ≠BEGIN≠
                        OUTPUT(61,≠(≠/,3B≠)≠);
                        PRINTSYMBOL(I,NCHAR);
                        ≠END≠ OF FIND LEFT RECURSIVE ;

                   ≠COMMENT≠ FIND RIGHT RECURSIVE SYMBOLS;
          OUTPUT(61,≠(≠/,/,/,5B,≠(≠RIGHT RECURSIVE SYMBOLS≠)≠,/≠)≠);
                   ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
 2050**            ≠IF≠ RIGHTRECURSIVE(I,I) ≠THEN≠
                        ≠BEGIN≠
                        OUTPUT(61,≠(≠/,3B≠)≠);
                        PRINTSYMBOL(I,NCHAR);
                        ≠END≠ OF FIND RIGHT RECURSIVE ;

                   ≠COMMENT≠ FIND THE IMBEDDED RECURSIVE SYMBOLS ;
          OUTPUT(61,≠(≠/,/,/,5B,≠(≠IMBEDDED RECURSIVE SYMBOLS≠)≠,/≠)≠);
                   ≠FOR≠ I:=300 ≠STEP≠ 1 ≠UNTIL≠ NTERMNO ≠DO≠
 2060**                 ≠BEGIN≠
                        PC:=I; IPTR:=IABS(NONTERMINALS[I,4]);
                        INO[1]:=0;
                        ≠IF≠ IMBED(IPTR) ≠THEN≠
                            ≠BEGIN≠
                            OUTPUT(61,≠(≠/,3B≠)≠);
                             PRINTSYMBOL(I,NCHAR);
                             ≠END≠ ;
                        ≠END≠ OF FIND IMBEDDED;
 2070**

               ≠END≠ OF RECURSIVE CHECK ;

           ≠COMMENT≠ ***** CALL THE PRECEDENCE PROCEDURES ***** ;

 2080**
               EMPTYCHECK(EXIT);

           ≠COMMENT≠ GET L(NON-T) AND R(NON-T).,
                LEFTRIGHT.,
          WWPRECEDENCE;
```

```
                  REMOVE CONFLICTS;
                  PFUNCTIONS(PMATPIX,SIZE);
 2090**           TRANSPOSE SYNTAX GRAPH;
                  RECURSIVE CHECK;

            EXIT: ≠END≠ OF SECOND MAJOR PROGRAM BLOCK ;

            ≠END≠
                      ≠EOP≠
                      FINIS
```