EXTENDED SIMPLE PRECEDENCE

Thesis for the Degree of Ph. D.
MICHIGAN STATE UNIVERSITY
LEWIS HARVEY GREENBERG
1970

This is to certify that the

thesis entitled

EXTENDED SIMPLE PRECEDENCE

presented by

Lewis Harvey Greenberg

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in

Electrical Engineering and
System Science

Julian Kateley, Jr.
Major professor

Date December 12, 1969

ABSTRACT

EXTENDED SIMPLE PRECEDENCE

By

Lewis Harvey Greenberg

Grammars are tools for defining structures in languages. Their study has enabled the mechanization of several techniques for processing computer languages. These techniques depend to a large extent on being able to determine the grammatical structure of arbitrary strings of the language.

One method, first introduced by Wirth and Weber, called precedence analysis, is extremely easy to implement if the grammar under consideration has certain basic properties.

This dissertation extends the basic Wirth and Weber precedence concepts to a larger class of grammars. This is achieved by changing the definitions of the precedence relations so that these relations are mutually disjoint and adding a set of context sensitive productions to the original grammar so as to effectively leave the precedence parsing algorithm unchanged. The use of context sensitive productions also provides a technique for attacking the problem of equal right sides. This enables the new method, called extended simple precedence (ESP), to handle a much larger class of grammars than simple precedence does.

EXTENDED SIMPLE PRECEDENCE

By

Lewis Harvey Greenberg

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering and
System Science

1970

ACKNOWLEDGMENTS

I would like to express my gratitude and appreciation to Professor Julian Kateley for his guidance throughout the preparation of this dissertation. His generous assistance, encouragement, and inspiration, played an important part in the completion of this work.

I would also like to thank the other members of my committee, Carl Page, Richard Dubes, William Kilmer and J. Sutherland Frame for their constructive criticism and guidance.

I would also like to extend my gratitude to my wife Maxine, for her assistance in preparing the rough draft and her encouragement and inspiration during the preparation of this dissertation.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

Introduction

The modern digital computer, has made it increasingly important to improve man-machine communications. To this end, many compilers have been designed and built to translate man-oriented languages, i.e. FORTRAN, ALGOL, COBOL, to machine-oriented languages. The man-oriented language which, in general, is machine independent allows the programmer to specify his problem in much the same way he would if he were to solve it by hand. The machine-oriented language, on the other hand, is, in general, completely machine dependent and often impossible to readily comprehend.

In the last few years, much progress has been made in the field of formalized languages. Yet the hand coded ad-hoc variety of languages still plays a dominant role in todays computer systems, mainly because of greater speed and error detecting capability. In order to close the gap between formal and non-formal languages, many papers have appeared in the literature which propose formal methods for accomplishing the task of translation. These methods deal with the structure or grammar of the language to be translated and generate output by assigning meaning to syntactical structures. In order for these methods to be efficient, the algorithm which is to accomplish the syntactic analysis should be made as simple as possible. The simplicity of the algorithm in most instances places unwarranted restrictions on the type and the complexity of the grammar it may effectively handle. In some cases the grammar

can be changed into a form which will make it acceptable to the algorithm, but these changes are usually ad-hoc and in some instances force unacceptable changes in the original structure of the grammar. This dissertation will deal with an extension of a method first proposed by Wirth and Weber [Wi 66] which provides an efficient algorithm for syntactic analysis together with good error detecting capability.

1.1 Basic Notation

The basic entities to be used in the classical model of a grammar are given as follows:

(1) A finite set V. V is called the vocabulary and the elements of V are called symbols.

(2) The operation of concatenation as denoted by juxtaposition will be used to form strings from the elements of V. Strings may also result from the juxtaposition of strings.

(3) The null symbol is denoted by Λ.

Using these primitives several definitions can now be given.

(1) The set of all strings of finite length over a vocabulary V is denoted by V^*

$$V^* = \{x \mid x = \Lambda \vee (\exists y)(\exists Y) y \in V^* \; Y \in V \Rightarrow x = yY\}$$

Lower case Latin letters will be used to denote elements of V^*. Upper case Latin letters will denote elements of V. In either case they may be subscripted.

(2) A production denoted by $l \rightarrow r$ is an ordered pair of strings. The left and right parts of a production are designated l and r respectively. The set of

productions is denoted by P.

(3) The set of symbols on the left hand side of the productions is denoted by V^L

$$V^L = \{A \mid (\exists x)(\exists y)(\exists z)\ yAz \rightarrow yxz \in P\}$$

(4) The set of symbols on the right hand side of the productions in P is denoted by V^R.

$$V^R = \{A \mid (\exists x)(\exists y)(\exists z)\ x \rightarrow yAz \in P\}$$

(5) Σ designates a subset of V called <u>terminal</u> symbols.

$$\Sigma = \{V^R - V^L\}$$

(6) V^N designates the set of <u>non-terminal</u> symbols.

$$V^N = \{V - \Sigma\}$$

When it becomes necessary to distinguish between terminal and non-terminal symbols and strings the following notation will be used:

$\bar{X}$ implies $X \in V^N$

$\bar{y}$ implies $y \in (V^N)^*$

$\hat{X}$ implies $X \in \Sigma$

$\hat{y}$ implies $y \in \Sigma^*$

The length of a string a is denoted by $|a|$ and if $|a| = K$ then the individual symbols of the string a will be denoted by $A_1, A_2, \ldots, A_K$.

When a binary relation, say Δ, is defined for a symbol pair, the following notation will be used to extend the notation to strings:

Let $|a| = K$

$a \Delta b$ implies $A_K \Delta B_1$

$\underline{\Delta}\, a$ implies $A_i \Delta A_{i+1}$ $\quad i = 1,2,\ldots,K-1$

To denote that a relation does not hold between a pair of symbols or strings the $\neg$ sign will be used preceeding the relation i.e.

$a \neg \Delta b$. Where it would be ambiguous to use the = relation (meaning equality) the symbol $\equiv$ will be used.

A <u>context free grammar</u> (CFG) is a 4 tuple, $\langle V,\Sigma,P,G\rangle$

where: $V = \{V_1,V_2,\ldots,V_M,\ldots,V_N\}$ is the vocabulary.

$\Sigma = \{V_{M+1},V_{M+2},\ldots,V_N\}$ is the set of terminal symbols.

P is a restricted set of productions of the form $\bar{X} \rightarrow y$.

G is the goal symbol or the starting type and is an element of the non-terminal set.

A <u>derivation</u> is denoted by $x \Rightarrow y$ and is defined as follows:

$x \Rightarrow y$ iff $x = w\bar{A}z$ and $y = wsz$ then either

$A \rightarrow s \in P$ or $A \rightarrow t \in P$ and $t \Rightarrow s$

The <u>language</u> $L(Z)$ defined by the grammar Z is the set

$$L(Z) = \{y \mid G \Rightarrow \hat{y}\}$$

The set of <u>head symbols</u>, $HS(X)$, is the set of all symbols with which some derivation from X may begin.

$$HS(X) = \{Y \mid X \Rightarrow Yz\}$$

$$HS(\hat{X}) = \phi$$

Similarly, the set of <u>tail symbols</u>, $TS(X)$, is the set of all symbols with which some derivation from X may end.

$$TS(X) = \{Y \mid X \Rightarrow zY\}$$

$$TS(\hat{X}) = \phi$$

A <u>sentential form</u> (SF) is any string that can be derived from the goal symbol. The set of sentential forms of which the language of a grammar is a proper subset is defined as

$$\{z \mid G \Rightarrow z\}$$

The derivation of an element of a language will, in general, entail the generation of a goodly number of SF's as intermediate

steps. Since at each step there will in general exist several non-terminals, any one of which may be replaced using a production, the set of SF's used in the derivation is not unique. For this reason a canonical form for derivations will next be defined.

A derivation is said to be rightmost (leftmost) if at each step of the derivation the rightmost (leftmost) non-terminal symbol is the symbol upon which the next SF is derived. If it is the case that there exists more than one rightmost (leftmost) derivation for the same SF then the grammar is said to be ambiguous`.

Parsing, the reverse process of generation, is an algorithmic method for determining whether a string is an element of a language and if so, what productions were used in its generation. There are two main approaches to parsing. The first of these methods, called top-down, tries to match the string in question by generating a new string which is identical to the original. Cheatham and Sattley [Ch 64] give a clear treatment of this method together with some of its pit-falls and draw-backs. The second method, called bottom-up, tries to reduce the original string by replacing substrings in the SF which correspond to the right hand side of some production with the left hand side. The algorithm is successful if the original string can be reduced to the goal symbol. These methods differ mainly in the amount of foresight and hindsight used in determining what constitutes a proper substitution.

A parsing algorithm is said to be deterministic if at every point in the analysis the next step is uniquely determinable. This is a very important consideration when evaluating the efficiency of an algorithm since if there exists a choice at some point in

the analysis, the fact that the wrong choice was made may not become apparent until after several additional steps. Then if the analysis is to succeed the algorithm must make some provision for backing up to the point of the incorrect choice so that an alternative choice can be made. Clearly this type of analysis complicates the algorithm and puts great restrictions on generation of output.

A parsing algorithm is said to yield a left to right canonical parse (LRCP) if the substring to be reduced is always the leftmost reducible substring in the sentential form. For bottom-up parsing, a parse is an LRCP if the order in which the productions are applied to reduce the string is just the reverse of the rightmost derivation (RMD) for that string.

1.2 Dissertation Objectives

Given a grammar $\langle V,\Sigma,P,G\rangle$, it is a trivial task, starting with the goal G and repeatedly using the various productions, to derive a string of terminal symbols such that this string is in the language specified by the grammar. However, parsing, the reverse process of generation, is in general much more complex. Since it is the objective of a compiler to determine the structure of an arbitrary string and to generate code dependent on this structural information, a solution to the parsing problem has practical significance. Given a grammar, several algorithms exist for the parsing of strings and this dissertation will deal with an extension to one of the simplest of these methods.

In Chapter II the concepts of precedence are introduced. This method attempts to reconstruct the SF's of the grammar by

the bottom-up method. This is accomplished by insertion of brackets at appropriate points in the string so that the substring that is to be replaced is clearly delineated. This substring always corresponds to the right hand side of some production in the grammar. The process for determining where the brackets will be placed in the string is based on a set of relations between symbol pairs which may legally appear adjacent to one another in a SF. If however, the set of relations are not disjoint then the brackets are not unique and the substrings are not properly delineated. Even when the substrings are properly delineated there may still be a problem in determining which production is to be applied since there may exist two productions which have the same right sides.

In Chapter III a new formulation of precedence is defined in such a way that the set of relations is always disjoint. However the set of substrings delineated by the brackets no longer corresponds directly to the right hand sides of productions and therefore a new set of replacement rules must be defined. In general these rules take the form of context sensitive productions which we will loosely define to be any production which has more than one symbol on the left hand side. A method for determining the set of context sensitive productions is also given in this chapter.

Since the set of context sensitive productions is not in general finite, the concept of bounds is introduced in Chapter IV. If the set of bounds exists then there exists a finite subset of productions defined by the set of bounds which is sufficient to

parse any string in the language.

One method for dealing with the problem of equal right sides is to examine the context in which the reduction is to take place. Since the mechanism already exists for examining the context under the new formulation, this method is also discussed in Chapter IV.

Figure 1.1, taken from Feldman [Fl 68], shows the relationship among various types of grammars. Thus LR(k) is seen to be the most general of the various grammars shown. Under certain conditions, as presented in Chapter V, ESP grammars are at least as general as LR(k) grammars.

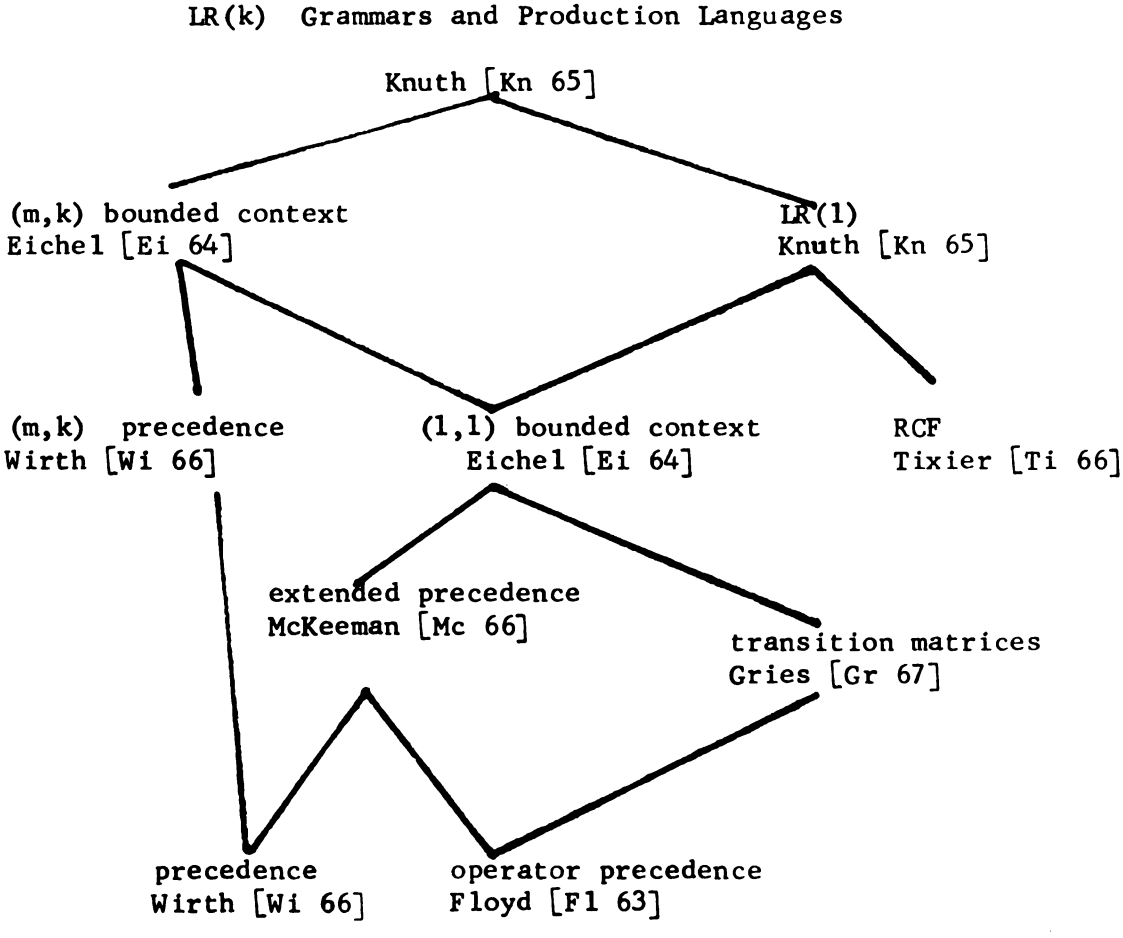


Figure 1.1. Inclusion Tree of Parsing Algorithms

CHAPTER II

Precedence Grammars

There are two distinct types of precedence analysis in use today. The first, called operator precedence, was first introduced by Floyd [Fl 63] and is concerned with the interrelation between terminal symbols of the grammar. This method of syntactic analysis is restricted to a specific form of context free languages, known as operator grammars. A grammar is said to be an operator grammar if none of its productions contains two adjacent non-terminal symbols. This type of analysis was extended to a larger class of grammars by Colmerauer [Co 68] and Gries [Gr 68].

The second type of precedence analysis, sometimes referred to as total precedence, is concerned with the interrelation of all the symbols in the grammar. This technique was first described by Wirth and Weber [Wi 66] and has been extended by McKeeman [Mc 66] and Cheatham [Ch 68]. Although there do exist grammars which are both operator and total precedence, the class of grammars which exhibit the simplest form of total precedence (referred to as simple precedence) does not contain the entire class of operator precedence grammars.

2.1 Simple precedence

The precedence relations for simple precedence (SP) are defined between pairs of symbols as follows:

$X \doteq Y$ if $L \rightarrow aXYb \in P$

$X \lessdot Y$ if $L \rightarrow aXZb \in P$ and $Y \in HS(Z)$

$X \gtrdot Y$ if $L \rightarrow aZYb \in P$ and $X \in TS(Z)$

or $L \rightarrow aZWb \in P$ and $X \in TS(Z)$ and $Y \in HS(W)$

If there exists at most one relation between each pair of symbols in V and no two productions have the same right sides or are of the form $X \rightarrow \Lambda$ then the grammar is said to be simple precedence.

The relationship between Boolean matrices and relations can be utilized to easily formulate the precedence relations. Three Boolean matrices are constructed from the production set as follows:

$E(I,J) = 1$ if $L \rightarrow aV_IV_Jb \in P$

$= 0$ otherwise

$h(I,J) = 1$ if $V_I \rightarrow V_Jb \in P$

$= 0$ otherwise

$t(I,J) = 1$ if $V_I \rightarrow aV_J \in P$

$= 0$ otherwise

From the definition of the set of head symbols HS it is easily seen that the transitive closure of the relation h is the set of ordered pairs HS. Similarly the set of tail symbols TS is given by the transitive closure of the relation t. Letting H and T denote Boolean matrices which represent the transitive closure of the relations h and t respectively, the following can easily be shown:

$H(I,J) = 1$ if $V_J \in HS(V_I)$

$= 0$ otherwise

$T(I,J) = 1$ if $V_J \in TS(V_I)$

$= 0$ otherwise

The above transitive closure operations on the Boolean matrices can be obtained by using Warshall's algorithm [Wa 62]. The complexity of this algorithm is on a par with matrix multiplication but its efficiency can be increased by arranging the rows and columns so that all the non-terminal symbols precede the terminal symbols, thus taking advantage of the zero entries that occur in the rows for terminal symbols.

Using the Boolean equivalent of matrix multiplication and addition, the following four matrices are defined:

$$L = EH \qquad G_2 = T'L$$

$$G_1 = T'E \qquad G = G_1 + G_2$$

The precedence relations can then be defined as follows:

$V_I \doteq V_J$ if $E(I,J) = 1$

$V_I \lessdot V_J$ if $L(I,J) = 1$

$V_I \gtrdot V_J$ if $G(I,J) = 1$

The following matrix notation will be used throughout this dissertation:

(1) AB will denote matrix multiplicator using and and or.

(2) T' will denote the transpose of the matrix T

(3) $A \cdot B$ will denote the matrix the elements of which equal the logical product of the corresponding elements of A and B.

$C = A \cdot B$ implies $C(I,J) = A(I,J)$ and $B(I,J)$

(4) E^c will denote the logical complement of the matrix E.

$E^c(I,J) = 1$ if $E(I,J) = 0$

$= 0$ otherwise

(5) A + B will denote matrix addition using the logical operator or.

When it is important to distinguish between the two types of $\triangleright$ the following notation will be used:

$$V_I >_1 V_J \quad \text{if} \quad G_1(I,J) = 1$$

$$V_I >_2 V_J \quad \text{if} \quad G_2(I,J) = 1$$

Since the above algorithm is well suited for computer implementation, it might be interesting to examine some of the data for a few of the languages that have been analyzed over the past two years using this method. Figure 2.1 summarizes the results of these analyses.

Similar Boolean matrix formulations for precedence grammars have been presented by Colmerauer [Co 68] and Martin [Ma 68].

Language	Number of productions	Total time for analysis (SEC)	Percent of non-blank to total entries in the precedence matrix
MAD/I [Mi 68]	106	5.767	24.10
MKL [Mc 66]	95	6.484	52.08
ALGOL [Co 68]	353	31.494	8.11
PI/I [A1 68]	689	96.124	9.90

Figure 2.1

2.2 The precedence parsing algorithm

The precedence parsing algorithm as presented by Wirth and Weber [Wi 66] is extremely simple to implement. Two stacks are used to hold the SF being analyzed. In the initial configuration (Fig. 2.2a), the string to be analyzed minus its first symbol resides on the right stack and the relation $\lessdot$ followed by the first symbol of the string resides on the left stack. Designating

the top elements of the left and right stacks L_1 and R_1 respectively, the algorithm proceeds as follows. If $L_1 \lessdot R_1$ (Fig. 2.2b) then the symbol $\lessdot$ followed by the top symbol in the right stack is pushed down on the left stack. If $L_1 \doteq R_1$ (Fig. 2.2c) then the top symbol on the right stack is pushed down on the left stack. If $L_1 \gtrdot R_1$ (Fig. 2.2d) then the symbols on the left stack between the most recent $\lessdot$ symbol and the top of the stack constitute the right hand side of some production. These symbols together with the $\lessdot$ symbol are popped off the left stack and the corresponding left hand side of the production is pushed onto the right stack. The process continues until only the goal symbol remains on the left stack and the right stack is empty. Whenever the right stack is empty the algorithm assumes that $L_1 \gtrdot R_1$. Whenever the left stack is empty the $\lessdot$ symbol followed by R_1 is pushed onto the left stack.

$\lessdot$ A B C D E F ...

Left stack Right stack

(a)

$\lessdot$ A $\lessdot$ B C D E F ...

L_1 R_1

If $L_1 \lessdot R_1$ then the result is:

$\lessdot$ A $\lessdot$ B C $\lessdot$ D E F ...

(b)

If $L_1 \doteq R_1$ then the result is:

$\lessdot$ A $\lessdot$ B C D E F ...

(c)

If $L_1 \blacktriangleright R_1$ and there exists a production of the form $S \rightarrow BC \in P$ then the result is:

$\blacktriangleleft$ A S D E F ...

(d)

Figure 2.2 Initial configuration and next possible steps of the precedence analyzer.

Error detection comes at two points in the algorithm. The first case arises when there is no relation between L_1 and R_1. This implies that the string is not an element of the language since if a symbol can appear adjacent to some other symbol in a SF then they are related by at least one of the precedence relations. The second case arises when a substring bounded by the relations $\blacktriangleleft$ and $\blacktriangleright$ does not correspond to the right hand side of any production. This condition also indicates the occurrence of an invalid string.

Before going on it might be advantageous to give an example of a simple precedence grammar together with the parse of a valid string.

Example 2.1 The formation of the precedence relations and the parsing technique.

The grammar

A → BCD
B → BA
B → D
C → Y

The h matrix

	A	B	C	D	Y
A	0	1	0	0	0
B	0	1	0	1	0
C	0	0	0	0	1
D	0	0	0	0	0
Y	0	0	0	0	0

The t matrix

	A	B	C	D	Y
A	0	0	0	1	0
B	1	0	0	1	0
C	0	0	0	0	1
D	0	0	0	0	0
Y	0	0	0	0	0

The H matrix

	A	B	C	D	Y
A	0	1	0	1	0
B	0	1	0	1	0
C	0	0	0	0	1
D	0	0	0	0	0
Y	0	0	0	0	0

The T matrix

	A	B	C	D	Y
A	0	0	0	1	0
B	1	0	0	1	0
C	0	0	0	0	1
D	0	0	0	0	0
Y	0	0	0	0	0

The E matrix

	A	B	C	D	Y
A	0	0	0	0	0
B	1	0	1	0	0
C	0	0	0	1	0
D	0	0	0	0	0
Y	0	0	0	0	0

The L matrix

	A	B	C	D	Y
A	0	0	0	0	0
B	0	1	0	1	1
C	0	0	0	0	0
D	0	0	0	0	0
Y	0	0	0	0	0

The G matrix

	A	B	C	D	Y
A	1	1	1	1	1
B	0	0	0	0	0
C	0	0	0	0	0
D	1	1	1	1	1
Y	0	0	0	1	0

The precedence matrix

	A	B	C	D	Y
A	$\doteq$	$\doteq$	$\doteq$	$\doteq$	$\doteq$
B	$\doteq$	$\lessdot$	$\doteq$	$\lessdot$	$\lessdot$
C	0	0	0	$\doteq$	0
D	$\doteq$	$\doteq$	$\doteq$	$\doteq$	$\doteq$
Y	0	0	0	$\doteq$	0

The parse of the string DDDYDYDYD

Sentential forms	Production used
$\lessdot$ D $\doteq$ DDYDYDYD	B → D
$\lessdot$ B $\lessdot$ D $\doteq$ DYDYDYD	B → D
$\lessdot$ B $\lessdot$ B $\lessdot$ D $\doteq$ YDYDYD	B → D
$\lessdot$ B $\lessdot$ B $\lessdot$ B $\lessdot$ Y $\doteq$ DYDYD	C → Y
$\lessdot$ B $\lessdot$ B $\lessdot$ B $\doteq$ C $\doteq$ D $\doteq$ YDYD	A → BCD
$\lessdot$ B $\lessdot$ B $\doteq$ A $\doteq$ YDYD	B → BA
$\lessdot$ B $\lessdot$ B $\lessdot$ Y $>$ DYD	C → Y
$\lessdot$ B $\lessdot$ B $\doteq$ C $\doteq$ D $\doteq$ YD	A → BCD
$\lessdot$ B $\doteq$ A $\doteq$ YD	B → BA
$\lessdot$ B $\lessdot$ Y $\doteq$ D	C → Y
$\lessdot$ B $\doteq$ C $\doteq$ D $\doteq$	A → BCD
$\lessdot$ A $\doteq$	

2.3 Conflicts in simple precedence

Conflicts arise in simple precedence when more than one relation exists between a pair of symbols. These conflicts consist of five basic forms; ($\lessdot$,$\doteq$), ($\doteq$,$\gtrdot_1$), ($\doteq$,$\gtrdot_2$), ($\lessdot$,$\gtrdot_1$), and ($\lessdot$,$\gtrdot_2$). Squires [Sq 66] has shown that there exist special cases of the ($\lessdot$,$\doteq$) and ($\doteq$,$\gtrdot_1$) conflicts which may be resolved by the addition of a new non-terminal symbol and a trivial production.

The first special case arises when a symbol W is its own head symbol. If such a symbol appears to the right of some other

symbol V in some production then by definition $V \doteq W$ and $V \lessdot W$. Removing the conflict involves the addition of a new symbol, say W', which is substituted for each occurrence of W appearing to the right of some symbol in a production. Then the trivial production, $W' \rightarrow W$ is added to the production set. Now since there exists no symbol V such that $V \doteq W$ the conflict has been removed.

The second special case involves a symbol which is its own tail symbol appearing to the left of some symbol in a production. A similar method will remove this $(\doteq, \gtrdot_1)$ conflict.

The third conflict which we would like to examine involves the $\lessdot$ and $\gtrdot$ relations. Colmerauer [Co 68] points out that this type of conflict arises from the fact that the precedence algorithm always yields a LRCP. If one is willing to forego the requirement of a LRCP then any $\gtrdot_2$ relation may be arbitrarily changed to $\lessdot$ without effecting the parsing algorithm. By examining the general form of the $(\lessdot, \gtrdot_2)$ conflict we observe that the following must be present in the grammar:

$$\left.\begin{array}{ll} (1) & V \rightarrow cXYd \\ (2) & X \Rightarrow eA \\ (3) & Y \Rightarrow Bf \end{array}\right] \Rightarrow A \gtrdot_2 B$$

$$\left.\begin{array}{ll} (4) & W \rightarrow gAUh \\ (5) & U \Rightarrow Bi \end{array}\right] \Rightarrow A \lessdot B$$

In applying the parsing algorithm we can't decide whether to complete the definition of X, i.e., $A \gtrdot B$, or start building the definition of U, i.e., $A \lessdot B$. But in either case, we must eventually start working on the definitions of either U or Y, i.e., $X \lessdot B$, and therefore we might just as well form U or Y before considering

whether A constitutes the end of some definition. Clearly we no longer have a LRCP since this would require the completion of X before Y. However once either U or Y is found the problem of what to do with A is resolved since if Y is found then $A \gtrdot Y$ and if U is found then $A \doteq U$.

Cheathem [Ch 68] has shown that any grammar may be changed so that there do not exist any precedence conflicts between symbol pairs. This is accomplished by requiring that no symbol may appear more than once in the right hand side of all the non-trivial productions of the grammar. To convert any grammar to this form, one need only replace each repeated symbol B with a new symbol B_i and add the trivial production $B_i \rightarrow B$. The following example will demonstrate the procedure.

Example 2.2 The convertion of a non-precedence grammar

The grammar

$A \rightarrow CBC$

$B \rightarrow XBX$

$B \rightarrow X$

The precedence matrix

	A	B	C	X
A	0	0	0	0
B	0	0	$\doteq$	$\doteq$
C	0	$\doteq$	0	$\lessdot$
X	0	$\doteq$	$\gtrdot$	$\lessdot,\gtrdot$

The new Grammar

$A \rightarrow C_1B_1C_2$

$B \rightarrow X_1B_2X_2$

$B \rightarrow X$

$B_1 \rightarrow B$

$B_2 \rightarrow B$

$C_1 \rightarrow C$

$C_2 \rightarrow C$

$X_1 \rightarrow X$

$X_2 \rightarrow X$

The precedence matrix

	A	B	B_1	B_2	C_1	C_2	X_1	X_2	C	X
A	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	$\gtrdot$	$\gtrdot$	0	0	0
B_1	0	0	0	0	0	$\doteq$	0	0	0	0
B_2	0	0	0	0	0	0	$\doteq$	0	0	0
C_1	0	$\lessdot$	$\doteq$	0	0	0	$\lessdot$	0	0	$\lessdot$
C_2	0	0	0	0	0	0	0	0	0	0
X_1	0	$\lessdot$	0	$\doteq$	0	0	$\lessdot$	0	0	$\lessdot$
X_2	0	0	0	0	0	$\gtrdot$	0	$\gtrdot$	$\gtrdot$	$\gtrdot$
C	0	$\gtrdot$	$\gtrdot$	0	0	0	$\gtrdot$	0	0	$\gtrdot$
X	0	$\gtrdot$	$\gtrdot$	$\gtrdot$	0	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$

There doesn't exist a deterministic parsing algorithm which will successfully handle the grammar in example 2.2. This is because the algorithm has no way of determining the middle of the string of X's which is where the first production should be applied. Therefore although we have eliminated all of the precedence conflicts the new grammar is not simple precedence. All that we have done is to exchange the precedence conflicts for a set of productions with equal right sides. Also we note that the number of non-terminal symbols has greatly increased as has the set of productions.

2.4 Extended precedence

Wirth and Weber [Wi 66] also defined an extended version of precedence by extending the definition to strings. This higher order precedence is defined as follows:

Let $|x| = N$

$x \doteq y$ if $A \; bX_N Y_1 c \in P$ and $bX_N \overset{*}{\Rightarrow} b'x$, $Y_1 c \overset{*}{\Rightarrow} yc'$

$x < y$ if $A \; bX_N Vc \in P$ and $bX_N \overset{*}{\Rightarrow} b'x$, $Vc \Rightarrow yc'$

$x > y$ if $A \; bUY_1 c \in P$ and $bU \Rightarrow b'x$, $Y_1 c \overset{*}{\Rightarrow} yc'$

or $A \; bUVc \in P$ and $bU \Rightarrow b'x$, $Vc \Rightarrow yc'$

where $a \overset{*}{\Rightarrow} b$ implies $a \equiv b$ or $a \Rightarrow b$

A grammar is called a precedence grammar of order (M,N) if for the strings x, $|x| = M$, and y, $|y| = N$, there exists at most one relation between x and y. Clearly if a and b are strings such that $|a| < M$ and $|b| < N$ and there exists at most one relation between a and b then for any strings ca and db at most one relation holds between them. This new definition of precedence will tend to increase the size of the precedence matrix

to unwieldy proportion unless some procedure is introduced so that the analyzer will look at strings of length greater than one only when it is necessary. McKeeman [Mc 66] took this into consideration in his extension of the precedence algorithm.

McKeeman decomposed the precedence algorithm into two distinct parts. The first part started scanning the input string from left to right, looking for a $\doteq$ relation as defined in a precedence grammar of order (2,1). Taking advantage of the fact that a reducible substring is always bounded on the right by a terminal symbol in a LRCP, McKeeman was able to simplfy the definition of the $\doteq$ relation to:

$$x \doteq \hat{Y} \quad \text{if} \quad A \rightarrow bWU\hat{Y}c \quad \text{and} \quad WU \Rightarrow w'x$$

$$\text{or} \quad A \rightarrow bWUVc \quad \text{and} \quad WU \Rightarrow w'x \ , \ V \Rightarrow \hat{Y}d$$

The second part involves scanning from the point at which the $\doteq$ relation has been found, back to the left, looking for a $\lessdot$ relation as defined by a precedence grammar of order (1,2). Again the definition of $<$ can be simplified to:

$$X \lessdot y \quad \text{if} \quad A \rightarrow bXVWc \quad \text{and} \quad VW \Rightarrow yw'$$

The two sets of triples are used to find the precedence relations only when there is a conflict in the simple precedence relations. These conflicts are flagged by a special symbol in the simple precedence matrix which causes the analyzer to reference the appropriate set of triples for the correct relation.

Both of these extensions contain Floyd's operator precedence as a subset since both have the ability to look at the operators which in an operator grammar are never separated by more than one symbol. However, neither of these methods can handle the problem

of equal right sides in the production set.

In Chapter III we will examine another method for extending simple precedence which will use an augmented set of productions to overcome precedence conflicts.

CHAPTER III

Extended Simple Precedence

As was shown in Chapter II, Wirth and Weber extended the notion of simple precedence by looking at more than two symbols in order to determine the proper precedence relation. It was pointed out that the size of the precedence matrix grew as a power of the number of symbols used in the determination. In this chapter we will introduce a new method for extending simple precedence without increasing the size of the precedence matrix. Instead, the set of production rules will be augmented with a set of context sensitive productions (CSP) which will be used to remove conflicts in the precedence relations.

3.1 The New Precedence Relations

Conflicts in simple precedence, as mentioned before, take five basic forms: $(\lessdot,\doteq)$, $(\doteq,\gtrdot_1)$, $(\lessdot,\gtrdot_1)$ and $(\lessdot,\gtrdot_2)$. As was shown before, the $(\lessdot,\gtrdot_2)$ conflict is of no consequence if one is willing to forego the restriction of a LRCP. In the new method a LRCP will not be a criterion. Therefore, the new precedence relations are defined as follows:

Definition 3.1

$X = Y$ iff $X \doteq Y$ or $(X \lessdot Y$ and $X \gtrdot_1 Y)$

$X < Y$ iff $X \lessdot Y$ and $X \neg = Y$

$X > Y$ iff $X \gtrdot Y$ and $X \neg = Y$ and $X \neg < Y$

Now there is no possibility of a conflict in the relations between two symbols because the definitions are mutually exclusive.

In matrix notation the definitions are:

$$V_I = V_J \quad \text{iff} \quad \mathcal{E}(I,J) = 1$$

$$V_I < V_J \quad \text{iff} \quad \mathcal{L}(I,J) = 1$$

$$V_I > V_J \quad \text{if} \quad \mathcal{G}(I,J) = 1$$

where:

$$\mathcal{E} = E + L \cdot G_1$$

$$\mathcal{L} = L \cdot \mathcal{E}^C = L \cdot E^C \cdot G_1^C$$

$$\mathcal{G} = G \cdot \mathcal{E}^C \cdot \mathcal{L}^C = G \cdot E^C \cdot L^C$$

Clearly, $\mathcal{E} \cdot \mathcal{L} = \mathcal{E} \cdot \mathcal{G} = \mathcal{L} \cdot \mathcal{G} = 0$ by inspection.

The main idea behind this new definition is the fact that by making all conflicts between any two symbols = we have enlarged the number of symbols which can be examined in determining the proper reduction to be applied. That is, we have changed the problem by delaying the decision of whether to include a symbol in the reducible string by always including it in the string whenever there is a chance that it might be allowed.

Before going on, consider an example. This is a grammar taken from Wirth and Weber's paper which is not simple precedence.

Example 4.1

G: $A \rightarrow B;B$

$B \rightarrow [A]$

$B \rightarrow [L]$

$B \rightarrow L$

H matrix

	A	B	[	]	;	L
A	0	1	1	0	0	1
B	0	0	1	0	0	1

T matrix

	A	B	[	]	;	L
A	0	1	0	1	0	1
B	0	0	0	1	0	1

Precedence matrix

	A	B	[	]	;	L
A	0	0	0	$\doteq$	0	0
B	0	0	0	$\gtrdot$	$\doteq$	0
[	$\doteq$	$\lessdot$	$\lessdot$	0	0	$\leqslant$
]	0	0	0	$\gtrdot$	$\gtrdot$	0
;	0	$\doteq$	$\lessdot$	0	0	$\lessdot$
L	0	0	0	$\geqslant$	$>$	0

There are two conflicts in the grammar which under the new definitions will be changed to the = relation. Examining a typical string, L;[[L;[L]];L] we note that the reducible string [L] comes up whenever it appears since [=L=].[1] [L will come up iff [L; is in the string since ; is the only symbol S such that L > S. By the same reasoning, L] appears iff ;L] was in the string.

The new set of productions is:

A → B;B	B → L
B → [L]	[B → [L
B → [A]	B] → L]

and the parse of the sentential form [L;L];L is as follows:

<[=L>;L];L	[B → [L
<[B=;<L=]>;L	B] → L]
<[B=;=B>],L	A → B;B
<[=A=]>;L	B → [A]
<B=;<L>	B → L
<B=;=B>	A → B;B
<A>	

3.2 The Formal Mechanism

Theorem 3.1

Let aNb be a SF

Let N → n

Let b' ⇒ b

Let anb' be a SF

then, if the grammar is unambiguous, aNb' is a SF.

---

[1] The only relation between] and any symbol is the > relation and the only relation between any symbol and [is the < relation.

Proof:

First, since aNb is a SF, then anb is also and any derivation sequence which is used to derive anb must contain $N \rightarrow n$ by the definition of unambiguity.

$$\begin{array}{ll} \sigma \overset{1}{\Rightarrow} aNb \overset{4}{\rightarrow} anb \\ 2 \Downarrow \\ anb' \underset{3}{\Rightarrow} anb \end{array}$$

The derivation sequence of $b' \Rightarrow b$ must be included in 1 since it was used in 3 to derive the same string. Then, if in 1, we remove the proper subsequence which corresponds to 3, the result must be the derivation of aNb'.

Theorem 3.2

Let nb be a SF

Let $\overset{\leq}{n}$

Let $b \equiv cde$ such that $\overset{\leq}{c} < \overset{=}{d} > e$

Then there exists a d' such that $d' \Rightarrow d$ and a SF nb' such that $b' \equiv cd'e$.

Proof:

Since each element of the string d is equal to the next, there are, in reality, 3 possible relations between the elements of the string. Therefore, there are, at most, $3^{|d|}$ possible ways that d can be factored and at least one of these is correct for this SF form. Let $R_i \in \{<, \doteq, >\}$. There exists one and only one sequence of R's for the given SF such that $R_0 D_1 R_1 D_2 \cdots R_{N-1} D_N R_N$ is correct. We locate the smallest i such that $R_i \equiv >$. Clearly, there always exists an i for which this is true. Then we find the largest j less than i such that $R_j \equiv <$. Then for all k, j less than k less than i, $R_k \equiv \doteq$ which implies that $D_j \cdots D_i$

is an alternative of some non-terminal symbol Q. Then $d' = D_1 \ldots D_{j-1} Q D_i \ldots D_N$ and the SF nb' exists. Q. E. D.

Given the SF aNb such that $\overset{\leq}{a}$, $a \leq N$ and that $N \rightarrow n$, we would next like to consider under what conditions a context sensitive production is needed to reduce the SF anb. Clearly, if the grammar is SP, then $n > b$ and there is no need to use a CSP. However, under the new definitions for simple precedence, hereafter referred to as ESP, four additional cases arise.

Case I $a < n$, $n = b$

If b is of the form cd with the properties $\overset{=}{c}$ and $c > d$ then by adding the CSP, $Nc \rightarrow nc$, the proper reduction can be made. If b cannot be factored into the above form, it must be of the form $\overset{\leq}{c} > d$ for some c and d. Then, by Theorem 3.1 there exists a SF anb' and by Theorem 3.2 a SF aNb', which will delay the finding of the proper CSP until the sentential forms anb' and aNb' are examined.

Case II $a = n$, $n > b$

From the hypothesis a can be factored into ef such that $\overset{\leq}{e} < \overset{=}{f}$ and the CSP $fN \rightarrow fn$ can be used to find the proper reduction.

Case III $a = n$, $n = b$

This is a combination of Cases I and II. If as in Case I $b \equiv \overset{=}{c} > d$ then the CSP $fNc \rightarrow fnc$ can be used to find the proper reduction. If, on the other hand, b cannot be factored, then as in Case I, there exists a SF anb', and the CSP will be found when the SF aNb' is examined.

Case IV $a \leq n,\ n < b$

In this case no reduction can be made. But, by Theorem 3.1 there exists a SF anb' and by Theorem 3.2 a SF aNb', which will delay the finding of the proper CSP until the sentential forms anb' and aNb' are examined.

The new definitions would be of little use if the entire sentential form had to be examined in order to decide what CSP was to be applied. Therefore, we will define an entity called a phrase which will contain sufficient information for the decision.

Define a phrase as any sub-string of a SF for which the following properties are true.

Definition 3.2

If abc is a SF and

1. $a < b$
2. $b > c$
3. $\bar{\bar{b}}$

then b is a phrase.

Now, if it is the case that all phrases are known together with the CSP's which reduce them, then the algorithm for parsing is identical with the algorithm for parsing simple precedence. To this end, we would next like to consider a method for determining the phrases of a grammar.

Lemma 3.1

If for some symbol pair (A,B) $A = B$ then $X,\ X \in HS(B)$, $A \neg > X$

Proof:

It is sufficient to prove that $(\mathcal{S}H) \cdot \mathcal{L} = 0$

Expanding the above

$$(E + L \cdot G_1)\, H \cdot (G \cdot E^C \cdot L^C) = 0$$

$$(EH) \cdot G \cdot E^C \cdot L^C) + (L \cdot G_1)H \cdot G \cdot E^C \cdot L^C) = 0$$

and since $EH = L$

$$(EH \cdot G \cdot E^C \cdot L^C) = 0$$

Assume $(L \cdot G_1)H) \cdot G \cdot E^C \cdot L^C) \neq 0$

then there exist I,J,K, and L such that

$$L(I,K)G_1(I,K)H(K,J)G(I,J)E^C(I,J)L^C(I,J) = 1$$

But $L(J,K)H(K,J)$ implies $L(I,J)$ which contradicts $L^C(I,J)$. Therefore

$$(\mathcal{S}H) \cdot \mathcal{L} = 0$$

Q. E. D.

Lemma 3.2

If for some symbol pair (A,B), $A = B$ then X, $X \in TS(A)$, $X \neg < B$

Proof:

It is sufficient to show that $(T'\mathcal{S}) \cdot \mathcal{L} = 0$

Expanding the above we get

$$T'(E+L \cdot G_1) \cdot (L \cdot E^C \cdot G_1^C) = 0$$

$$((T'E) \cdot L \cdot E^C \cdot G_1^C) + ((T'(L \cdot G_1)) \cdot L \cdot E^C \cdot G_1^C) = 0$$

but $T'E = G_1$ therefore

$$((T'E) \cdot L \cdot E^C \cdot G_1^C) = 0$$

Assume $(T'(L \cdot G_1)) \cdot L \cdot E^C \cdot G_1^C \neq 0$

then there exists an I,J,K, and L such that

$T(K,I)L(K,J)G_1(K,J)L(I,J)E^C(I,J)G_1^C(I,J)G_1(K,J) = 1$. This implies there exists an M such that $T(M,K)E(M,J)$

$T(M,K)T(K,I)$ implies $T(M,I)$

and $T(M,I)E(M,J)$ implies $G_1(I,J)$

But $G_1(I,J)$ and $G_1^c(I,J)$ is a contradiction,

therefore $(T'\delta)\mathcal{L} = 0$

Q. E. D.

Lemma 3.3

If for some symbol pair (A,B) $A < B$ then $X, X \in HS(B)$, $A \neg > X$

Proof:

It is sufficient to prove that $(\mathcal{L}H)\mathcal{G} = 0$

Expanding the above we get $((L \cdot E^c \cdot G_1^c)H) \cdot G \cdot E^c \cdot L^c = 0$

Assume $((L \cdot E^c \cdot G_1^c)H) \cdot G \cdot E^c \cdot L^c \neq 0$

then there exists an I,J, and K such that

$$L(I,K)E^c(I,K)G(I,K)H(K,J)G(I,J)E^c(I,J)L^c(I,J)$$

but $L(I,K)H(K,J)$ implies $L(I,J)$ which contradicts $L^c(I,J)$.

Q. E. D.

3.3 Generating Strings

Definition 3.3

Define the 3 Link matrices as follows:

Let $X \in TS(V_I)$, $Y \in HS(V_J)$

$Q_1(I,J) = 1$ if $X = Y$

$Q_2(I,J) = 1$ if $V_I = Y$

$Q_3(I,J) = 1$ if $X = V_J$

or equivalently in matrix notation

$Q_1 = T\mathcal{G}H'$

$Q_2 = \mathcal{G}H'$

$Q_3 = T\mathcal{G}$

Lemma 3.4

If a grammar G is SP then $(Q_1 + Q_2 + Q_3) \cdot (\mathcal{L} + \mathcal{G} + \mathcal{S}) = 0$

Proof:

Let T_R and H_R represent the reflexive closure of T and H respectively.

Since G is SP, $\mathcal{L} \equiv L$, $\mathcal{G} \equiv G$ and $\mathcal{S} \equiv E$

and $(Q_1 + Q_2 + Q_3) = T_R E H' + TE$

$(L + G + E) = T_R' E H_R$

assume $(T_R E H' + TE) \cdot (T_R' E H) \neq 0$

then there exists an I,J,K,L,M and N such that one of the following is true.

1) $T_R(K,I)E(K,L)H_R(L,J)T_R(I,M)E(M,N)H(J,N)$

2) $T_R(K,J)E(K,L)H_R(L,J)T(I,M)E(M,J)$

If number 1 is true then $T_R(K,I)$ and $T(I,M)$ imply $T_R(K,M)$. $H_R(L,J)$ and $H(J,N)$ imply $H_R(L,N)$. $T_R(K,M)E(K,L)H_R(L,N)$ implies there is a relation between M and N. But $E(M,N)$ implies $V_M \triangleq V_N$ which forces $K \equiv M$ and $L \equiv N$. Then rewriting 1 we get

$T_R(M,J)E(M,N)H_R(N,J)T_R(I,M)H(J,N)$

but $H_R(N,J)H(J,N)$ implies $H(N,N)$

and $E(M,N)H(N,N)$ implies $V_M < V_N$ which will contradict $E(M,N)$.

Therefore, 1 is false.

If 2 is true then $T_R(K,I)$ and $T(I,M)$ implies $T_R(K,M)$. $T_R(K,M)E(K.L)H_R(L,J)$ implies there exists a relation between V_M and V_J. But $E(M,J)$ implies $V_M = V_J$ therefore $K \equiv M$ and $L \equiv J$. Then rewriting 2 we get

$T_R(M,I)E(M,L)H_R(J,J)T(I,M)E(M,J)$.

But $T_R(M,I)$ and $T(I,M)$ implies $T(M,M)$. $T(M,M)$ and $E(M,J)$ implies $V_M > V_J$ which contradicts with $V_M = V_J$. Therefore 2 is false.

Q. E. D.

Intuitively the Q matrices predict when there exists a derivation on a symbol which will link to its context though the equals relation.

Given a string c of length K, we define two sequences <D> and <M> each consisting of the K elements $D_1,D_2,\ldots,D_K$ and $M_1,M_2,\ldots,M_K$ respectively. We define the relation R on these two sequences as follows:

1. $D_i R D_{i-1}$ if $C_{i-1} = C_i$
2. $M_i R D_{i-1}$ if $Q_2(C_{i-1},C_i)$
3. $M_i R M_{i-1}$ if $Q_1(C_{i-1},C_i)$
4. $D_i R M_{i-1}$ if $Q_3(C_{i-1},C_i)$

Letting R_T be the transitive closure of R, we define linked strings as follows:

Definition 3.4

c is a (M,M) linked string if $M_K R_T M_1$

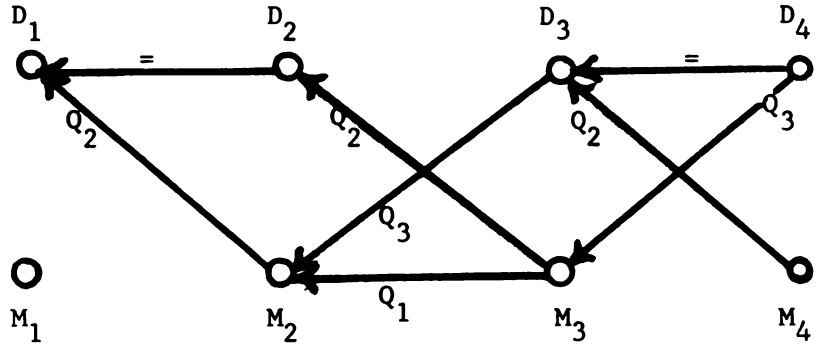
c is a (M,D) linked string if $D_K R_T M_1$

c is a (D,M) linked string if $M_K R_T D_1$

c is a (D,D) linked string if $D_K R_T D_1$

Example 3.2 Finding Linked Strings

For the string $a = A_1A_2A_3A_4$



GRAPH OF THE RELATION R

the following are linked strings

STRING	TYPE
$A_1A_2A_3A_4$	(D,D), (D,M)
$A_2A_3A_4$	(M,D), (D,D), (M,M)
$A_1A_2A_3$	(D,M), (D,D)

Definition 3.5

A string cXd is said to be a generating string, GS(X), with distinguished non-terminal X if the string has all of the following properties.

Let $|c| \equiv I$, $|d| \equiv J$

1. $\overset{\leq}{c}$

2. At least one of the following is true:

   a. c is a (D,M) or (M,M) linked string and $Q_1(C_I,X)$

   b. c is a (D,D) or (M,D) linked string and $Q_2(C_I,X)$

   c. $I \equiv 0$

3. At least one of the following is true:

   a. $Q_3(X,D_1)$ and $\bar{\bar{d}}$

   b. $J \equiv 2$ and $D_1 < D_2$

   c. $J \equiv 0$

4. $C_I \leq X$

Definition 3.6

We define a generator as an ordered pair (x,N) where x is a generating string and N is the distance to the distinguished non-terminal when measured from the right hand end of the string.

If (cXd,K) is a generator we can produce a new set of generators by applying to the distinguished non-terminal X all of its alternatives. For each alternative, say $X \rightarrow b$, we examine the string cbd for new generators. First, since cXd was a GS, by Lemma 3.1 cb has the property $\bar{\bar{c}} \leq \bar{\bar{b}}$. Now for each non-terminal Z in cb, factor cb into rZs and examine the string $rZ\,sd$ for $GS(Z)$. Since, in the definition of a GS, the strings to the left and the right of the distinguished non-terminal have to satisfy disjoint requirements, we will examine each side separately.

For the left side, we will factor r in two ways

$$r \equiv tu \qquad |u| \equiv I$$

$$r \equiv ln \qquad |n| \equiv J$$

where u is the longest of (M,D) or (D,D) linked strings and n is the longest of (M,M) or (D,M) linked strings. Then there are four possible cases for the left side.

Case I: $Q_2(U_I,Z) \neg Q_1(N_J,Z)$

Then the left hand side is uZ since this satisfies requirement 2.b in definition 3.5.

Case II: $\neg Q_2(U_I,Z)\ Q_1(N_J,Z)$

Then the left side is nZ since this satisfies requirement 2.a in definition 3.5.

Case III: $Q_2(U_I,Z)\ Q_1(N_J,Z)$

Then, if I is greater than J, the left hand side is uZ as in Case I.

If I is less than J then the left side is nZ as in Case II.

If $I \equiv J$ then $u \equiv n$ and the left side is nZ which is equivalent to uZ.

Case IV: $\neg Q_2(U_I,Z)\ \neg Q_1(N_J,Z)$

Then the left side is just Z which satisfies 2.c in the definition 3.5.

For the right side we factor sd as follows:

$$sd \equiv ef \equiv p \quad |e| \equiv I$$

where $e \overset{=}{>} f$

There are two possible cases for the right side.

Case I: I is greater than 0 and $Q_3(Z,e_1)$

Then the right side is e which satisfies 3.a of definition 3.5.

Case II: $\neg Q_3(Z,e_1)$ or $I \equiv 0$

If $Z = P_1$, then the right side is Λ.

If $Z \neq P_1$, then the right side is $P_1{}^{\perp}$.

Then the constructed string satisfies the requirements of a GS.

The set of generators for a grammar is defined to be the set of all generators that can be derived from $(\sigma,1)$ where σ is the goal of the grammar.

In each step of the analysis the precedence analyzer will return a phrase for reduction. In each case we know that the string to the left of the phrase has the property $\leq$ and that the next reduction must be based solely on the phrase presented.

A GS is used to predict in which context a production may become context sensitive by supplying a string which may be the result of a reduction of a phrase. For a given GS, say aBc and a production $B \rightarrow b$, we try to determine whether a phrase will exist in the string abc. The same cases will arise that arose when we considered the sentential forms.

Case I: $a < b$, $b = c$

From the definition of a GS, c may have one of two forms.

Form 1: $\overset{=}{c}$

Then the CSP is $Bc \rightarrow bc$ since by the construction of the GS we know that $c>$ the context in which the GS was derived.

Form 2: $c \equiv C_1C_2$, $C_1 < C_2$

The only way in which this case could arise is when, in the formation of the GS, the right side could not be factored into some string ef such that $\overset{=}{e} > f$. It was specifically for this purpose that C_1 and C_2 were placed in the GS as such. Intuitively, what we are saying is that it is not possible for a reduction to take place on this string because it is not a phrase. However, this GS is still to be included in this set because of the GS's which may be derived from it. Also, it is not necessarily the case that all the tail symbols of the distinguished non-terminal will be equal to C_1. Therefore, C_1 also keeps track of the context in which case a tail symbol of the distinguished non-terminal

can appear so that if the tail symbol is $> C_1$ the production will be found.

Case II: $a = b$, $n > b$

In this case a production is always found, since by definition a is always factorable into some ef such that $\overset{\leq}{e} < \overset{=}{f}$ and the CSP is just $fB \rightarrow fb$.

Case III: $a = b$, $b = c$

This is just a combination of Cases I and II. If a reduction exists for the right side, the CSP would be $fBc \rightarrow fbc$. Otherwise, there is no reduction for the same reason given in Case I.

Case IV: $a \leq b$, $b < c$

Clearly, in this case there are no reductions since there exists no phrase in this context which contains b.

One can now see that the method for finding CSP's is already embedded in the process of determining the GS's and, therefore, one can combine them into a simple algorithm.

Starting with the starting type of a grammar, we obtain a set of generators and a corresponding set of CSP's. The cardinality of these sets is not necessarily finite. However, at this point, we will only consider the cases of finite cardinality and investigate the infinite case in the next chapter.

Now we will examine, in more detail, a single step in the algorithm. Given a GS(B) of the form aBc with $\overset{=}{c}$, we will examine all the possibilities that will arise and show that the algorithm is complete.

First, we note that the only way in which c can be $\overset{=}{c}$ is if the GS from which this GS was derived had on the right side of its non-terminal B, a string which was of the form cd and that there exists a tail symbol of B, say X, such that X = c. But, clearly, not all the tail symbols of B are necessarily related to c by the = relation, so that two other cases are possible.

Case I: X > c

In this case, the context of c is not needed for the reduction but is necessary for establishing the existence of the reduction.

Case II: X < c

This case can only arise if B ≠ c by Lemma 3.2. In this case, c is to be reduced since it is a phrase. But since c is a phrase, it must contain as a sub-string, at least one right side of some reduction. And since c is a sub-string in a GS, it must have been derived from some mapping on some distinguished non-terminal of some GS. Therefore, there exists a GS which will produce a string c when its distinguished non-terminal is mapped and the proper reduction will be found.

Now we will examine the other side. From Lemma's 3.1 and 3.3 we know that a < X or a = X for each X a head symbol of B since by the definition of a GS, a ≤ B. Therefore, either there is or is not a left context, but it is never the case that a > X. We also note that if there is a left context, then the production thus found cannot be applied in this context without its left context since, by Lemma 3.3, all the tail symbols of the right-most

symbol of the left context cannot be > the head symbol, X, of B for which a = X.

In the GS's, we have examined all the contexts in which a non-terminal can find itself in a sentential form and eliminated only those contexts where the precedence relations were such that they could not become part of a phrase. Therefore, if there are a finite number of GS's, then there are a finite number of productions and they will be enumerated by the algorithm.

Before discussing the case of an infinite set of GS's, the same example which was done earlier in this chapter will be used to demonstrate the algorithm.

Example 3.3

A → B,B
B → [A]
B → [L]
B → L

ESP precedence matrix

	A	B	[	]	,	L
A	0	0	0	=	0	0
B	0	0	0	>	=	0
[	=	<	<	0	0	=
]	0	0	0	>	>	0
,	0	=	<	0	0	<
L	0	0	0	=	>	0

Q_1

	A	B	[	]	,	L
A	0	0	0	0	0	0
B	0	0	0	0	0	0
[	0	0	0	0	0	0
]	0	0	0	0	0	0
,	0	0	0	0	0	0
L	0	0	0	0	0	0

Q_2

	A	B	[	]	,	L
A	0	0	0	0	0	0
B	0	0	0	0	0	0
[	1	1	0	0	0	0
]	0	0	0	0	0	0
,	1	0	0	0	0	0
L	0	0	0	0	0	0

Q_3

	A	B	[	]	,	L
A	0	0	0	1	1	0
B	0	0	0	1	0	0
[	0	0	0	0	0	0
]	0	0	0	0	0	0
,	0	0	0	0	0	0
L	0	0	0	0	0	0

GS	
A	1
B	1
[A]	2
B]	2
[B	1

MAPPINGS

B ; B
[A] , [L] , L
[B , B]
[A]] , [L]] , L]
[[A] , [[L] , [L

CSP

A → B , B
B → [A]
B → [L]
B → L
B] → L]
[B → [L

CHAPTER IV

Infinite Strings and Equal Right Sides

In chapter III the concepts of ESP were introduced together with a method for generating the CSP's necessary for the proper operation of the precedence analyzer. It was stated that it is not necessarily the case that the set of CSP's, thus derived, will be finite. Therefore, we would next like to investigate the case of the infinite CSP's leaving the problem of equal right sides to the latter sections of this chapter.

4.1 Infinite Context Sensitive Productions

Infinite CSP arise in one of three ways. If there exists a derivation of the form $A \Rightarrow bAc$,

Then,

Case I: $\overline{\overline{b}} = b$ and $\overline{\overline{c}}$ or $c \neq c$

Case II: $\overline{\overline{c}} = c$ and $\overline{\overline{b}}$ or $b \neq b$

Case III: $\overline{\overline{c}} = c$ and $\overline{\overline{b}} = b$

In case I, the set of allowable phrases will contain $db*bAe$ where d and e are part of the context in which A may appear. A similar set exists for the other cases.

Definition 4.1 A symbol, P, is said to have an LL bound, K, if for every phrase which starts with P, there exists a reduction within K symbols to the right of P.

Defin

for e

withi

Defin

for

with

Defi

for

with

appa

dist

leng

whe

in

and

CSI

in

De

of

do

fo

Definition 4.2 A symbol P is said to have an LR bound, K, if for every phrase which starts with P, there exists a reduction within K symbols to the left of the right hand end of the phrase.

Definition 4.3 A symbol P is said to have an RR bound, K, if for every phrase which ends with P, there exists a reduction within K symbols to the left of P.

Definition 4.4 A symbol P is said to have an RL bound, K, if for every phrase which ends with P, there exists a reduction within K symbols to the right of the left hand end of the phrase.

Given phrases generated by either Case I or Case II, it is apparent that the reduction to be made always lies a finite fixed distance from one boundary of the phrase. For each phrase p of length K, we define a six-tuple,

$$<LL,LR,RR,RL,P_1,P_K>$$

where LL,LR, RR, and RL are the values of the bounds defined in definitions 4.1 through 4.4 and P_1 and P_K are the first and last symbols of the phrase p. For each given finite set of CSP's, there exists a maximum LL,LR,RR and RL for each symbol in the grammar.

Definition 4.5 The dominant bound of a production is the smallest of the four bounds (LL,LR,RR,RL) of a production.

The bounds associated with each symbol are the maximum dominant bounds for that symbol. They can be easily found as follows:

1. For each six-tuple defined above, we construct four 3-tuples <B,V,S>. Where B is the type of the bound (LL,LR,RR,RL), V is its value and S is the associated symbol.

does

cess

than

when

for

sub-

stri

by r

a. Construct one 3-tuple for the dominant bound, if more than one is dominant, construct one for each such dominant bound.

b. Construct one 3-tuple for each of the remaining bounds with a value one greater than the dominant value.

2. For each symbol and bound type, there exists a maximum value V, which is the maximum bound for that symbol and bound type.

3. Replace the bounds in each six-tuple by the corresponding maximum bound if the current bound is less than the maximum.

The above process is repeated until a set of maximum bounds does not change for two consecutive iterations. Clearly the process must halt since the size of the bounds cannot become greater than the largest original bound in the set.

The bounds, thus found, are used by the parsing algorithm whenever a phrase is found. The analyzer finds the minimum bound for the given end symbols and then tries to match the indicated sub-string of the phrase to the CSP set. In this way, infinite strings of Case I and Case II are easily handled.

Example 4.1 Consider the simple grammar:

$S \rightarrow AS$

$S \rightarrow AA$

In this grammar, A is both $< A$ and $=A$.

Under the formation rules of ESP, this conflict is resolved by making the relation between the A's equal. But there exists the

following two sentential forms, A*S and A*AA, which imply an infinite set of productions. If we generate the ESP CSP's for all strings of length less than or equal to 4, we get the following set of productions and bounds:

S	AA	<2,2,2,2,A,A>	<LL,2,A>	<LR,2,A>	<RR,2,A>	<RL,2,A>
AS	AAA	<3,2,2,3,A,A>	<LL,3,A>	<LR,2,A>	<RR,2,A>	<RL,3,A>
AAS	AAAA	<4,2,2,3,A,A>	<LL,4,A>	<LR,2,A>	<RR,2,A>	<RL,4,A>
S	AS	<2,2,2,2,A,S>	<LL,2,A>	<LR,2,A>	<RR,2,S>	<RL,2,S>
AS	AAS	<3,2,2,3,A,S>	<LL,3,A>	<LR,2,A>	<RR,2,S>	<RL,3,S>
AAS	AAAS	<4,2,2,3,A,S>	<LL,4,A>	<LR,2,A>	<RR,2,S>	<RL,4,S>

Maximum Bounds

	LL	LR	RR	RL
A	3	2	2	3
S	3	2	2	3

New six tuples

<3,2,2,3,A,A>	<LL,3,A>	<LR,2,A>	<RL,2,A>	<RL,3,A>
<3,2,2,3,A,A>	<LL,3,A>	<LR,2,A>	<RL,2,A>	<RL,3,A>
<4,2,2,3,A,A>	<LL,3,A>	<LR,2,A>	<RL,2,A>	<RL,3,A>
<3,2,2,3,A,S>	<LL,3,A>	<LR,2,A>	<RL,2,S>	<RL,3,S>
<3,2,2,3,A,S>	<LL,3,A>	<LR,2,A>	<RL,2,S>	<RL,3,S>
<4,2,2,3,A,S>	<LL,3,A>	<LR,2,A>	<RL,2,S>	<RL,3,S>

Maximum Bounds

	LL	LR	RR	RL
A	3	2	2	3
S	3	2	2	3

Theorem 4.1

Let K be the length of the longest CSP in the set which contains all the finite CSP's and at least one representative of each potentially infinite CSP and let S_K denote the set of all CSP's of length less than or equal to K. Then, if the set of bounds for the set S_K is the same as the set of bounds for the set S_{K+M}, where M is the maximum bound in the set S_K, then the set of bounds for the set S_K is sufficient for the set of infinite CSP's.

Proof:

Assume Case I. Then by the hypothesis there exists an N such that

$| ab^N d | \leq K < | ab^{N+1} d | \leq | ab^{N+J} d | \leq K+M < | ab^{N+J+1} d |$

since $| b | \leq M$. If there exists a RR or LR bound for $ab^N d$ in S_K it must be equal to $| bd | + C$, where C is a constant. But this is a sufficient bound for the entire set of CSP's of the form $ab^* d$ and therefore it will remain unchanged in the set S_{K+M} .

If on the other hand a LL or RL bound was the dominant bound for the phrase $ab^N d$ in the set S_K then the maximum bound M must satisfy the following for some I.

$$|ab^{N+I} d | \leq M < |ab^{N+I+1} d |$$

But $ab^{2N+I} d$ is in the set S_{K+M} which would imply that the dominant bound for $ab^{2N+I} d$ is greater then or equal to $| ab^{2N+I} |$ which is greater than the maximum bound M in S_K . Therefore the set of bounds for the two sets S_K and S_{K+M} could not be the same.

A similar proof can be used for Case II.

Clearly Case III can't be effectively handled by using bounds and therefore it is only necessary to show that if a Case III phrase exists then the set of bounds for the sets S_K and S_{K+M} must not be the same.

If there exists a Case III phrase then by the hypothesis there exists an N such that

$|ab^N cd^N e| \leq K < |ab^{N+1} cd^{N+1} e| \leq |ab^{N+J} cd^{N+J} e| \leq K+M < |ab^{N+J+1} cd^{N+J+1} e|$

since $|b^N cd^N| \leq M$. Then the maximum bound M must satisfy the following for some I.

$$|b^{N+I} cd^{N+I}| \leq M < |b^{N+I+1} cd^{N+I+1}|$$

But this implies that $ab^{2N+I} cd^{2N+I} e$ is an element of the set S_{K+M} .

The dominant bound for $ab^{2N+1}cd^{2N+1}e$ is greater than the maximum bound for the set S_K and therefore the set of bounds for S_K and S_{K+M} cannot be the same.

Q. E. D.

4.2 A Canonical Form

Even if a grammar exhibits only strings of Case I and Case II, there is no guarantee that the set of bounds will exist since the strings may start and end with the same symbol. For example, if Xbc*dY and X1m*nY were phrases of Cases I and II respectively, then the LL bound of X would be fixed for Case I and potentially infinite for Case II. Therefore, we would next like to consider a canonical form for a grammar for which, at most, only phrases of Case I may arise. If each production of the grammar is of the form

$$A \rightarrow bC \qquad b \in (V - \Sigma)^* \qquad C \in \Sigma$$

then the following matrices arise.

$$E = \begin{bmatrix} E_{11} & E_{12} \\ 0 & 0 \end{bmatrix} \qquad H = \begin{bmatrix} H_{11} & H_{12} \\ 0 & 0 \end{bmatrix} \qquad T = \begin{bmatrix} 0 & T_{12} \\ 0 & 0 \end{bmatrix}$$

$$L = \begin{bmatrix} E_{11} & H_{11} & E_{11} & H_{12} \\ 0 & & 0 & \end{bmatrix} \qquad G = \begin{bmatrix} 0 & 0 \\ T'_{12}\,E_{11}(H_{11} + U) & T'_{12}(E_{11}\,H_{12} + E_{12}) \end{bmatrix}$$

$$\mathcal{S} = E,\ \mathcal{L} = L \cdot E^c,\ \mathcal{G} = G$$

From the above some elementary properties are easily obtained.

Property 1. There are no $>$ conflicts

Property 2. The right end of every phrase is known

Property 3. The parse of any string is an LRCP.

Property 4. At most only Case I phrases exist.

Property 5. There always exists a set of bounds.

The above form always exists for any grammar since this is just the reverse of Griebach Normal Form [Gr 65].

Several authors have pointed out that a sufficient condition for parsing a grammar in Griebach Normal Form by a top-down method is that each alternative of a given definition starts with a unique terminal symbol. This method is based upon the fact that if a non-terminal is to be replaced by one of its alternatives, one need only look one symbol ahead on the input string to uniquely determine which alternative to apply. In these grammars, there may be many productions which have equal right sides, but, in a sense, only a small subset of the productions can be applied at a given time. Therefore, the restriction for the given subset, that the productions be unique, reduces for top-down parsing to the case of unique starting terminal symbols within the alternatives of the definition. It would be convenient to have a similar set of subsets of productions for bottom-up parsing. Taking advantage of the LRCP of the canonical form, we will next investigate such a procedure.

Definition 4.6 The production grammar G_P of a grammar G is defined as follows: $G = \langle V,\Sigma,P,\sigma\rangle$ $G_P = \langle V',\Sigma',P',\sigma\rangle$. Let i denote the i-th production in P and let $\Sigma' = \{i \mid i \equiv 1,2,\ldots,\#P\}$ and let $V' \equiv \{V - \Sigma\} \cup \{\Sigma'\}$.

For each production in P, say the i-th $R \rightarrow W_1Z_1W_2Z_2\ldots W_KZ_K$ where $W_j \in \Sigma^*$ and $Z_j \in (V - \Sigma)^*$ define a corresponding production for P', $R \rightarrow Z_1Z_2\ldots Z_Ki$.

Now for each production in P, there is a corresponding production in P' which has the same non-terminal symbols in the same order and a single terminal symbol at the right end which represents the production number in P.

Theorem 4.2

The language generated by a production grammar is the set of allowable LRCP's for the grammar.

Proof:

Clearly, the language generated by the production grammar contains only production numbers since these are the only terminal symbols in the grammar. If one starts to derive an SF in G_P using an RMD, then at any point in the derivation, the SF will have the form zAb where $z \in (V' - \Sigma')^*$, $A \in (V' - \Sigma')$ and $b \in (\Sigma')^*$. Now, whenever a production is applied to an SF, using an RMD on a production grammar, its production number is recorded immediately to the right of its application and immediately to the left of the most recent previous application. Therefore, the string generated by the RMD is just the reverse of the application of the productions in the RMD. But an LRCP is just the reverse of an RMD and, since there exists an RMD for every string in the language, then the language generated by the production grammar is the set of LRCP's.

Q. E. D.

Theorem 4.3

The set of LRCP's of a grammar G and its production grammar are the same. Given any RMD, at any point in the derivation, the number and order of the non-terminals in the SF's of the two

grammars are the same. This is evident from the construction of the production grammar. Now, since the right-most non-terminal in each of the two SF's is the same, if corresponding productions are applied to this non-terminal in each of the respective SF's, the resulting SF's are still identical with respect to non-terminal symbols. Therefore, for every derivation in G, there exists a corresponding derivation in G_P and visa versa.

Q. E. D.

It was noted in Chapter II that the only way a symbol B could appear immediately to the right of a symbol A was if $A <$, $=$ or $> B$. Therefore, if the precedence relations are known for the terminal symbols of a production grammar, then given any production, the set of production which may legally follow it in an LRCP are given by these relations. Since the form of the production grammar is identical with the canonical form described above, the relation matrix N for the production set is just $N = T'_{12} \ (E_{11} H_{12} + E_{12})$. The restriction on equal right sides can now be relaxed by only requiring productions within any given subset to be unique. Clearly, this technique can be used for any bottom-up method which is governed by an LRCP. As an example, a precedence grammar, with equal right sides, will use the above technique to resolve this problem.

Example 4.2

1 S → LRD 2 S → MTC 3 L → C

4 M → D 5 R → ARB 6 R → AB

7 T → ATB 8 T → AB

Production Grammar

1 S → LR1 2 S → MT2 3 L → 3

4 M → 4 5 R → R5 6 R → 6

7 T → T7 8 T → 8

Precedence Matrix for G

	S	L	M	R	T	A	B	C	D
S	0	0	0	0	0	0	0	0	0
L	0	0	0	=	0	<	0	0	0
M	0	0	0	0	=	<	0	0	0
R	0	0	0	0	0	0	=	0	=
T	0	0	0	0	0	0	=	=	0
A	0	0	0	=	=	<	=	0	0
B	0	0	0	0	0	0	>	>	>
C	0	0	0	>	0	>	0	0	0
D	0	0	0	0	>	>	0	0	0

The N Matrix for G

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	1
5	1	0	0	0	1	0	0	0
6	1	0	0	0	1	0	0	0
7	0	1	0	0	0	0	1	0
8	0	1	0	0	0	0	1	0

Clearly the first applicable production must be a head symbol of the starting type in the production grammar. For this particular grammar it must be either production 3 or 4.

The parse of a string in the language would proceed as follows:

DAAABBBC	Production to be applied	Allowable successors
<D>AAABBBC	4	8
<M<A<A<A=B>BBC	8	2,7
<M<A<A=T=B>BC	7	2,7
<M<A=T=B>C	7	2,7
<M=T=C>	2	-
S		

4.3 Equal Right Sides and Bounded Context Grammars

Floyd [F1 64] introduced the concept of bounded context analysis. This is another convenient method for handling equal right sides in ESP. If there exists a context of finite length, for which the decision can be made as to which production is to be applied, then if this context can be forced to appear with the phrase, the correct reduction can easily be found. But clearly, the mechanism for examining this extended context is already available in ESP. One need only change the appropriate relations to = in order to extend the contexts. In so doing, additional CSP's will be created which will perform the same function as Floyd's Bounded Context Productions. To illustrate this technique, we will examine another precedence grammar which contains equal right sides.

Example 4.3

S → CR
S → DT
R → LRB
R → LB
L → A
T → MTNB
T → MNB
M → A
N → B

The bounded context reductions needed to decide whether to apply L → A or M → A when an A is found are:

LA use L → A
CA use L → A
DA use M → A
MA use M → L

The precedence matrix with the forced equalities circled is:

	S	R	T	M	L	N	A	B	C	D
S										
R								=		
T						=		<		
M			=	<		=	⊜	<		
L		=			<		⊜	=		
N								=		
A		≥	≥	>	>	>	>	>		
B						>		>		
C		=			<		⊜			
D			=	<			⊜			

The

Th

ESP, cre

free gra

grammar

The four CPS's added to the production set are:

MM → MA	CL → CA
LL → LA	DM → DA

The techniques discussed in this chapter, when combined with ESP, create a powerful method for deterministic parsing of context-free grammars. Some of the more interesting ramifications of ESP grammars will be discussed in the next chapter.

CHAPTER V

LR(k) Grammars and their Relation to ESP Grammars

In this chapter we will show how ESP compares with other techniques for deterministic parsing of context free languages.

5.1 LR(k) Grammars

Knuth [Kn 63] defines a grammar to be LR(k) if a phrase x is uniquely determined by the entire string y to the left of the phrase x and k terminal symbols to the right of the phrase x. Both Floyd [Fl 68] and Tixier [Ti 67] consider LR(k) grammars to be the most general form for which there exists a constructable algorithm for an LRCP which is derived directly from the grammar. This method always results in a LRCP.

Lemma 5.1

There exist grammars which are not LR(k) for any k but which are ESP.

Proof:

Consider the following grammar:

S → EBC
S → ABD
E → A
B → (B)
B → L

This grammar is not LR(k) for any k since one cannot tell whether the phrase A should be reduced to E or not by simply looking at a finite number of terminal symbols to the right of A.

The key t

ust be b

this gra

The cir

which

with

approp

of the

Defin

if bo

Exa

T

The key to the problem is that C or D, upon which the solution must be based, can be an unbounded number of symbols away. But this grammar is ESP:

CSP
S → ABD
S → EBC
EBC → ABC
B → (B)
B → L

	S	B	E	(	)	L	A	C	D
S									
B					=			=	=
E	=		<		<				
(	=		<		<				
)					>			>	>
L					>			>	>
A	(=)		(<)		<̇				

The circled relations represent conflicts in the original grammar which were removed using ESP. Now the decision of what to do with A is delayed until a C or D is found and then the appropriate action is taken. Here we have also taken advantage of the fact that the parse is not a LRCP.

Definition 5.1

A grammar G_1 is said to be synonomous to a grammar G_2 if both of the following conditions hold:

1. Both G_1 and G_2 generate the same language.
2. If in each grammar the trivial productions of the form $X \rightarrow Y$ are removed by substitution, then the two grammars are isomorphic.

Example 5.1

G_1
S → AAB
S → AS
S → AC

G_2
S → RTQ
S → TS
S → RV
S → TC
R → A
T → A
Q → B
V → S

These two grammars are synonomous.

Knut

a right l

ing the e

with the

referred

this rig

G

c

w

P' i

Si

Knuth has shown that if a grammar is LR(k), then there exists a right linear grammar which will generate a set of strings representing the entire string to the left of the right-most phrase together with the phrase and k symbols to the left of the phrase, hereafter referred to as a sentential head. The technique for constructing this right linear grammar is as follows:

Given $G = \langle V, \Sigma, P, \sigma \rangle$,

construct $F_G = \langle V', \Sigma', P', \sigma' \rangle$

where $m = |P|$

$V' = \{[Z,a] | Z \in V,\ a \in \Sigma^K\} \cup \{[i] | i = 1 \ldots M\}$

$\Sigma' = \{V\} \cup \{\$\}$

$\sigma' = [\sigma, \$]$

P' is defined as follows:

For each production in P, say

$$A_x \rightarrow Z_{x_1} Z_{x_2} \ldots Z_{x_n}, \text{ let}$$

1. for each $Z_{x_i} \in V - \Sigma$ add to P' a production of the form

   $[A_x,w] \rightarrow Z_{x_1} \ldots Z_{x_{i-1}} [Z_{x_i} b]$ for each $[A,w]$ and for each

   $[Z_{x_i},b]$ where $b \in \{z | Z_{x_{i+1}} \ldots Z_{x_N} w \Rightarrow \hat{z}q,\ |z| = K\}$

2. $[A_x,w] \rightarrow Z_{x_1} Z_{x_2} \ldots Z_{x_N} w[X]$

3. $[X] \rightarrow \Lambda$

The w is defined to be any b that appears on the right side with the corresponding A_x.

Example 5.2

		LR(1)
1.	S → CT	[S,$] → C[T,$]
		[S,$] → CT$[1]
2.	S → DR	[S,$] → D[R,$]
		[S,$] → DR$[2]
3.	T → ATB	[T,$] → A[T,B]
		[T,$] → ATB$[3]
4.	T → AB	[T,B] → A[T,B]
		[T,B] → ATB[3]
5.	R → ARBB	[T,$] → AB$[4]
		[T,B] → ABB[4]
6.	R → ABB	[R,$] → A[R,B]
		[R,$] → ARBB$[5]
		[R,B] → A[R,B]
		[R,B] → ARBBB[5]
		[R,$] → ABB$[6]
		[R,B] → ABBB[6]
		[1] → Λ [4] → Λ
		[2] → Λ [5] → Λ
		[3] → Λ [6] → Λ

Definition 5.2

A finite state automata (FSA) is a 5-tuple $A = \langle S,\Sigma,m,S_o,F\rangle$, where

S is a set of states

Σ is a set of input symbols

m is a mapping of $S \times \Sigma$ into 2^S

S_o is the start state

F is a subset of S designated final states

The domain of m is extended to include strings of input symbols as follows:

$$m(s,Ab) = m(m(s,A),b)$$

Definition 5.3

If the range of m is S instead of 2^S then the FSA is said to be deterministic (DFSA).

Definition 5.4

The set of strings excepted by a FSA is the set T(A)

$$T(A) = \{z \mid m(s_o,z) \subseteq F\}$$

Definit

A

That is

which

that t

state

genera

contr

deter

accep

Lemma

wher

resu

forr

for

M'(

M(s

by

and

hyp

Definition 5.5

A FSA is said to be input pure iff

$\therefore \forall X \forall I \forall Y \forall J(m(X,I) \equiv m(Y,J) \text{ implies } I \equiv J)$

That is, for each state there exists one and only one input for which the state is in the range of the m mapping.

Given any right linear grammar, Ginsberg [Gi 65] has shown that there exists an effective procedure for finding a finite state automata which will accept those and only those strings generated by the right linear grammar. However, the automata thus contrived is not necessarily deterministic.

Rabin and Scott [Ra 60] have proved that for each non-deterministic automata there exists a deterministic automata which accepts the same set of input strings.

Lemma 5.2

For each DFSA there exists an equivalent input pure DFSA.

Proof:

For each state Z we define K new states $Z_{A_1}, Z_{A_2}, \ldots, Z_{A_K}$ where K is the number of unique input symbols $A_1, \ldots, A_K$ which result in transitions to Z. We replace each move function of the form $M(L,A_i) = Z$ with the move function $M'(L,A_i) = Z_{A_i}$ and for each move function of the form $M(Z,B) = P$ include $M'(Z_{A_i},B) = P$. If $Z \in F$ add $Z_{A_i} \ldots Z_{A_K}$ to F'. Now if $M(s_o,X) = D$ then $M'(s_o,X) = D_X$ and if $D \in F$ then $D_X \in F'$ by construction. Assuming that $M(s_o,aX) = Q$, $M'(s_o,aX) = Q'_X$ and $M(Q,Y) = R$, we precede by induction. By the induction hypothesis $M(s_o,aXY) = R$ and by the construction, if $M(Q,Y) = R$

then there exists $M'(Q'_I,Y) = R'_Y$ for all states Q'_I. In particular $M'(Q_X,Y) = R'_Y$. Therefore, $M'(s_o,aXY) = R'_Y$ and $R \in F$ then $R'_Y \in F'$.

Q. E. D.

Lemma 5.3

If a grammar is LR(K) there exists an input pure DFSA which accepts those and only those strings which represent the sentential heads of the grammar.

We would now like to construct a grammar which is synonomous to the original LR(K) grammar. Let $G_S = \langle V',\Sigma',P',\sigma'\rangle$ be the new grammar and $G = \langle V,\Sigma,P,\sigma\rangle$ be the LR(K) grammar for which $A = \langle S,X,M,s_o,F\rangle$ is the input pure DFSA. Define $V' = \{V\} \cup \{S\}$ and $\Sigma' = \Sigma$. The production set P' is constructed as follows:

For each production in P, say $Z_N \rightarrow A_1A_2\ldots A_M$, there exists at least one production in the LR(k) right linear grammar of the form

$$[Z_N,w] \rightarrow A_1A_2\ldots A_MB_{M+1}\ldots B_{M+K}[N]$$

Then for each sequence d of length M+K which will, from some state E, allow the automata to reach a final state $N_{B_{M+K}}$, add the following production to P'.

$$Z_N \rightarrow M(E,A_1)M(E,A_1A_2)\ldots M(E,A_1A_2\ldots A_N)$$

Since the automata is input pure each state represents one and only one symbol from V. By adding the following productions the correspondence between the states and the symbols in the automata is put in the production form. For each move function of the form $M(Z,A) = X$ add the production $A \rightarrow X$. Now each state represents a single symbol and each production which contains states is clearly

reducible to the original production in G by substitution. And since there exists at least one encoded (symbols replaced by states) production for each production G the constructed grammar is synonomous to the original grammar.

The precedence relations can be taken directly from the automaton and the new grammar since the transitive closure type operations are implicit in the automaton.

5.2 LR(1) Grammars

The synonomous grammar constructed from the automata is in general ambiguous and contains several productions which have the same right hand sides. However, the automata is deterministic and therefore, for each input, there is only one transition possible. Since the constructed grammar contains states of the automaton as non-terminal symbols, we can use this information to guide the precedence analyzer in choosing the proper production to apply. That is, the last state entered contains all the information necessary to determine the next state given any input. Since there are no productions with equal right sides which contain states, only the trivial productions need this consideration. The construction of the precedence matrix directly from the automaton and the non-trivial productions proceeds as follows:

The E matrix is obtained from the non-trivial productions of the synonomous grammar together with the forced equalities which uniquely determine which trivial production is to be applied.

Thus

$$E(I,J) \equiv 1 \quad \text{if} \quad V \rightarrow aV_I\, V_J b$$

$$\text{or} \quad V \rightarrow V_J \quad \text{and} \quad M(V_I, V_J) = V$$

We note that the two conditions are disjoint, one dealing only with relations between states and the other only between states and inputs.

The L matrix reflects only relations between states and is obtained directly from the m function.

Thus

$$L(I,J) \equiv 1 \quad \text{if} \quad m(V_I,V) \equiv V_J$$

The G matrix is just a relation between inputs since in a LRCP only terminal strings can appear to the right of a production and each production will have a terminal as its right most symbol by the definition of LR(1).

Thus

$$G(I,J) \equiv 1 \quad \text{if} \quad m(m(V,V_I),V_J)$$

The matrices have the following form:

STATES INPUTS

$$E \equiv \begin{bmatrix} E_{11} & E_{12} \\ 0 & 0 \end{bmatrix} \begin{matrix} \text{STATES} \\ \text{INPUTS} \end{matrix} \qquad L \equiv \begin{bmatrix} L_{11} & 0 \\ 0 & 0 \end{bmatrix} \qquad G \equiv \begin{bmatrix} 0 & 0 \\ 0 & G_{22} \end{bmatrix}$$

and therefore

$$\mathcal{E} = E, \ \mathcal{L} = L \cdot E^c, \ \mathcal{G} = G$$

The Q matrix may be found in the usual manner.

In order to prove that the grammar just constructed is ESP it is only necessary to show that there exists bounds for each production and that there are no equal right sides. If there were two encoded productions which had the same right side, then the original grammar could not have been LR(1) since the automaton would not have been able to distinguish between these two productions either. As for the trivial productions, since the automaton is deterministic,

the last state entered must uniquely determine the next state for a given input. Therefore, there are no equal right sides. Since the length of each production is known and the right end of every production is also known, i.e., there are no > conflicts, the RR bound is just the longest production ending in each symbol and this is always finite. In fact, the longest bound must be less than or equal to the longest production in the grammar plus 1.

Theorem 5.4

If a grammar is LR(1) then there exists a synonomous grammar G' which is an ESP grammar.

Knuth [Kn 64] has shown that if a language is deterministic then there exists an LR(1) grammar which generates it.

Theorem 5.5

If L is a deterministic language then there exists an ESP grammar which generates this language.

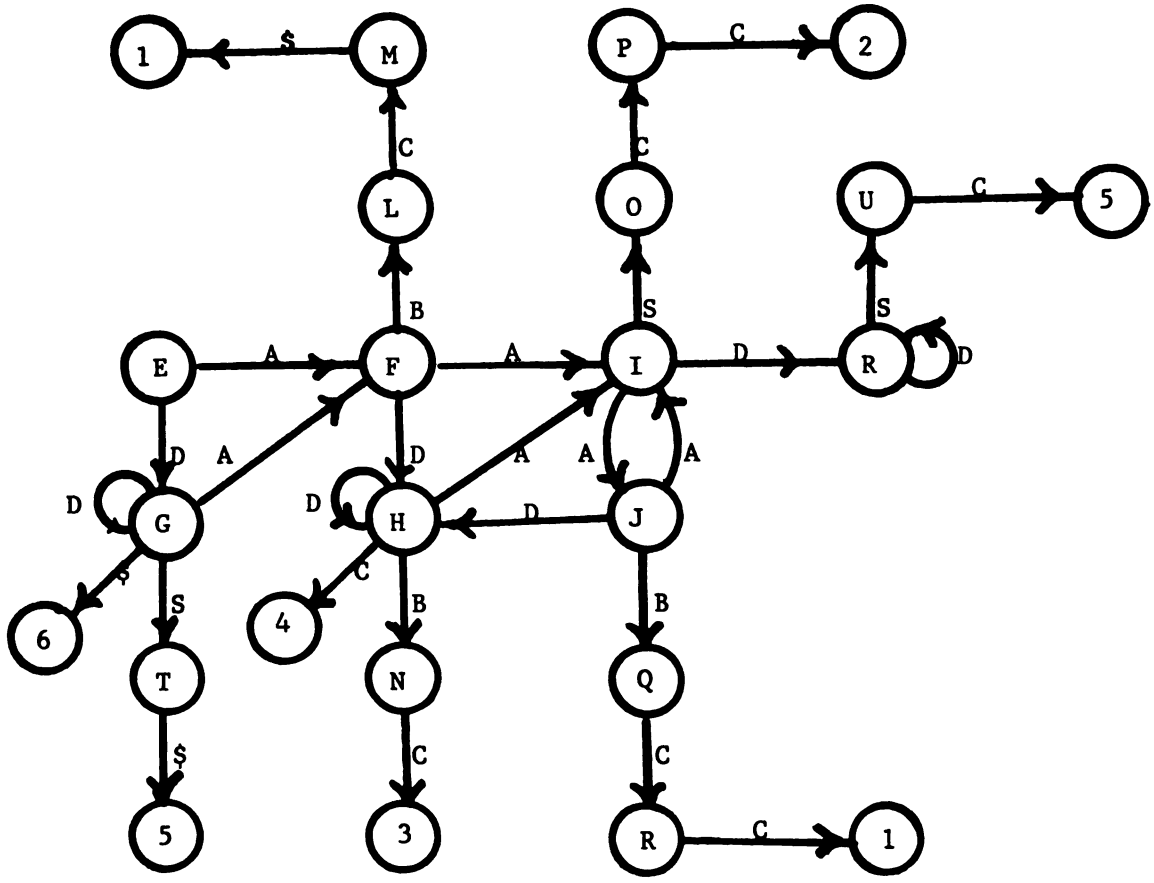
Example 5.3

1. S → ABC
2. B → AS¢
3. B → DB
4. B → D
5. S → DS
6. S → D

This grammar is not a simple precedence grammar since there are equal right sides and conflicts between B and C, and also between S and C. It is not bounded context since one must know how many A's have been stacked in order to determine whether to use B → D or S → D. But it is LR(1), as illustrated below:

LR(1) right linear grammar

[S,$] → A[B,C]
[S,$] → ABC$[1]
[S,C] → A[B,C]
[S,C] → ABCC[1]
[S,$] → D[S,$]
[S,$] → DS$[5]
[S,C] → D[S,C]
[S,C] → DSC[5]
[S,$] → D$[6]
[B,C] → D[B,C]
[B,C] → DBC[3]
[B,C] → DC[4]
[B,C] → A[S,C]
[B,C] → ASCC[2]
[1] → Λ → [4] → Λ
[2] → Λ → [5] → Λ
[3] → Λ → [6] → Λ



SYNONOMOUS GRAMMAR

F → A	N → B	S → JQR
G → D	O → S	S → FLM
H → D	P → C	B → IOP
I → A	Q → C	B → HN
J → A	R → C	N → B
K → D	T → S	S → GT
L → B	U → S	S → KV
M → C		G → S

FORCED EQUALITIES

FD,FA,FB,GS,GD,GA,HD,HA,HB,HC,IA,ID,IS,JB,JD,IA,KD,KS,KC,KA,LC, NC,OC,PC,QC,RC,VC.

Figure 5.1 Automaton for Example 5.3.

ESP Productions

F → A	HN → HN	FLM → FLC
FI → FA	JQ → JB	IOP → IOC
FL → FB	JH → JD	JQR → JQC
FH → FD	JI → JA	SC → JQRC
GF → GA	KK → KD	BC → IOPC
GG → GD	KJ → KA	S → FLM
GT → GS	KU → KS	SC → KUC
HH → HD	IO → IS	BC → HNC
HI → HA	IK → ID	S → GT
BC → HC	IJ → IA	S → G

Precedence Matrix

	F	N	S	G	O	H	P	B	I	Q	J	R	K	T	L	U	M	A	D	C
F	0	0	0	0	0	<	0	=	<	0	0	0	0	0	=	0	0	=	=	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	>
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	>
G	<	0	=	<	0	0	0	0	0	0	0	0	0	=	0	0	0	=	=	0
O	0	0	0	0	0	0	=	0	0	0	0	0	0	0	0	0	0	0	0	=
H	0	=	0	0	0	<	0	=	<	0	0	0	0	0	0	0	0	=	=	=
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=
B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	>
I	0	0	=	0	=	0	0	0	0	0	<	0	<	0	0	0	0	=	=	0
Q	0	0	0	0	0	0	0	0	0	0	0	=	0	0	0	0	0	0	0	0
J	0	0	0	0	0	<	0	=	<	=	0	0	0	0	0	0	0	=	=	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=
K	0	0	=	0	0	0	0	0	0	0	<	0	<	0	0	=	0	=	=	=
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	0	0	=
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	>	>	0
D	0	0	>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	>	>	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	>

An example parse	Production used
<AADADCCC	
<A>ADADCCC	F → A
<F=A>DADCCC	FI → FA
<F<I=D>ADCCC	IK → ID
<F<I<K=A>DCCC	KJ → KA
<F<I<K<J=D>CCC	JH → JD
<F<I<K<J<H=C>CC	BC → HC
<F<I<K<J=B>CCC	JQ → JB
<F<I<K<J=Q=C>CC	JQR → JQC
<F<I<K<J=Q=R=C>C	SC → JQRC
<F<I<K=S>CC	KU → KS
<F<I<K=U=C>C	SC → KUC
<F<I=S>CC	IO → IS
<F<I=O=C>C	IOP → IOC
<F<I=O=P=C>	BC → IOPC
<F=B>C	FL → FB
<F=L=C>	FLM → FLC
<F=L=M>	S → FLM

Lemma 5.4

Tnere exists a context sensitive language which has an ESP grammar.

Proof:

The grammar:

1 G → R
2 R → AF
3 ARE → AAFE
4 F$ → REC$
5 F$ → DC$
6 FE → DE
7 FC → RECC
8 FC → DCC
9 DE → DF
10 DRE → DZE
11 DZ → DR
12 RZ → RE
13 DC → BC
14 DB → BB

ESP productions:

R → AF D → B
F → REC F → DC
F → D G → R
E → F R → Z
RZ → RE D → R

The precedence matrix:

	R	F	D	E	Z	A	B	C	$
R	0	0	0	=	>	0	0	0	=
F	0	0	0	>	0	0	0	>	>
D	<	<	<	>	<	0	<	=	0
E	0	0	0	>	0	0	0	=	0
Z	0	0	0	>	0	0	0	0	0
A	<	=	<	0	0	<	<	0	0
B	0	0	0	0	0	0	>	>	0
C	0	0	0	0	0	0	0	>	>
$	=	0	0	0	0	<	0	0	0

This grammar generates $\$A^NB^NC^N\$$ which has been shown to be a context sensitive language. To show that the above grammar actually generates $\$A^NB^NC^N\$$ for $N \geq 1$ we will examine all sentential forms in a general derivation together with the precedence relations.

Sentential Form	Prod. No.	Resulting Sentential Form	Prod. No.	Resulting Sentential Form
G	1	$\lessdot\$=R=\$\gtrdot$		
R	2	$\lessdot\$\lessdot A=F\gtrdot\$$		
AF	4	$\lessdot\$\lessdot A\lessdot R=E=C\gtrdot\$$	5	$\lessdot\$A\lessdot D=C\gtrdot\$$
$AREC$	3	$\lessdot\$\lessdot A\lessdot A=F\gtrdot EC\$$		
$AAFEC$	6	$\lessdot\$\lessdot A\lessdot A\lessdot D\gtrdot EC\$$		
$\$A^NFE^{N-1}C^{N-1}\$$	6	$\lessdot\$\lessdot A^{\lessdot N}\lessdot D\gtrdot E^{N-1}C^{N-1}\$$		
$\$A^NDE^{N-1}C^{N-1}\$$	9	$\lessdot\$\lessdot A^{\lessdot N}\lessdot D\lessdot F\gtrdot E^{N-2}C^{N-1}\$$		
$\$A^ND^KFE^{N-K-1}C^{N-1}$	6	$\lessdot\$\lessdot A^{\lessdot N}\lessdot D^{\lessdot K}\lessdot D\gtrdot E^{N-K-1}C^{N-1}\$$		

Sentential Form	Prod. No.	Resulting Sentential Form	Prod. No.	Resulting Sentential Form
$\$A^N D^K E^{N-K} C^{N-1}\$$	9	$<\$<A^{\leq N}<D^{\leq K}<F>E^{N-K-1}C^{N-1}\$$		
$\$A^N D^{N-1} E C^{N-1}\$$	9	$<\$<A^{\leq N}<D^{\leq N-1}<F>C^{N-1}\$$		
$\$A^N D^{N-1} F C^{N-1}\$$	7	$<\$<A^{\leq N}<D^{\leq N-1}<R=E=C>C^{N-1}\$$	8	$<\$<A^{\leq N}<D^{N-1}<D=C>C^{N-1}\$$
$\$A^N D^{N-1} R E C^N\$$	10	$<\$<A^{\leq N}<D^{\leq N-1}<Z>E C^N\$$		
$\$A^N D^K Z E^{N-K} C^N\$$	11	$<\$<A^{\leq N}<D^{\leq K-1}<R>Z E^{N-K} C^N\$$		
$\$A^N D^{K-1} R Z E^{N-K} C^N\$$	12	$<\$<A^{\leq N}<D^{\leq K-1}<R=E>E^{N-K} C^N\$$		
$\$A^N R E^N C^N\$$	3	$<\$<A^{\leq N}<A=F>E^N C^N\$$		
$\$A^N C^N C^N\$$	13	$<\$<A^{\leq N}<D^{\leq N-1}<B>C^N\$$		
$\$A^N D^K B^{N-K} C^N\$$	14	$<\$<A^{\leq N}<D^{\leq K-1}<B>B^{N-K} C^N\$$		

Note that the only place in any derivation where there exists a choice of productions is when either 4, 5 or 7, 8 can be applied. If 5 or 8 is applied the size of N becomes fixed and the only applicable productions are 13, 14 which convert D's to B's. Clearly for each sentential form, the proper reduction is made by the precedence algorithm as demonstrated in the table. The only remaining question is that the analyzer may also accept strings of the form $A^I B^J C^K$ of $A^N B^{N-K} C^N B^K$. Examination of the precedence relations between A,B and C shows that C's can only appear to the right of B's and C's. Similar rules apply to both A and B so that the latter problem does not exist. By examining the sentential forms we can see that a precedence violation will occur with $ or the phrase R$ will come up with no reduction applicable in the former case.

Q. E. D.

5.3 Synonomous Grammar for LR(K) Grammars

To extend the concept of synonomous grammars to the general case of LR(K) grammars requires only a change in the definition of the set of productions used. That is, the automaton can be made

input pure and deterministic as shown in Lemma 5.3. However, since the automaton, from which the productions are found, must look ahead K symbol in order to determine if a reduction is to take place, the first K-1 symbols of the K symbol string are encoded as states. But after the reduction is performed the states which represent the encoding of the K-1 symbols have lost their value as states. That is they are not necessarily related to the positions in the automaton for which they were encoded.

This situation is easily rectified by replacing the right context on the left hand side of each production with the original terminal string. This can always be accomplished since the automaton is input pure and therefore there exists a unique deviation from the right context back to the original terminal string by the application of K trivial productions.

For example if $aSb \rightarrow asb$ is a production and $|b| = K$ then the string $B_1B_2\ldots B_{K-1}$ is a string of states and there exists trivial productions such that $B_1B_2\ldots B_{K-1} \Rightarrow B'_1B'_2\ldots B'_{K-1}$ and $B'_1, B'_2,\ldots,B'_{K-1}$ are terminal. Then the production to be used is just $aSB'_1B'_2\ldots B'_{K-1}B_K \rightarrow aSb$ and the following theorem has been proved.

Theorem 5.6

For each LR(K) grammar there exist a synonomous ESP grammar.

CHAPTER VI

Conclusions

Section 6.1 gives a summary of the results of this dissertation and section 6.2 describes some possible extensions.

6.1 Summary of Results

In chapter II we introduced the concept of precedence together with several extensions of the concept. Each of these extensions increased the class of grammars which could be effectively handled by either extending the definition of precedence or changing the grammar. However none of these methods contain any mechanism for handling the problem of equal right sides in the production set. In this dissertation we have changed the definition of precedence to include such a mechanism through the use of bounded context techniques [Fl 64].

Since the grammars which may be successfully handled by both McKeeman's [Mc 66] and Wirth and Weber's [Wi 66] extensions are subsets of LR(K) [Ku 64] grammar, any properties of ESP which are related to LR(K) are also related to these extensions.

To summarize, the following have been shown:

(1) Every deterministic language has an ESP grammar.

(2) Every LR(K) grammar has a synonomous grammar which is ESP.

(3) There exist grammars which are not LR(K) for any K but which are ESP.

(4) There exist context sensitive languages which are ESP.

Although the above results are of theoretical significance their practical value is marred by the large increase in non-terminals required for the synonomous grammar. However there do exist many grammars which are not precedence under any of the aforementioned formulations, but which can be effectively handled by ESP without changing the grammar. The use of context sensitive productions allows ESP to attack the equal right hand sides problem as part of its basic mechanism. This point is significant in that no other precedence formulation can accomplish this directly and in many such techniques it is completely ignored. Therefore we feel that ESP is an effective technique for syntactic analysis.

6.2 Some Possible Extensions of ESP

The formulation of the precedence relations under ESP can be applied to any method of precedence analysis. In particular, the methods of Floyd [Fl 63] and Colmerauer [Co 68] can be easily extended in this way.

Another interesting question involves the set of languages accepted by deterministic linear bounded automata [My 60]. Several languages from the above set have ESP grammars and we feel that the mechanism of the analysis is sufficient to handle the entire set although we have not been able to prove it.

Given a set of productions and a precedence matrix not derived from the productions a third interesting problem can be posed. Does this set define a language and if it does can a grammar be constructed from the set so that it generates this language? Again from the mechanism of the analyzer the class of languages so defined would be larger than the class of context free languages.

BIBLIOGRAPHY

BIBLIOGRAPHY

[Al 68] Alber, K., Oliva, P., Urachler, G., Concrete Syntax of PL/1, Technical Report No. TR25.084, IBM Laboratory Veinna, Veinna, Austria, June 1968.

[Ba 64] Bar-Hillel, Y., Language and Information, Addison-Wesley Publishing Company, Inc., Reading, Mass., 1964.

[Cd 67] ALGOL Generic Reference Manual, Publication No. 60214900, Documentation Department, Control Data Corporation, Palo Alto, Calif., 1967.

[Ch 64] Cheatham, T.E., and Sattley, K., "Syntax Directed Compiling", Proc. AFIPS 1964 SJCC, Vol. 25, pp. 31-57.

[Ch 68] _____, On the Generation and Implementation of Reduction Analysis Programs, CA-6804-0212, Computer Associates, Inc., Wakefield, Mass., 1968.

[Co 67] Colmerauer, A., Precedence, Analyse Syntaxeque et Languages de Programmation, Thesis, University of Grenoble, Grenoble, France, 1967.

[Ei 63] Eickel, J., Paul, M., Bauer, F.L., and Samelson, K., "A Syntax Controlled Generator of Formal Language Processes". Comm. ACM, Vol. 6, pp. 451-455, August 1963.

[Ei 64] _____, Generation of Parsing Algorithms for Chomsky 2-Type Languages. Mathematisches Institut der Techneschen Hochschule, Munich, Germany, 1964.

[Fe 68] Feldman, J. and Gries, D., "Translator Writing Systems," Comm. ACM, Vol. 11, pp. 77-113, February 1968.

[Fl 63] Floyd, R.W., "Syntactic Analysis and Operator Precedence", J. ACM, Vol. 10, pp. 316-333, July 1963.

[Fl 64] _____, "Bounded Context Syntactic Analysis", Comm. ACM, Vol. 10, pp. 62-67, February 1964.

[Gi 62] Ginsberg, S., An Introduction to Mathematical Machine Theory, Addison-Wesley Publishing Company, Inc., Reading, Mass., 1962.

[Gi 66] _____, The Mathematical Theory of Context Free Languages, McGraw-Hill, Inc., New York, 1966.

[Gr 65] Greiback, S., "A New Normal Form Theorem for Context Free Phrase Structure Grammars", J. ACM, Vol. 12, pp. 42-52, January 1965.

[Gr 68] Gries, D., "The Use of Transition Matrices in Compiling", Comm. ACM, Vol. 11, pp. 26-34, January 1968.

[Kn 65] Knuth, D.E., "On Translation of Languages from Left to Right," Information and Control, Vol. 8, pp. 607-639, October 1965.

[Ma 68] Martin, D.F., "Boolean Matrix Method for the Detection of Simple Precedence Grammars," Comm. ACM, Vol. 10, pp. 685-688, October 1968.

[Mc 66] McKeeman, W.M., An Approach to Computer Language Design, Technical Report CS48, Computer Science Department, Stanford University, Stanford, Calif., August 1966.

[Mi 68] Mills, D.L., "The Syntactic Structure of MAD/1," Technical Report 7, Concomp, University of Michigan, Ann Arbor, Mich., June 1968.

[My 60] Myhill, J., Linear Bounded Automata, Technical Report 60-165, Wright Air Development Division, Cincinnati, Ohio, June 1960.

[Ra 59] Rabin, M.O., and Scott, D., "Finite Automata and Their Decision Problems," IBM J. Research and Development, Vol. 3, pp. 114-125, 1959.

[Sq 66] Squires, E.B., Precedence Grammars and the Euler System, Parts I, II, III, and IV, Technical Report 701, Computer Science Department, University of Illinois, Urbana, Ill., August 1966.

[Ti 67] Tixier, V., Recursive Functions of Regular Expressions in Language Analysis, Technical Report CS58, Computer Science Department, Stanford University, Stanford, Calif., March 1967.

[Wa 62] Warshall, S., "A Theorem on Boolean Matrices," J. ACM, Vol. 9, pp. 11-12, January 1962.

[Wi 66] Wirth, N. and Weber, H., "Euler-A Generalization of ALGOL and its Formal Definition," Part I, Comm. ACM, Vol. 9, pp. 13-25, January 1966.

APPENDIX

APPENDIX

Analysis of MAD/I

This appendix will deal with the analysis of MAD/I under ESP. While it is not a simple precedence grammar, it is an operator precedence grammar and therefore the problem of equal right sides is assumed to be solvable by other considerations. The problem is not solvable under ESP because the grammar is ambiguous. That is, one may derive (IDR) in 3 ways from PLS and (DES) in 2 ways from PLS.

The analysis is presented in the following order.

(1) The original productions start on page 70.

(2) The precedence matrix with the simple precedence conflicts appears next, starting on page 72.

(3) The Q matrix, starting on page 76, is encoded as follows:

a) $Q_1(I,J)$ implies $Q(I,J) = E$

b) $Q_2(I,J)$ implies $Q(I,J) = G$

c) $Q_3(I,J)$ implies $Q(I,J) = L$

(4) The set of generating strings appears next, starting on page 80.

(5) Next the set of CSP's appears, starting on page 83.

(6) The bounds for the set of CSP's are listed next, starting on page 89.

(7) Finally the set of CSP's with the bounds applied is listed starting on page 91.

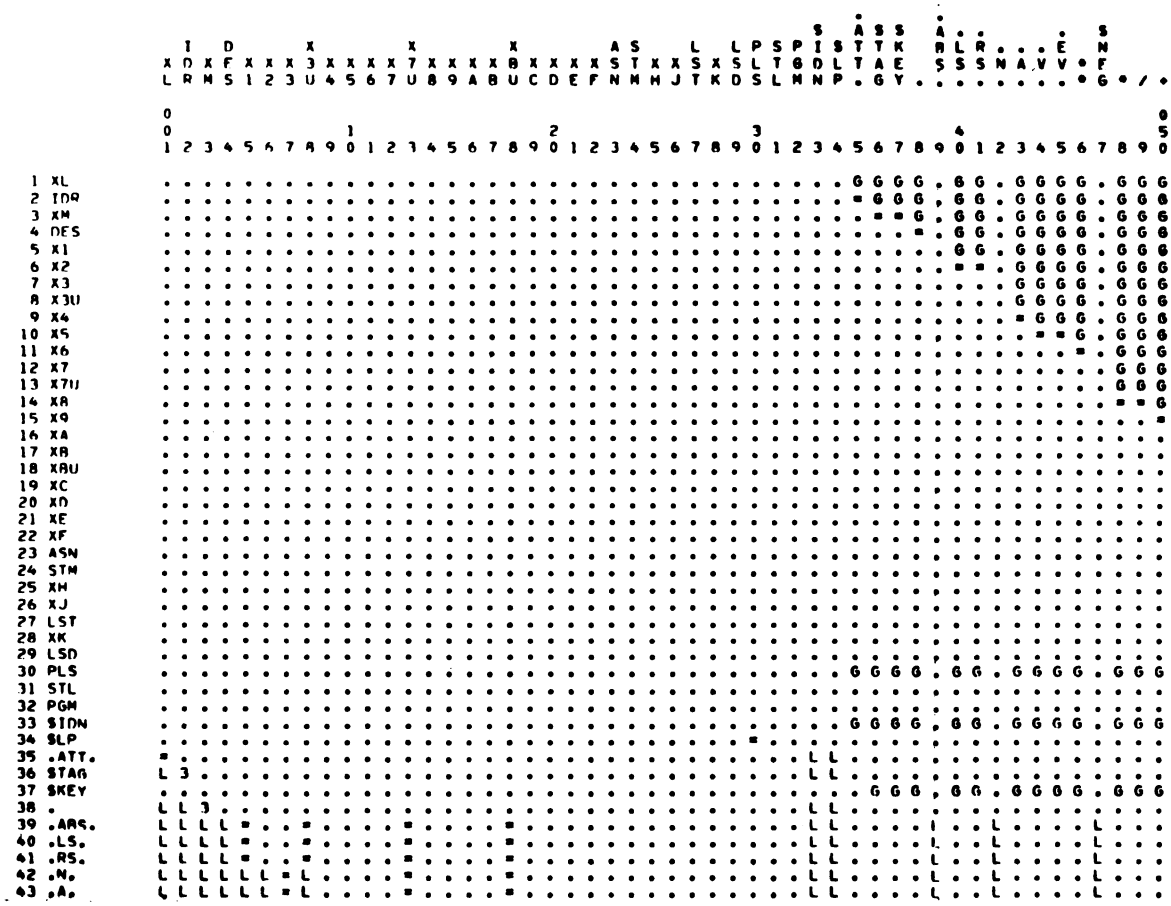
It is interesting to note that only two productions were added to the original set under ESP.

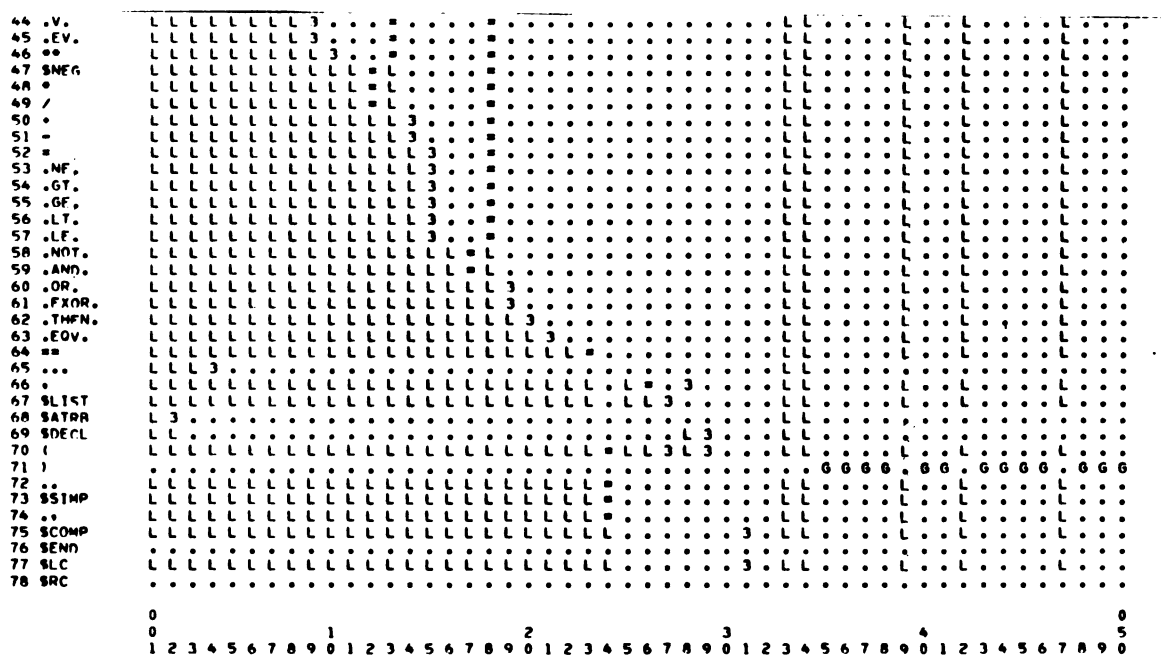
```
C      SYNTAX FOR MAD/1 OPERATOR PRECEDENCE KERNEL GRAMMAR
C    FROM MILLS, DAVID L., THE SYNTATIC STRUCTURE OF MAD/I, THE
C    UNIVERSITY OF MICHIGAN, CONCOMP PROJECT, ANN ARBOR, MICHIGAN,
C    TECHNICAL REPORT 7, JUNE 1968, 91 PP., ALSO AD-671 683
C
C          PROGRAM PRIMITIVES
   1 XL   $IDN
   2 XL   $LP   PLS
   3 IDR   XL
   4 IDR   IDR   .ATT.   XL
   5 XM   IDR
   6 XM   XM   $TAG   IDR
   7 XM   XM   $KEY
   8 DES   XM
   9 DES   DES   .   XM
C
C          ASSIGNMENT STATEMENT
  10 X1   DES
  11 X1   .ABS.   X1
  12 X2   X1
  13 X2   X2   .LS.   X1
  14 X2   X2   .RS.   X1
  15 X3   X2
  16 X3   X3U
  17 X3U   .ABS.   X3U
  18 X3   X2   .LS.   X3U
  19 X3   X2   .RS.   X3U
  20 X3U   .N.   X3
  21 X4   X3
  22 X4   X4   .A.   X3
  23 X5   X4
  24 X5   X5   .V.   X4
  25 X5   X5   .EV.   X4
  26 X6   X5
  27 X6   X6   **   X5
  28 X7   X6
  29 X7   X7U
  30 X7U   .ABS.   X7U
  31 X7   X2   .LS.   X7U
  32 X7   X2   .RS.   X7U
  33 X7U   .N.   X7U
  34 X7   X4   .A.   X7U
  35 X7   X5   .V.   X7U
  36 X7   X5   .EV.   X7U
  37 X7   X6   **   X7U
  38 X7U   $NEG   X7
  39 X8   X7
  40 X8   X8   *   X7
  41 X8   X8   /   X7
  42 X9   X8
  43 X9   X9   +   X8
  44 X9   X9   -   X8
  45 XA   X9
  46 XA   XA   =   X9
  47 XA   XA   .NE.   X9
  48 XA   XA   .GT.   X9
  49 XA   XA   .GE.   X9
  50 XA   XA   .LT.   X9
  51 XA   XA   .LE.   X9
  52 XB   XA
  53 XB   XBU
  54 XBU   .ABS.   XBU
```

```
 55 XB   X2   .LS.   XBU
 56 XB   X2   .RS.   XBU
 57 XBU   .N.   XBU
 58 XB   X4   .A.   XBU
 59 XB   X5   .V.   XBU
 60 XB   X5   .EV.   XBU
 61 XB   X6   **   XBU
 62 XBU   $NEG   XBU
 63 XB   X8   *   XBU
 64 XB   X8   /   XBU
 65 XB   X9   +   XBU
 66 XB   X9   -   XBU
 67 XB   XA   =   XBU
 68 XB   XA   .NE.   XBU
 69 XB   XA   .GT.   XBU
 70 XB   XA   .LE.   XBU
 71 XB   XA   .LT.   XBU
 72 XB   XA   .GE.   XBU
 73 XBU   .NOT.   XB
 74 XC   XB
 75 XC   XC   .AND.   XB
 76 XD   XC
 77 XD   XD   .OR.   XC
 78 XD   XD   .FXOR.   XC
 79 XE   XD
 80 XE   XE   .THEN.   XD
 81 XF   XE
 82 XF   XF   .EQV.   XE
 83 ASN   XF
 84 ASN   DES   ==   ASN
 85 STM   ASN
C
C         LIST STATEMENT
 86 XH   DES
 87 XH   XH   ...   DES
 88 XJ   XH
 89 XJ   ASN
 90 LST   XJ
 91 LST   LST   ,   XJ
 92 STM   $LIST   LST
C
C         DECLARATION STATEMENT
 93 XK   IDR
 94 XK   XK   $ATRB   IDR
 95 LSD   XK
 96 LSD   LSD   ,   XK
 97 STM   $DECL   LSD
C
C         PROGRAM STRUCTURE
 98 PLS   (   LSD   )
 99 PLS   (   LST   )
100 PLS   (   STM   )
101 STM   DES   ..   STM
102 STM   $SIMP   STM
103 STL   STM
104 STL   STL   .,   STM
105 STM   $COMP   STL   $END
106 PGM   $LC   STL   $RC
E         END OF MAD/1 SYNTAX

106 PRODUCTIONS READ

  NDT =  32   NTT =  78   NPD = 106
```

PRECEDENCE ARRAY . . . . . .

ENTRY	=	L	G	3	5	6	7
MEANS	EQ	LT	GT	LT.EQ	GT.EQ	LT.GT	LT.GT.EQ




44 .V.
45 .EV.
46 **
47 $NEG
48 *
49 /
50 +
51 -
52 =
53 .NF.
54 .GT.
55 .GE.
56 .LT.
57 .LE.
58 .NOT.
59 .AND.
60 .OR.
61 .FXOR.
62 .THEN.
63 .EQV.
64 ==
65 ...
66 ,
67 $LIST
68 $ATRR
69 $DECL
70 (
71)
72 ..
73 $STMP
74 .*
75 $COMP
76 $END
77 $LC
78 $RC

PRECEDENCE ARRAY . . . . . .

ENTRY	=	L	G	3	5	6	7
MEANS	EQ	LT	GT	LT.EQ	GT.EQ	LT.GT	LT.GT.EQ

		-	=	NE	GT	LT	NOT	AND	OR	XOR	THEN	EQV	=	.	SLIST	SLAST	SADR	(	)	SSIP	SCOMP	SNL	SCC	SRC
1	XL	G	G	G	G	G	G	G	.	G	G	G	G	G	G	G	.	G	.	.	G	G	.	G	.	G
2	IDR	G	G	G	G	G	G	G	.	G	G	G	G	G	G	G	.	G	.	.	G	G	.	G	.	G
3	XM	G	G	G	G	G	G	G	.	G	G	G	G	G	G	G	.	.	.	.	G	G	.	G	.	G
4	DES	G	G	G	G	G	G	G	.	G	G	G	G	G	=	G	G	.	.	.	.	G	=	.	G	.	G
5	X1	G	G	G	G	G	G	G	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
6	X2	G	G	G	G	G	G	G	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
7	X3	G	G	G	G	G	G	G	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
8	X3U	G	G	G	G	G	G	G	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
9	X4	G	G	G	G	G	G	G	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
10	X5	G	G	G	G	G	G	G	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
11	X6	G	G	G	G	G	G	G	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
12	X7	G	G	G	G	G	G	G	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
13	X7U	G	G	G	G	G	G	G	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
14	X8	G	G	G	G	G	G	G	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
15	X9	=	G	G	G	G	G	G	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
16	XA	.	=	=	=	=	=	=	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
17	XB	.	.	.	.	.	.	.	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
18	XBU	.	.	.	.	.	.	.	.	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
19	XC	.	.	.	.	.	.	.	=	G	G	G	G	G	.	.	G	.	.	.	.	G	.	.	G	.	G
20	XD	.	.	.	.	.	.	.	.	=	=	G	G	.	.	.	G	.	.	.	.	G	.	.	G	.	G
21	XE	.	.	.	.	.	.	.	.	.	=	G	.	.	.	.	G	.	.	.	.	G	.	.	G	.	G
22	XF	.	.	.	.	.	.	.	.	.	.	=	.	.	.	.	G	.	.	.	.	G	.	.	G	.	G
23	ASN	.	.	.	.	.	.	.	.	.	.	.	.	.	.	G	.	.	.	.	.	G	.	.	G	.	G
24	STM	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	5	.	.	G	.	G
25	XM	.	.	.	.	.	.	.	.	.	.	.	.	.	=	G	.	.	.	.	.	G	.	.	G	.	G
26	XJ	.	.	.	.	.	.	.	.	.	.	.	.	.	.	G	.	.	.	.	.	G	.	.	G	.	G
27	LST	.	.	.	.	.	.	.	.	.	.	.	.	.	.	=	.	.	.	.	.	5	.	.	G	.	G
28	XK	.	.	.	.	.	.	.	.	.	.	.	.	.	.	G	.	=	.	.	.	G	.	.	G	.	G
29	LSD	.	.	.	.	.	.	.	.	.	.	.	.	.	.	=	.	.	.	.	.	5	.	.	G	.	G
30	PLS	G	G	G	G	G	G	G	.	G	G	G	G	G	G	G	.	G	.	.	G	G	.	G	.	G
31	STL	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	=	.	=	.
32	PGM	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
33	SIDN	G	G	G	G	G	G	G	.	G	G	G	G	G	G	G	.	G	.	.	G	G	.	G	.	G
34	SLP	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	L	.	.	.	.	.	.
35	.ATT.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
36	STAG	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
37	SKEY	G	G	G	G	G	G	G	.	G	G	G	G	G	G	G	.	.	.	G	G	.	G	.	G	.
38	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
39	.ABS.	.	.	.	.	.	.	.	L	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
40	.LS.	.	.	.	.	.	.	.	L	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
41	.RS.	.	.	.	.	.	.	.	L	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
42	.N.	.	.	.	.	.	.	.	L	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
43	.A.	.	.	.	.	.	.	.	L	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

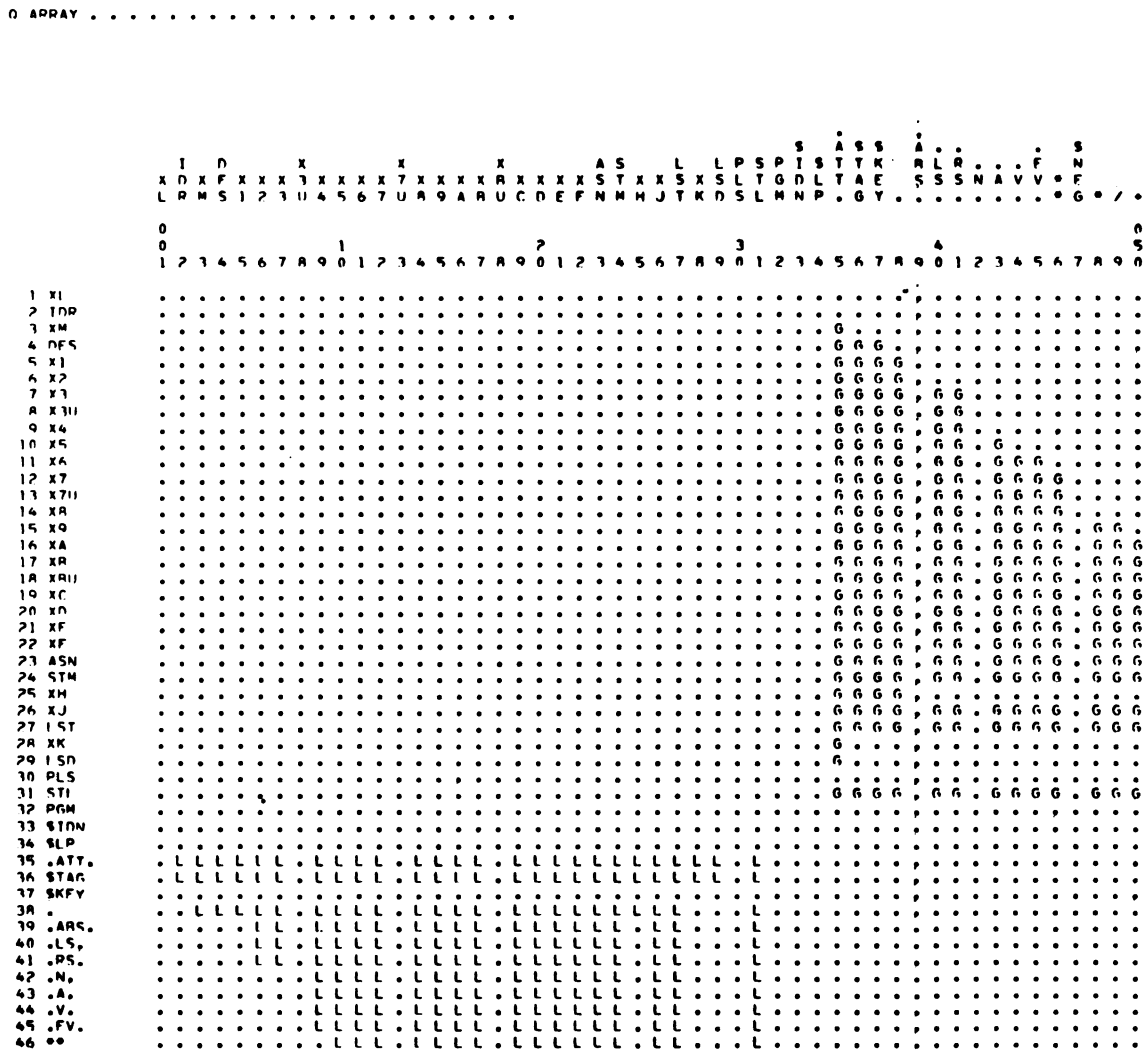
```
44 .V.          . . . . . L . . . . . . . . . . . . . . . . . . . . .
45 .EV.         . . . . . L . . . . . . . . . . . . . . . . . . . . .
46 **           . . . . . L . . . . . . . . . . . . . . . . . . . . .
47 $NEG         . . . . . L . . . . . . . . . . . . . . . . . . . . .
48 *            . . . . . L . . . . . . . . . . . . . . . . . . . . .
49 /            . . . . . L . . . . . . . . . . . . . . . . . . . . .
50 +            . . . . . L . . . . . . . . . . . . . . . . . . . . .
51 -            . . . . . L . . . . . . . . . . . . . . . . . . . . .
52 =            . . . . . L . . . . . . . . . . . . . . . . . . . . .
53 .NE.         . . . . . L . . . . . . . . . . . . . . . . . . . . .
54 .GT.         . . . . . L . . . . . . . . . . . . . . . . . . . . .
55 .GE.         . . . . . L . . . . . . . . . . . . . . . . . . . . .
56 .LT.         . . . . . L . . . . . . . . . . . . . . . . . . . . .
57 .LE.         . . . . . L . . . . . . . . . . . . . . . . . . . . .
58 .NOT.        . . . . . L . . . . . . . . . . . . . . . . . . . . .
59 .AND.        . . . . . L . . . . . . . . . . . . . . . . . . . . .
60 .OR.         . . . . . L . . . . . . . . . . . . . . . . . . . . .
61 .EXOR.       . . . . . L . . . . . . . . . . . . . . . . . . . . .
62 .THEN.       . . . . . L . . . . . . . . . . . . . . . . . . . . .
63 .EQV.        . . . . . L . . . . . . . . . . . . . . . . . . . . .
64 ==           . . . . . L . . . . . . . . . . . . . . . . . . . . .
65 ...          . . . . . . . . . . . . . . . . . . . . . . . . . . .
66 .            . . . . . L . . . . . . . . . . . . . . . . . . . . .
67 $LIST        . . . . . L . . . . . . . . . . . . . . . . . . . . .
68 $ATRR        . . . . . . . . . . . . . . . . . . . . . . . . . . .
69 $DFCL        . . . . . . . . . . . . . . . . . . . . . . . . . . .
70 (            . . . . . L . . . . . . . . . L . L . . . L . L . . .
71 )            G G G G G G G . G G G G G G G . G . . G G . G . G . G
72 ..           . . . . . L . . . . . . . . . L . L . . . L . L . . .
73 $STMP        . . . . . L . . . . . . . . . L . L . . . L . L . . .
74 ..           . . . . . L . . . . . . . . . L . L . . . L . L . . .
75 $COMP        . . . . . L . . . . . . . . . L . L . . . L . L . . .
76 $END         . . . . . . . . . . . . . . . . . . G . . G . G . G
77 $LC          . . . . . L . . . . . . . . . L . L . . . L . L . . .
78 $RC          . . . . . . . . . . . . . . . . . . . . . . . . . . .

                0                                                   1
                5                   6               7               0
                1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
```

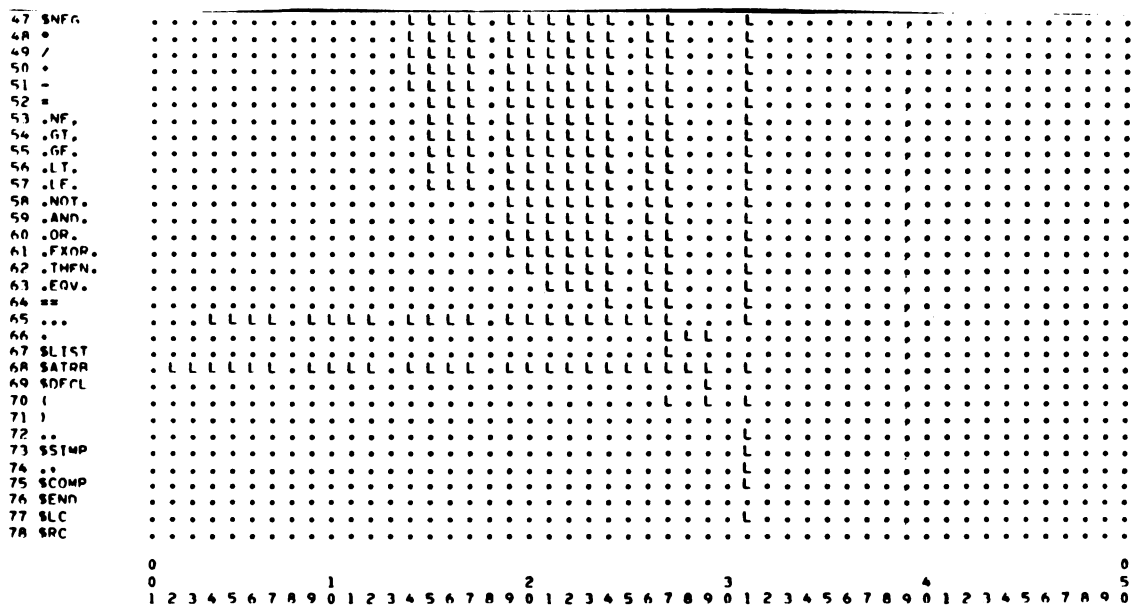
NUMBER OF ENTRIES OF EACH TYPE IN PRECEDENCE ARRAY

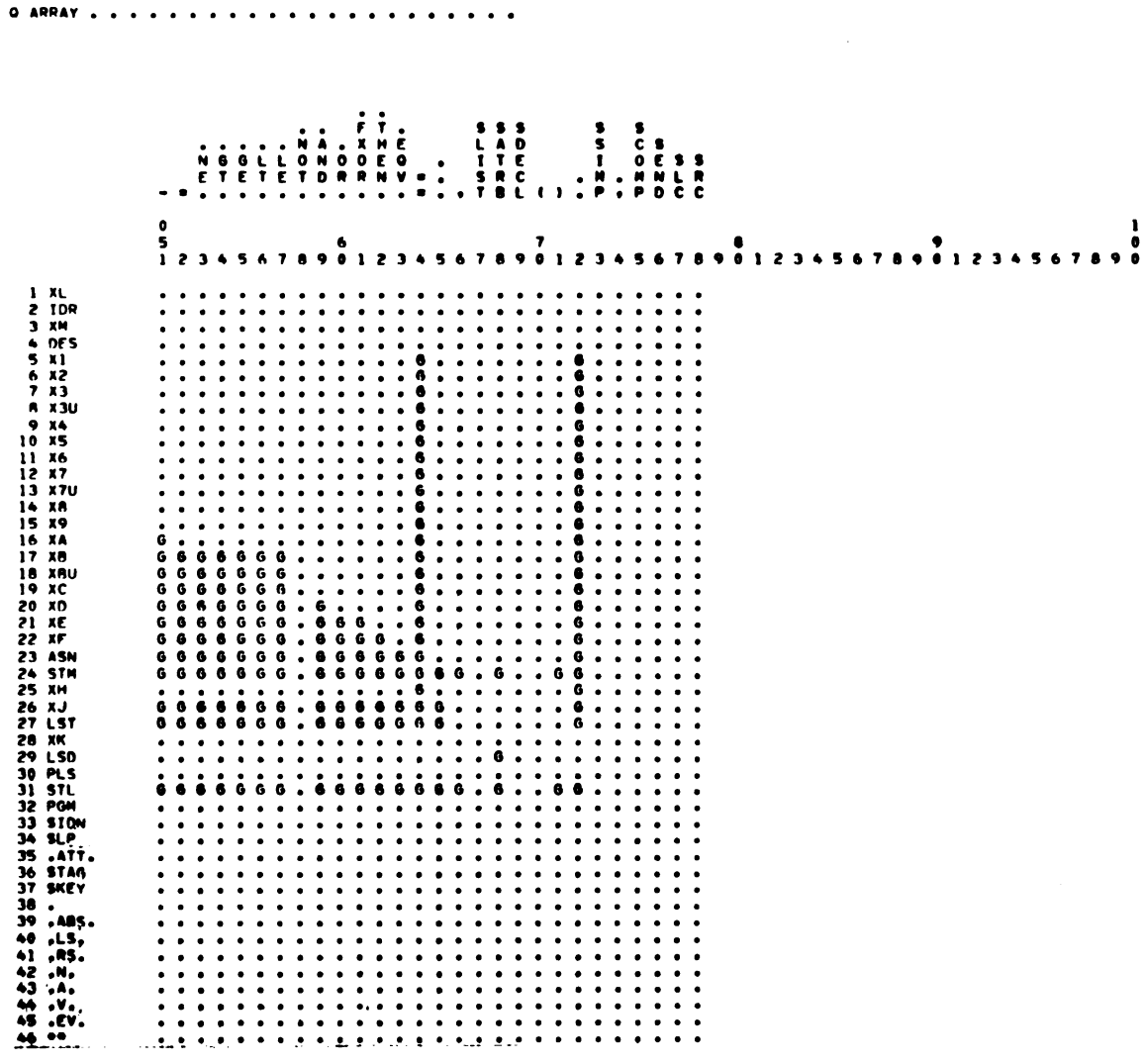
.....EQ	.....LT	.....GT
82	771	584

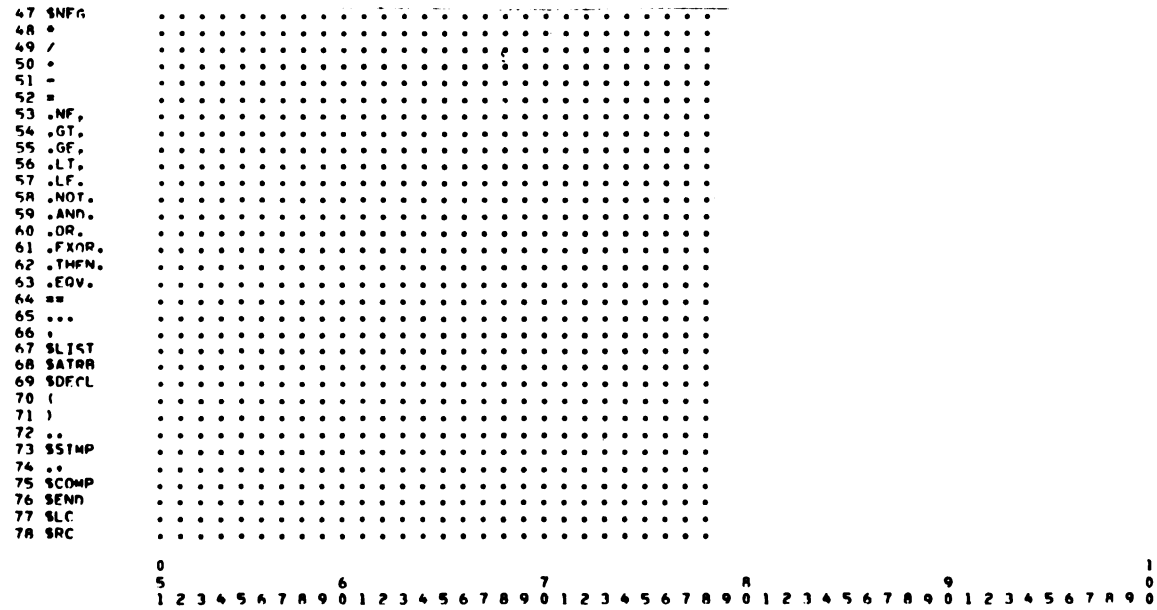
...LT.EQ	...GT.EQ	...LT.GT	LT.GT.EQ
26	3	0	0

TOTAL NUMBER OF ENTRIES = 6084
NUMBER OF BLANK ENTRIES = 4618
RATIO OF NON-BLANK TO TOTAL ENTRIES = 24.10 PCT


0 ARRAY
1 XL
2 IDR
3 XM
4 DFS
5 X1
6 X2
7 X3
8 X3U
9 X4
10 X5
11 X6
12 X7
13 X7U
14 XA
15 X9
16 XA
17 XB
18 XBU
19 XC
20 XD
21 XE
22 XF
23 ASN
24 STM
25 XH
26 XJ
27 LST
28 XK
29 LSD
30 PLS
31 STI
32 PGM
33 $TON
34 $LP
35 .ATT.
36 $TAG
37 $KEY
38 .
39 .ARS.
40 .LS.
41 .RS.
42 .N.
43 .A.
44 .V.
45 .FV.
46 **


47 $NEG
48 *
49 /
50 +
51 -
52 =
53 .NE.
54 .GT.
55 .GE.
56 .LT.
57 .LE.
58 .NOT.
59 .AND.
60 .OR.
61 .EXOR.
62 .IMPN.
63 .EQV.
64 **
65 ...
66 .
67 $LIST
68 $ATRR
69 $DECL
70 (
71)
72 ..
73 $STMP
74 ..
75 $COMP
76 $END
77 $LC
78 $RC
0 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 0 5 0


0 ARRAY
1 XL
2 IDR
3 XM
4 DFS
5 X1
6 X2
7 X3
8 X3U
9 X4
10 X5
11 X6
12 X7
13 X7U
14 XA
15 X9
16 XA
17 XB
18 XBU
19 XC
20 XD
21 XE
22 XF
23 ASN
24 STM
25 XM
26 XJ
27 LST
28 XK
29 LSD
30 PLS
31 STL
32 PGM
33 SIQN
34 SLP
35 .ATT.
36 STAQ
37 SKEY
38 .
39 .ABS.
40 ,LS,
41 ,RS.
42 .N.
43 .A.
44 .V.
45 .EV.
46 **



TOTAL TIME REQUIRED FOR ANALYSIS = 9.775 SECONDS

```
GENERATOR STRINGS
PGM
$LC     STL
STM
ASN
$LIST   LST
$DECL   LSD
DES
$COMP   STL
XF
XJ
XK
$DECL   LSD     ,       XK
XM
DES     .       XM
XE
XF      .EQV.   XE
XH
IDR
XK      $ATRB   IDR
$DECL   LSD     ,       XK      $ATRB   IDR
XM      $TAG    IDR
DES     .       XM      $TAG    IDR
XD
XE      .THEN.  XD
XF      .EQV.   XE      .THEN.  XD
XH      ...     DES
XL
XC
XD      .OR.    XC
XD      .FXOR.  XC
XE      .THEN.  XD      .OR.    XC
```

```
XE      .THEN.  XD      .FXOR.  XC
XF      .EQV.   XE      .THEN.  XD      .OR.    XC
XF      .EQV.   XE      .THEN.  XD      .FXOR.  XC
XH      ...     DES     .       XM
PLS
XB
XH      ...     DES     .       XM      STAG    IDR
(       LSD
(       LST
STM     )
XA
XBU
X2
X4
X5
X6
X8
X9
(       LSD     ,       XK
XA      =       X9
XA      .NE.    X9
XA      .GT.    X9
XA      .GE.    X9
XA      .LT.    X9
XA      .LE.    X9
X1
X3
X5      .V.     X4
X5      .EV.    X4
X6      **      X5
X7
X9      +       X8
X9      -       X8
```

(	LSD	,	XK	$ATRB	IDR
XA	=	X9	+	X8	
XA	=	X9	-	X8	
XA	.NE.	X9	+	X8	
XA	.NE.	X9	-	X8	
XA	.GT.	X9	+	X8	
XA	.GT.	X9	-	X8	
XA	.GE.	X9	+	X8	
XA	.GE.	X9	-	X8	
XA	.LT.	X9	+	X8	
XA	.LT.	X9	-	X8	
XA	.LE.	X9	+	X8	
XA	.LE.	X9	-	X8	
X3U					
X6	**	X5	.V.	X4	
X6	**	X5	.EV.	X4	
X7U					

```
PRODUCTIONS LEFT TO RIGHT
$LC     STL     $RC      <<===== PGM
$LC     STL     .,       STM      <<===== $LC     STL
$COMP   STL     $END     <<===== STM
$COMP   STL     .,       STM      <<===== $COMP   STL
$SIMP   STM     )        <<===== STM     )
$SIMP   STM      <<===== STM
(       LSD     )        <<===== PLS
(       LSD     ,        XK       $ATRB   IDR     .ATT.   XL       <<===== (       LSD     ,       XK      $ATRB   IDR
(       LSD     ,        XK       $ATRB   IDR      <<===== (       LSD     ,       XK
(       LSD     ,        XK       <<===== (       LSD
(       LST     )        <<===== PLS
(       LST     ,        XJ       <<===== (       LST
(       STM     )        <<===== PLS
$DECL   LSD     )        <<===== STM     )
$DECL   LSD     ,        XK       $ATRB   IDR     .ATT.   XL       <<===== $DECL   LSD     ,       XK      $ATRB   IDR
$DECL   LSD     ,        XK       $ATRB   IDR      <<===== $DECL   LSD     ,       XK
$DECL   LSD     ,        XK       <<===== $DECL   LSD
$DECL   LSD      <<===== STM
$LIST   LST     )        <<===== STM     )
$LIST   LST     ,        XJ       <<===== $LIST   LST
$LIST   LST      <<===== STM
.NOT.   XB       <<===== XBU
$NEG    XBU      <<===== XBU
$NEG    X7       <<===== X7U
.N.     XBU      <<===== XBU
.N.     X7U      <<===== X7U
.N.     X3       <<===== X3U
.ABS.   XBU      <<===== XBU
.ABS.   X7U      <<===== X7U
.ABS.   X3U      <<===== X3U
```

```
.ARS.   X1      <<==== X1
$LP     PLS     <<==== XL
$IDN     <<==== XL
XK      $ATRB   IDR     .ATT.   XL      <<==== XK       $ATRB   IDR
XK      $ATRB   IDR      <<==== XK
XK       <<==== LSD
XJ       <<==== LST
XH      ...     DES     .       XM      $KEY     <<==== XH      ...     DES     .       XM
XH      ...     DES     .       XM      STAG    IDR     .ATT.   XL       <<==== XH      ...     DES     .       XM
$TAG    IDR
XH      ...     DES     .       XM      STAG    IDR      <<==== XH      ...     DES     .       XM
XH      ...     DES     .       XM       <<==== XH      ...     DES
XH      ...     DES      <<==== XH
XH       <<==== XJ
STM      <<==== STL
ASN      <<==== STM
ASN      <<==== XJ
XF      .EQV.   XE      .THEN.  XD      .FXOR.  XC      .AND.   XB       <<==== XF      .EQV.   XE      .THEN.  XD
.FXOR.  XC
XF      .EQV.   XF      .THEN.  XD      .FXOR.  XC       <<==== XF      .EQV.   XE      .THEN.  XD
XF      .EQV.   XE      .THEN.  XD      .OR.    XC      .AND.   XB       <<==== XF      .EQV.   XE      .THEN.  XD
.OR.    XC
XF      .EQV.   XE      .THEN.  XD      .OR.    XC       <<==== XF      .EQV.   XE      .THEN.  XD
XF      .EQV.   XF      .THEN.  XD       <<==== XF      .EQV.   XF
XF      .EQV.   XE       <<==== XF
XF       <<==== ASN
XE      .THEN.  XD      .FXOR.  XC      .AND.   XB       <<==== XE      .THEN.  XD      .FXOR.  XC
XE      .THEN.  XD      .FXOR.  XC       <<==== XE      .THEN.  XD
XE      .THEN.  XD      .OR.    XC      .AND.   XB       <<==== XE      .THEN.  XD      .OR.    XC
XE      .THEN.  XD      .OR.    XC       <<==== XE      .THEN.  XD
XE      .THEN.  XD       <<==== XE
XE       <<==== XF
XD      .FXOR.  XC      .AND.   XB       <<==== XD      .FXOR.  XC
XD      .FXOR.  XC       <<==== XD
```

```
XD      .OR.    XC      .AND.   XB      <<==== XD      .OR.    XC
XD      .OR.    XC       <<==== XD
XD       <<==== XE
XC      .AND.   XB       <<==== XC
XC       <<==== XD
XBU      <<==== XB
XB       <<==== XC
XA      .LF.    XBU      <<==== XB
XA      .LE.    X9      -       X8      /       X7       <<==== XA      .LF.    X9      -       X8
XA      .LF.    X9      -       X8      *       X7       <<==== XA      .LF.    X9      -       X8
XA      .LF.    X9      -       X8       <<==== XA      .LE.    X9
XA      .LE.    X9      *       X8      /       X7       <<==== XA      .LE.    X9      *       X8
XA      .LF.    X9      *       X8      *       X7       <<==== XA      .LE.    X9      *       X8
XA      .LF.    X9      *       X8       <<==== XA      .LE.    X9
XA      .LE.    X9       <<==== XA
XA      .LT.    XBU      <<==== XB
XA      .LT.    X9      -       X8      /       X7       <<==== XA      .LT.    X9      -       X8
XA      .LT.    X9      -       X8      *       X7       <<==== XA      .LT.    X9      -       X8
XA      .LT.    X9      -       X8       <<==== XA      .LT.    X9
XA      .LT.    X9      *       X8      /       X7       <<==== XA      .LT.    X9      *       X8
XA      .LT.    X9      *       X8      *       X7       <<==== XA      .LT.    X9      *       X8
XA      .LT.    X9      *       X8       <<==== XA      .LT.    X9
XA      .LT.    X9       <<==== XA
XA      .GE.    XBU      <<==== XB
XA      .GF.    X9      -       X8      /       X7       <<==== XA      .GF.    X9      -       X8
XA      .GF.    X9      -       X8      *       X7       <<==== XA      .GF.    X9      -       X8
XA      .GE.    X9      -       X8       <<==== XA      .GE.    X9
XA      .GF.    X9      *       X8      /       X7       <<==== XA      .GF.    X9      *       X8
XA      .GE.    X9      *       X8      *       X7       <<==== XA      .GF.    X9      *       X8
XA      .GE.    X9      *       X8       <<==== XA      .GE.    X9
XA      .GE.    X9       <<==== XA
XA      .GT.    XBU      <<==== XB
XA      .GT.    X9      -       X8      /       X7       <<==== XA      .GT.    X9      -       X8
```

```
XA      .GT.    X9      -       X8      *       X7      <<==== XA      .GT.    X9      -       X8
XA      .GT.    X9      -       X8       <<==== XA      .GT.    X9
XA      .GT.    X9      +       X8      /       X7      <<==== XA      .GT.    X9      +       X8
XA      .GT.    X9      +       X8      *       X7      <<==== XA      .GT.    X9      +       X8
XA      .GT.    X9      +       X8       <<==== XA      .GT.    X9
XA      .GT.    X9       <<==== XA
XA      .NE.    XBU      <<==== XB
XA      .NE.    X9      -       X8      /       X7      <<==== XA      .NE.    X9      -       X8
XA      .NE.    X9      -       X8      *       X7      <<==== XA      .NE.    X9      -       X8
XA      .NE.    X9      -       X8       <<==== XA      .NE.    X9
XA      .NE.    X9      +       X8      /       X7      <<==== XA      .NE.    X9      +       X8
XA      .NE.    X9      +       X8      *       X7      <<==== XA      .NE.    X9      +       X8
XA      .NE.    X9      +       X8       <<==== XA      .NE.    X9
XA      .NE.    X9       <<==== XA
XA      =       XBU      <<==== XB
XA      =       X9      -       X8      /       X7      <<==== XA      =       X9      -       X8
XA      =       X9      -       X8      *       X7      <<==== XA      =       X9      -       X8
XA      =       X9      -       X8       <<==== XA      =       X9
XA      =       X9      +       X8      /       X7      <<==== XA      =       X9      +       X8
XA      =       X9      +       X8      *       X7      <<==== XA      =       X9      +       X8
XA      =       X9      +       X8       <<==== XA      =       X9
XA      =       X9       <<==== XA
XA       <<==== XB
X9      -       XBU      <<==== XB
X9      -       X8      /       X7       <<==== X9      -       X8
X9      -       X8      *       X7       <<==== X9      -       X8
X9      -       X8       <<==== X9
X9      +       XBU      <<==== XB
X9      +       X8      /       X7       <<==== X9      +       X8
X9      +       X8      *       X7       <<==== X9      +       X8
X9      +       X8       <<==== X9
X9       <<==== XA
XB      /       XBU      <<==== XB
```

```
X8      /       X7      <<==== X8
X8      *       X8U     <<==== X8
X8      *       X7      <<==== X8
X8       <<==== X9
X7U      <<==== X7
X7       <<==== X8
X6      **      X8U     <<==== X8
X6      **      X7U     <<==== X7
X6      **      X5      .FV.    X4      .A.     X3      <<==== X6      **      X5      .EV.    X4
X6      **      X5      .FV.    X4      <<==== X6      **      X5
X6      **      X5      .V.     X4      .A.     X3      <<==== X6      **      X5      .V.     X4
X6      **      X5      .V.     X4      <<==== X6      **      X5
X6      **      X5      <<==== X6
X6       <<==== X7
X5      .FV.    X8U     <<==== X8
X5      .FV.    X7U     <<==== X7
X5      .FV.    X4      .A.     X3      <<==== X5      .EV.    X4
X5      .EV.    X4      <<==== X5
X5      .V.     X8U     <<==== X8
X5      .V.     X7U     <<==== X7
X5      .V.     X4      .A.     X3      <<==== X5      .V.     X4
X5      .V.     X4      <<==== X5
X5       <<==== X6
X4      .A.     X8U     <<==== X8
X4      .A.     X7U     <<==== X7
X4      .A.     X3      <<==== X4
X4       <<==== X5
X3U      <<==== X3
X3       <<==== X4
X2      .RS.    X8U     <<==== X8
X2      .RS.    X7U     <<==== X7
X2      .RS.    X3U     <<==== X3
X2      .RS.    X1      <<==== X2
```

```
X2      .LS.    XBU      <<==== XB
X2      .LS.    X7U      <<==== X7
X2      .LS.    X3U      <<==== X3
X2      .LS.    X1       <<==== X2
X2       <<==== X3
X1       <<==== X2
DES     ..      STM     )        <<==== STM     )
DES     ..      STM      <<==== STM
DES     ==      ASN      <<==== ASN
DES     .       XM      $KEY     <<==== DES     .       XM
DES     .       XM      $TAG    IDR     .ATT.   XL       <<==== DES     .       XM      $TAG    IDR
DES     .       XM      $TAG    IDR      <<==== DES     .       XM
DES     .       XM       <<==== DES
DES      <<==== XH
DES      <<==== X1
XM      $KEY     <<==== XM
XM      $TAG    IDR     .ATT.   XL       <<==== XM      $TAG    IDR
XM      $TAG    IDR      <<==== XM
XM       <<==== DES
IDR     .ATT.   XL       <<==== IDR
IDR      <<==== XK
IDR      <<==== XM
XL       <<==== IDR
```

BOUNDS ON THE VARIABLES

		LL	LR	RR	RL
1	XI	1	1	3	4
2	IOR	3	3	3	4
3	XM	4	3	3	4
4	DFS	4	4	3	3
5	X1	1	1	3	3
6	X2	3	3	1	1
7	X3	1	1	3	4
8	X3U	1	1	3	3
9	X4	3	3	3	4
10	X5	4	3	3	3
11	X6	4	3	1	1
12	X7	1	1	3	4
13	X7U	1	1	3	3
14	X8	3	3	3	4
15	X9	4	3	3	3
16	XA	4	3	1	1
17	XB	1	1	3	4
18	XBU	1	1	3	3
19	XC	3	3	3	4
20	XD	4	3	3	4
21	XE	4	3	3	3
22	XF	4	3	1	1
23	ASN	1	1	3	3
24	STM	1	1	3	4
25	XH	4	3	1	1
26	XJ	1	1	3	4
27	LST	0	0	2	2
28	XK	4	3	3	4
29	LSD	0	0	2	2
30	PLS	0	0	2	2
31	STL	0	0	0	0
32	PGM	0	0	0	0
33	$TDN	1	1	1	1
34	$IP	2	2	0	0
35	.ATT.	0	0	0	0
36	$TAG	0	0	0	0
37	$KEY	0	0	2	3
38	.	0	0	0	0
39	.ABS.	2	2	0	0
40	.IS.	0	0	0	0
41	.RS.	0	0	0	0
42	.N.	2	2	0	0
43	.A.	0	0	0	0
44	.V.	0	0	0	0
45	.FV.	0	0	0	0
46	**	0	0	0	0
47	$NEG	2	2	0	0
48	*	0	0	0	0
49	/	0	0	0	0
50	+	0	0	0	0
51	-	0	0	0	0
52	=	0	0	0	0
53	.NE.	0	0	0	0
54	.GT.	0	0	0	0
55	.GE.	0	0	0	0
56	.LT.	0	0	0	0
57	.LE.	0	0	0	0
58	.NOT.	2	2	0	0
59	.AND.	0	0	0	0
60	.OR.	0	0	0	0
61	.EXOR.	0	0	0	0

62	.THEN.	0	0	0	0
63	.EQV.	0	0	0	0
64	==	0	0	0	0
65	...	0	0	0	0
66	.	0	0	0	0
67	$LIST	4	3	0	0
68	$ATRB	0	0	0	0
69	$DECL	4	3	0	0
70	(	4	3	0	0
71	)	0	0	4	3
72	..	0	0	0	0
73	$SIMP	2	3	0	0
74	..	0	0	0	0
75	$COMP	4	3	0	0
76	$END	0	0	3	3
77	$LC	4	3	0	0
78	$RC	0	0	3	3

```
PRODUCTIONS WITH BOUNDS APPLIED
$LC      STL      $RC       <<==== PGM
$COMP    STL      $END      <<==== STM
$SIMP    STM       <<==== STM
(        LSD      )         <<==== PLS
(        LST      )         <<==== PLS
(        STM      )         <<==== PLS
$DECL    LSD      )         <<==== STM      )
$DECL    LSD       <<==== STM
$LIST    LST      )         <<==== STM      )
$LIST    LST       <<==== STM
.NOT.    XB        <<==== XBU
$NEG     XBU       <<==== XBU
$NEG     X7        <<==== X7U
.N.      XBU       <<==== XBU
.N.      X7U       <<==== X7U
.N.      X3        <<==== X3U
.ABS.    XBU       <<==== XBU
.ABS.    X7U       <<==== X7U
.ABS.    X3U       <<==== X3U
.ABS.    X1        <<==== X1
$LP      PLS       <<==== XL
$IDN      <<==== XL
STL      ..        STM       <<==== STL
LSD      .         XK        <<==== LSD
XK       $ATRB     IDR       <<==== XK
XK        <<==== LSD
LST      .         XJ        <<==== LST
XJ        <<==== LST
XH       ...       DES       <<==== XH
XH        <<==== XJ
STM       <<==== STL
```

```
ASN      <<==== STM
ASN      <<==== XJ
XF       .EQV.  XE       <<==== XF
XF       <<==== ASN
XE       .THEN. XD       <<==== XE
XE       <<==== XF
XD       .FXOR. XC       <<==== XD
XD       .OR.   XC       <<==== XD
XD       <<==== XE
XC       .AND.  XB       <<==== XC
XC       <<==== XD
XBU      <<==== XB
XB       <<==== XC
XA       .LE.   XBU      <<==== XB
XA       .LE.   X9       <<==== XA
XA       .LT.   XBU      <<==== XB
XA       .LT.   X9       <<==== XA
XA       .GE.   XBU      <<==== XB
XA       .GE.   X9       <<==== XA
XA       .GT.   XBU      <<==== XB
XA       .GT.   X9       <<==== XA
XA       .NE.   XBU      <<==== XB
XA       .NE.   X9       <<==== XA
XA       =      XBU      <<==== XB
XA       =      X9       <<==== XA
XA       <<==== XB
X9       -      XBU      <<==== XB
X9       -      X8       <<==== X9
X9       +      XBU      <<==== XB
X9       +      X8       <<==== X9
X9       <<==== XA
X8       /      XBU      <<==== X8
X8       /      X7       <<==== X8
```

```
X8      *       X8U     <<==== X8
X8      *       X7      <<==== X8
X8       <<==== X9
X7U      <<==== X7
X7       <<==== X8
X6      **      X8U     <<==== X8
X6      **      X7U     <<==== X7
X6      **      X5      <<==== X6
X6       <<==== X7
X5      .EV.    X8U     <<==== X8
X5      .EV.    X7U     <<==== X7
X5      .EV.    X4      <<==== X5
X5      .V.     X8U     <<==== X8
X5      .V.     X7U     <<==== X7
X5      .V.     X4      <<==== X5
X5       <<==== X6
X4      .A.     X8U     <<==== X8
X4      .A.     X7U     <<==== X7
X4      .A.     X3      <<==== X4
X4       <<==== X5
X3U      <<==== X3
X3       <<==== X4
X2      .RS.    X8U     <<==== X8
X2      .RS.    X7U     <<==== X7
X2      .RS.    X3U     <<==== X3
X2      .RS.    X1      <<==== X2
X2      .LS.    X8U     <<==== X8
X2      .LS.    X7U     <<==== X7
X2      .LS.    X3U     <<==== X3
X2      .LS.    X1      <<==== X2
X2       <<==== X3
X1       <<==== X2
DFS     ..      STM     <<==== STM
```

```
DES      ==       ASN       <<==== ASN
DES      .        XM        <<==== DES
DES       <<==== XH
DES       <<==== X1
XM       $KEY      <<==== XM
XM       $TAG     IDR       <<==== XM
XM        <<==== DES
IDR      .ATT.    XL        <<==== IDR
IDR       <<==== XK
IDR       <<==== XM
XL        <<==== IDR
```