Topics on formal grammars.

Carolina Pacienza Salas
TOPICS ON FORMAL GRAMMARS

by

Carolina Paciencia Salas

A Thesis
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the degree of
Master of Science
in
Computer Science

Lehigh University
1978
This thesis is accepted and approved in partial fulfillment of the requirement for the degree of Master of Science.

Sept. 11, 1978
Date

Professor Gerhard Rayna
Professor in Charge

Professor Gilbert A. Steggle
Chairman, Department of Mathematics
Acknowledgement

I wish to express my sincerest gratitude and appreciation to Professor Gerhard Rayna for his patience and understanding in guiding me through this thesis, to Professor Everett Pitcher for giving me the opportunity to continue my studies at Lehigh University, to Professor Samuel Gulden for the many things he taught me, to Father Bienvenido Nebres, S.J. for his support and encouragement, to Doctor and Mrs. Gaylord Ojers for their kindness and concern, to Mr. and Mrs. Alfred Taglang for their loving care, to my sisters, Consuelo and Carina, my grandmother, Mrs. Josefa Paciencia, and my parents, Mr. and Mrs. Conrado Salas for their love and prayers, for their trust and confidence in me and for patiently awaiting my return.
Table of Contents

Abstract 1

1. Introduction 3

2. Formal Grammars and Formal Languages 5
   2.1. Introduction 5
   2.2. Fundamental Concepts 6
   2.3. Engeler's Definition of Formal Grammar 9
   2.4. Conventional Definition of Formal Grammar 10
   2.5. Non-Ambiguity or Unique Readability 13

3. Partial Recursive Functions and Machines 28
   3.1. Some Basic Definitions 28
   3.2. Partial Recursive Functions 31
   3.3. Shepherdson-Strugis or SS Machine 33
   3.4. Tag Machine 37
   3.5. Turing Machine 42

4. Regular Expressions 47
   4.1. Regular Expressions and Regular Sets 47
4.2. Finite State Machines and Finite Deterministic Automata 49

4.3. Regular Language 52

5. Simple Precedence Grammars 60

5.1. Precedence Grammars 60

5.2. Computation of the Precedence Relations 70

Bibliography 84

Vita 85
Abstract

Engeler (see Bibliography) defines formal grammars and formal languages as relational structures whereas the conventional definition presents them as ordered quadruples. It is shown how one can be derived from the other and how both approaches handle unambiguity or unique readability of a context-free language.

It is shown that a machine that can effectively simulate partial recursive functions may also perform the numerical operations addition, multiplication, exponentiation, magnitude, minimum, maximum, subtraction, division and modular division. It is also shown how the SS Machine, Tag Machine and Turing Machine can effectively simulate partial recursive functions.

It is shown that regular set, regular language and
finite deterministic automaton are in fact equivalent.

It is illustrated how the precedence relations \(<\), \(\leq\) and \(\geq\) may be computed. It is shown that a precedence grammar is unambiguous and that parsing by locating the handle is an efficient way of determining the validity of a word.
Chapter 1

Introduction

A colleague of mine once told me he found out that the prevailing concept among many applied scientists, such as chemists, physicists, engineers, and the like, is that computer science is equated to programming in FORTRAN and programming in FORTRAN is computer science. Among businessmen, it is slightly different. Computer science is programming in COBOL and programming in COBOL is computer science.

Is it any wonder then that students who are not computer science majors think the same way since, very often, as undergraduates they are usually first exposed to these people and precisely to programming in FORTRAN by virtue of the science courses they have to take?
If one finds some satisfaction because most of his trials are accompanied by what one may call a "success" because the programs "run", the curiosity of the individual may be aroused. Hopefully, this leads to questions arising in one's mind, such as: "What makes the computer work the way it does?" or "How does the computer know what to do?" or "How can the computer understand English?" or "How does one know that a particular instruction will be understood by the computer?" and many others. Hopefully too, these questions lead the individual to further investigate and even go into computer science. Once in it, one discovers what really comprises computer science and what fallacies he had been exposed to. One learns about programming techniques, hardware, file structures, data structures, processing, compiler design, computer languages, coding, information system, formal languages, and many others. All these are undoubtedly interesting, but one area which hardly fails to find a student in awe is formal languages. As a classmate once exclaimed: "I never imagined languages had such extensive mathematical basis!" Indeed it has, and it was precisely this which led me to do my master's thesis on this subject. I wanted to learn more about formal languages.
Chapter 2
Formal Grammars and Formal Languages

2.1. Introduction

We realize that it is practically an impossible task to come up with a fixed system of rules by which to determine all the possible meanings of a given statement. Although writers are careful to make their statements unambiguous, very often they do so without much success. Under such circumstances, they usually appeal to context, common sense and good will. Formal grammars are designed as an attempt to overcome these shortcomings of colloquial languages.
2.2. Fundamental Concepts

We recall the following definitions and concepts.

Definition. A finitary relation $f$ of rank $n$, $n \geq 1$, on a universal set $A$, is a function defined on a subset $D$ of one of the finite cartesian powers of $A$, $A^n$, with values on $A$. It is called **total** if the domain $D$ is all of $A^n$. If $D$ is a proper subset of $A^n$, then it is called **partial**.

Definition. A relational structure is a sequence of the form $(A_1, A_2, \ldots, A_m; f_1, f_2, \ldots; e_1, e_2, \ldots)$ where for $i=1, 2, \ldots, m$, $m \geq 1$, all the $A_i$ are disjoint non-empty classes consisting of a partition of a non-empty universe $A$; $f_1, f_2, \ldots$ is a finite sequence of finitary operations on $A$; and $e_1, e_2, \ldots$ is a finite or infinite sequence of distinguished elements of $A$.

Definition. The **type** $t$ of a relational structure $R=(A_1, A_2, \ldots, A_m; f_1, f_2, \ldots; e_1, e_2, \ldots)$ is a sequence of non-negative integers of the form $(m; n_1, n_2, \ldots; c)$ where $m$ is the number of partitions of $A$; $n_i$ is the rank of $f_i$, $i=1, 2, \ldots$; and $c$ is either a non-negative integer or $\infty$. 
indicating the number of elements in the sequence \( e_1, e_2, \ldots \).

An alphabet \( V \) is a finite non-empty set whose elements are called letters. The alphabet \( V \) is divided into two disjoint subsets: the terminals, denoted by \( V_T \), and the non-terminals, denoted by \( V_N \). A word over the alphabet \( V \) is a finite string consisting of zero or more letters of \( V \) where a letter may occur several times. The string consisting of zero letters is called the empty word and is denoted by \( \lambda \). The set of all words over an alphabet \( V \) is infinite and is denoted by \( V^* \). We denote the set corresponding to \( V^* - \{\lambda\} \) by \( V^+ \).

If \( \alpha \) and \( \beta \) are words over the alphabet \( V \), then their catenation \( \alpha \beta \) is also a word over \( V \). Catenation is an associative operation and the empty word \( \lambda \) is the identity with respect to this operation, thus \( \alpha \lambda = \lambda \alpha = \alpha \) for any \( \alpha \). For any word \( \alpha \) and any integer \( k \geq 0 \), \( \alpha^k \) means that word obtained by catenating \( k \) copies of \( \alpha \), hence \( \alpha^0 \) denotes \( \lambda \). The length of a word \( \alpha \), denoted by \( lg(\alpha) \), is the number of letter in \( \alpha \), duplications counted, hence \( lg(\lambda) = 0 \). The string \( \alpha \) occurs in a word \( \mu \) if there exist words \( \delta \) and \( \gamma \) such that \( \mu = \delta \alpha \gamma \).
A production $P_i$, denoted by $P_i = \alpha \rightarrow \beta$, is a rule which allows an occurrence of $\alpha$ to be replaced by $\beta$, that is, a string of the form $\delta \alpha \gamma$ may be changed to $\delta \beta \gamma$. A word $\alpha$ generates $\beta$, denoted by $\alpha \rightarrow^* \beta$, if and only if there exists a finite sequence of words over $V$, $\gamma_0, \gamma_1, \gamma_2, \ldots, \gamma_n$, $n \geq 0$, where $\gamma_0 = \alpha$ and $\gamma_n = \beta$ and $\gamma_i \rightarrow \gamma_{i+1}$, for $0 \leq i < n$. Such sequence is called a derivation and may also be denoted by

$$
\gamma_0 \xrightarrow{P_{i_0}} \gamma_1 \xrightarrow{P_{i_1}} \gamma_2 \xrightarrow{P_{i_2}} \ldots \xrightarrow{P_{i_{n-1}}} \gamma_n.
$$

The number $n$ is the length of the derivation and $\gamma_0$ is called the initial symbol or the starting symbol.

In general, $\rightarrow$ is a binary relation on $V^*$ and $\rightarrow^*$ is a reflexive transitive closure of $\rightarrow$ defined as follows:

1) $\alpha \rightarrow^* \alpha$ for all $\alpha \in V^*$. \hspace{1cm} (2.2-1)

2) If $\alpha \rightarrow^* \gamma$ and $\gamma \rightarrow \beta$, then $\alpha \rightarrow^* \beta$. \hspace{1cm} (2.2-2)

3) $\alpha \rightarrow^* \beta$ only if it can be established by a finite number of applications of (1) and (2).
2.3. Engeler's Definition of Formal Grammar

Definition. (Engeler) A formal grammar $G$ is a sequence of the form $(C_1, C_2, \ldots, C_m; g_1, g_2, \ldots)$ with the following properties:

1) The $C_i$ form a family of non-empty mutually exclusive sets called syntactical classes. They are usually, but do not necessarily have to be, strings of symbols over some non-empty set $V$.

2) Each $g_j$ is called a syntactical operation. For each $g_j$, there is associated a number $n$ and an $(n+1)$-tuple of the form $(k_1, k_2, \ldots, k_n; i)$ of non-negative integers among $1, 2, \ldots, m$ such that $g_j : C_{k_1} x C_{k_2} x \ldots x C_{k_n} \rightarrow C_i$. We extend the definition of $g_j$ to include the possibility that $g_j$ may be defined on

$$(C_{k_1 U k_2 U \ldots U k_n}, s_1) x \ldots x (C_{k_1 U k_2 U \ldots U k_n}, s_n).$$

Definition. If all the $C_i$ are represented as strings of letters from an alphabet $V$, then $g_j$ will often operate by catenating their arguments, and possibly various delimiters, in a specified order. If this is the case for all the $g_j$ in the grammar $G$, then we call $G$ a desirable grammar.
Remark. It can be shown that any grammar has an equivalent desirable grammar.

Definition. The elements of $C_1^UC_2^U...U^UC_m^U$ are called expressions. An expression $\sigma$ is composite if there exists a syntactical operation $g_i$ and expressions $\sigma_1, \sigma_2, ..., \sigma_n$ such that $\sigma = g_i(\sigma_1, \sigma_2, ..., \sigma_n)$. Expressions that are not composite are called atomic expressions.

Engeler also requires unique readability of a grammar. This is taken up in Section 2.5.

2.4. Conventional Definition of Formal Grammar

Definition. A formal grammar $G$ is an ordered quadruple of the form $(V_N, V_T, S, P)$ where $V_N \cap V_T = \emptyset$, $V_N \cup V_T = V$, $S \in V_N$ is the starting symbol and $P$ is the set of productions.

Definition. A grammar $G = (V_N, V_T, S, P)$ is said to be context-free if each production in $P$ is of the form $\alpha \rightarrow \beta$ where $\alpha \in V_N$ and $\beta \in V^*$.  

10
The language generated by some grammar G, denoted by L(G), is the set of words μ such that S→*μ. L(G) is a subset of VT.

If S→*μ, then there may be several derivations -- a sequence of applications of (2.2-1) and (2.2-2) -- which show this. However, there is always at least one leftmost derivation.

Definition. The derivation
\[ \gamma_0 \xrightarrow{P_i} \gamma_1 \xrightarrow{P_i} \gamma_2 \xrightarrow{P_i} \ldots \xrightarrow{P_i} \gamma_n \]
where S=γ₀ and n≥1, according to a context-free grammar G, is leftmost if and only if for each k=0,1,2,...,n-1, the production P₁=γ₁=αYβ is used to replace aγ₀ by αZβ where γ₁=αYβ, γ₁+1=αZβ, α is a word over VT, and Y is the leftmost non-terminal in γ₁.

Example. Consider the context-free grammar
G=(V_N,V_T,S,P) where V_N={X}, V_T={a,b}, S=X, P=[P₁=X→XX, P₂=X→a, P₃=X→b]. L(G) consists of all non-empty words over V_T. Consider the derivation
\[ X \xrightarrow{P₁} XX \xrightarrow{P₂} aX \xrightarrow{P₁} aXX \xrightarrow{P₃} abX \xrightarrow{P₂} aba \]
and the derivation
\[ X \xrightarrow{P_1} xx \xrightarrow{P_2} x\alpha \xrightarrow{P_1} xx\alpha \xrightarrow{P_3} xba \xrightarrow{P_2} aba \]
and the derivation
\[ X \xrightarrow{P_1} xx \xrightarrow{P_2} a\alpha \xrightarrow{P_1} aXX \xrightarrow{P_2} a\alpha X \xrightarrow{P_3} aba. \]
The first is a leftmost derivation, whereas the second and third are not leftmost derivations.

A given language \( L \) may be generated by several grammars. If among these grammars some are context-free, then \( L \) is a context-free language.

Definition. A context-free grammar \( G \) is ambiguous if there is a word in \( L(G) \) possessing two distinct leftmost derivations from the starting symbol. Otherwise, \( G \) is unambiguous. A context-free language \( L \) is unambiguous if there is an unambiguous context-free grammar \( G \) such that \( L = L(G) \). Otherwise, \( L \) is called inherently ambiguous.
2.5. Non-Ambiguity or Unique Readability

Given a language generated by a grammar
\[ G=(C_1, C_2, \ldots, C_m; g_1, g_2, \ldots) \], \( G \) is not uniquely readable if there exist two syntactical operations \( g_i \) and \( g_j \), \( i \neq j \), such that \( a = g_i(\beta) \) and \( a = g_j(\delta) \) where \( a \in L(G) \), or if there exists a syntactical operation \( g_k \) such that \( a = g_k(\beta) \) and \( a = g_k(\delta) \) where \( \beta \neq \delta \). Otherwise, \( G \) is uniquely readable.

Non-ambiguity or unique readability plays a vital role in the interpretation of the syntax of a language determined by its grammar. Consider the following examples, where each \( C_i \) consists of strings of symbols.

Example. Suppose we have the following syntactical classes:
\[
\begin{align*}
C_1 &= \{k_1, k_2, \ldots\} \\
C_2 &= \{x_1, x_2, \ldots\} \\
C_3 &= \{t: t \text{ is of the form } 'a_i \cdot a_j' \text{ or } 'a_i \div a_j', a_i \in C_1 \cup C_2, a_j \in C_1 \cup C_3\} \\
C_4 &= \{e: e \text{ is of the form } 'a_i \cdot a_j' \text{ or } 'a_i \div a_j', a_i \in C_1 \cup C_2 \cup C_3 \cup C_4\} \\
\end{align*}
\]
\[
a_j \in C_1 \cup C_2 \cup C_3, \]
\[C_5 = [\div, \div],\]
\[C_6 = [\times, -],\]
\[C_7 = [f; f \text{ is of the form } 'a_1 = a_j', \]
\[a_1 \in C_2, \quad a_j \in C_1 \cup C_2 \cup C_3 \cup C_4,\]

and we define the syntactical operations as follows:

\[g_1 : C_2 \times (C_1 \cup C_2 \cup C_3 \cup C_4) \rightarrow C_7 \text{ such that} \]
\[g_1(a_k, a_1) = 'a_k = a_1',\]

\[g_2 : (C_1 \cup C_2 \cup C_3 \cup C_4) \times (C_1 \cup C_2 \cup C_3) \times C_6 \rightarrow C_4 \text{ such that} \]
\[g_2(a_k, a_1, a_m) = 'a_k = a_m',\]

\[g_3 : (C_1 \cup C_2 \cup C_3 \cup C_4) \times (C_1 \cup C_2) \times C_5 \rightarrow C_3 \text{ such that} \]
\[g_3(a_k, a_1, a_m) = 'a_k = a_m'.\]

Hence, \(G = (C_1, C_2, C_3, C_4, C_5, C_6, C_7; g_1, g_2, g_3)\).

Let the following correspondences hold:

\[X = C_1\]
\[V = C_2\]
\[F = XUV = C_1 \cup C_2\]
\[T = VUC_3 = C_1 \cup C_2 \cup C_3\] \hspace{1cm} (2.5-2)
\[E = TUC_4 = C_1 \cup C_2 \cup C_3 \cup C_4\]
\[M = C_5\]
\[A = C_6\]

We will also find that \(L(G) = C_7\).
The productions associated with the syntactical operations may be defined as follows:

\[ P_1 = S \rightarrow V = E \]
\[ P_2 = E \rightarrow \text{EAT} \]
\[ P_3 = T \rightarrow \text{TMF} \]
\[ P_4 = E \rightarrow T \]
\[ P_5 = T \rightarrow F \]
\[ P_6 = F \rightarrow V \]
\[ P_7 = F \rightarrow K \]
\[ P_8_i = V \rightarrow x_i \] (a series of productions for \( i = 1, 2, \ldots \))
\[ P_9_i = K \rightarrow k_i \] (a series of productions for \( i = 1, 2, \ldots \))
\[ P_{10} = M \rightarrow * \]
\[ P_{11} = M \rightarrow \dagger \]
\[ P_{12} = A \rightarrow + \]
\[ P_{13} = A \rightarrow - \]

where

\[ V_N = \{ S, E, T, F, V, K, M, A \} \]
\[ V_T = \{ k_1, k_2, \ldots, x_1, x_2, \ldots, *, \dagger, +, -, = \} \]
\( S = \) the starting symbol

\[ P = \{ P_i, \text{ } P_i \text{ is a production for } i = 1, 2, \ldots, 13 \} \]
We also note the correspondences between $P_1$ and $g_1$, $P_2$ and $g_2$ and $P_3$ and $g_3$. $P_i$, $i=4,5,6,7$, are inclusions as a result of the correspondences in (2.5-2); and $P_j$, $j=8,9,10,11,12,13$, are memberships arising from classes $C_1$, $C_2$, $C_5$ and $C_6$ defined by enumeration in (2.5-1).

Hence, $G=(V_N,V_T,S,P)$ and $L(G)=[f; f$ is of the form $'x=a'$, $f\in V_T^+]$. We now exhibit that there exists a leftmost derivation for the word $x_3=k_1*x_2\hat{=}k_4$ in $L(G)$.

Step

1. $S \longrightarrow V=E$
2. $P_8 \longrightarrow x_3=E$
3. $P_4 \longrightarrow x_3=T$
4. $P_3 \longrightarrow x_3=TMF$
5. $P_3 \longrightarrow x_3=TMFMF$
6. $P_5 \longrightarrow x_3=FMFMF$
7. $P_7 \longrightarrow x_3=KFMF$
8. $P_9 \longrightarrow x_3=k_1MFMF$
We shall now prove that the above leftmost derivation is unique for the given word.

Step

1. From the starting symbol $S$, only the production $P_1$ may be applied to it.

2. To obtain $x_3$, only production $P_8$ may be applied on the leftmost non-terminal $V$.

3. For the non-terminal $E$, either productions $P_2$ or $P_4$ can be applied; but since there is no string in the word of the form 'a+b' or 'a-b', then only production $P_4$ may be applied on $E$.

4. Similarly, either productions $P_3$ or $P_5$ can be
Step

applied on the leftmost non-terminal T; but since
the righthand side of the symbol = is not of length
1, then only production $P_3$ may be applied, making
it potentially of length 3.

5 Again, either productions $P_3$ or $P_5$ can be applied
on the leftmost non-terminal T; but since the
righthand side of the symbol = is of length 5, then
only production $P_3$ may be applied.

6 Having attained the required length of the word, we
now work on the leftmost non-terminal T to bring it
to a terminal. Hence, only production $P_5$ may be
applied on T.

7 Either productions $P_6$ or $P_7$ can be applied on the
leftmost non-terminal F; but in order to obtain $k_1$,
only production $P_7$ may be applied.

8 Similarly, to obtain $k_1$, only production $P_{9_1}$ may be
applied on the leftmost non-terminal K.

9 To the leftmost non-terminal M, either productions
$P_{10}$ or $P_{11}$ can be applied; but since the desired
symbol is *, then only production $P_{10}$ may be
applied.

10 To the leftmost non-terminal F, either productions
$P_6$ or $P_7$ can be applied; but in order to obtain $x_2$. 

18
Step

11 Similarly, to obtain $x_2$, only production $P_{8_2}$ may be applied on the leftmost non-terminal $V$.

12 To the leftmost non-terminal $M$, either productions $P_{10}$ or $P_{11}$ can be applied; but since the desired symbol is $\div$, then only production $P_{11}$ may be applied.

13 To the leftmost non-terminal $F$, either productions $P_6$ or $P_7$ can be applied; but in order to obtain $k_4$, only production $P_7$ may be applied.

14 Finally, to obtain $k_4$, only production $P_{9_k}$ may be applied on the leftmost non-terminal $K$.

It is easy to see that for every word in $L(G)$ there exists a unique leftmost derivation. Hence $G$ is unambiguous.

Suppose we introduce a set of changes as follows:

Let

$C'_1 = C_1$

$C'_2 = C_2$

$C'_3 = \{ t : t$ is of the form $'a_i \cdot a_j'$ or $'a_i \div a_j'$, $a_i, a_j \in C_1 \cup C_2 \cup C_3 \}$

$C'_4 = \{ e : e$ is of the form $'a_i \cdot a_j'$ or $'a_i - a_j'$, $a_i, a_j \in C_1 \cup C_2 \cup C_3 \cup C_4 \}$
\[ C_5' = C_5 \]
\[ C_6' = C_6 \]
\[ C_7' = \{ f : f \text{ is of the form } a_i = a_j, a_i \in C_2', a_j \in C_2', a_i \in C_2' \cup C_3' \cup C_4' \} \]

\[ e_1'C_2' x (C_1' \cup C_2' \cup C_3' \cup C_4') \rightarrow C_7' \]
\[ e_2'(C_1' \cup C_2' \cup C_3') x (C_1' \cup C_2' \cup C_3') x C_6' \rightarrow C_4' \]
\[ e_3'(C_1' \cup C_2' \cup C_3') x (C_1' \cup C_2' \cup C_3') x C_5' \rightarrow C_3' \]

Hence, \( G' = (C_1', C_2', C_3', C_4', C_5', C_6', C_7', e_1', e_2', e_3') \).

We also have the following changes in the associated productions:

\[ P_2' = E \rightarrow EAE \]
\[ P_3' = T \rightarrow TMT \]
\[ P_i' = P_i \text{ for all } i, i \neq 2, 3 \]
\[ V_N' = V_N \]
\[ V_T' = V_T \]
\[ S' = S \]
\[ P' = \{ P_i' : P_i \text{ is a production for } i = 1, 2, \ldots, 13 \} \]

Hence, \( G' = (V_N', V_T', S', P') \).

We shall now show that the word \( x_3 = k_1 x_2^- k_4 \) is in \( L'(G') \), but that it does not possess a unique leftmost derivation.
\[ S' \quad P_1' \quad \forall = E \]

\[ P_5' \quad (x_3 = E) \]

\[ P_4' \quad (x_3 = \text{TMT}) \]

\[ P_5' \quad (x_3 = \text{FMT}) \quad \text{or} \quad P_3' \quad (x_3 = \text{TMT\text{TMT}}) \]

\[ P_7' \quad (x_3 = \text{KMT}) \]

\[ P_9' \quad (x_3 = k_1 \text{MT}) \]

\[ P_10' \quad (x_3 = k_1 \text{*T}) \]

\[ P_3' \quad (x_3 = k_1 \text{*TMT}) \]

\[ P_5' \quad (x_3 = k_1 \text{*FMT}) \]

\[ P_6' \quad (x_3 = k_1 \text{*VMT}) \]

\[ P_8' \quad (x_3 = k_1 \text{*x}_2 \text{MT}) \]

\[ P_9' \quad (x_3 = k_1 \text{*x}_2 \text{\text{\textsuperscript{\texttt{T}}}}) \]

\[ P_5' \quad (x_3 = k_1 \text{*x}_2 \text{\textsuperscript{\texttt{F}}}) \]

\[ P_3' \quad (x_3 = k_1 \text{*x}_2 \text{\textsuperscript{\texttt{T}}}) \]

\[ P_5' \quad (x_3 = k_1 \text{*x}_2 \text{\textsuperscript{\texttt{F}}}) \]

21
The first derivation is equivalent to the interpretation 
\[ x_3 = k_1 \cdot x_2 \div k \]
whereas the second is equivalent to 
\[ x_3 = (k_1 \cdot x_2) \div k_4 \]. Hence, \( G' \) is ambiguous. Clearly, 
\[ L'(G') = L(G) = L \], therefore \( L \) is not inherently ambiguous.

We also wish to note that the string \( k_1 \cdot x_2 \div k_4 \) is generated by either of two ways: \( g_3(k_1 \cdot x_2, k_4, \div) \) or 
\( g_3(k_1, x_2 \div k_4, \times) \), which shows that \( G' \) is not uniquely readable.

Example. Consider the following example. Let

\[ V_N = \{S, A, B\} \]
\[ V_T = \{x\} \]
\( S = \) starting symbol
\( P = \) set of productions which are as follows:

\[ P_1 = S \rightarrow A \]
\[ P_2 = S \rightarrow B \]
\[ P_3 = S \rightarrow AB \]
\[ P_4 = A \rightarrow xxxA \]
\[ P_5 = A \rightarrow xxx \]
If $G=(V_N, V_T, S, P)$, then $L(G)=\{x^n : n=3, 4 \text{ or } n \geq 6\}$. Take $x^6$. Its derivation is $S \rightarrow x^3x^3$. Similarly, $x^7$ has derivation $S \rightarrow x^3x^4$, $x^8$ has derivation $S \rightarrow x^4x^4$, $x^9$ has derivation $S \rightarrow x^3x^3x^3$, $x^{10}$ has derivation $S \rightarrow x^3x^3x^4$ and $x^{11}$ has derivation $S \rightarrow x^3x^4x^4$. We encounter our first problem with $x^{12}$ since its derivation can either be $S \rightarrow x^3x^3x^3x^3$ of $S \rightarrow x^4x^4x^4$. Hence there does not exists a unique leftmost derivation for $x^{12}$. Therefore $G$ is ambiguous.

The corresponding syntactical classes and operations may be represented as follows:

- $C_1=\{x\}$
- $C_2=\{x^{3n_1} : n=0, 1, 2, \ldots\}$
- $C_3=\{x^{4n_2} : n=0, 1, 2, \ldots\}$
- $C_4=\{x^{3n_1}ax^{4n_2}b : n_1=0, 1, 2, \ldots ; n_2=0, 1, 2, \ldots\}$
- $C_5=\{x^n : n=3, 4 \text{ or } n \geq 6\}$
- $g_1 : C_1 \rightarrow C_2$ such that $g_1(x)=x^{3n_1}a$, $n=0, 1$
- $g_2 : C_1 \rightarrow C_3$ such that $g_2(x)=x^{4n_2}b$, $n=0, 1$
- $g_3 : C_1 \rightarrow C_4$ such that $g_3(x)=x^{3n_1}ax^{4n_2}b$, $n_1=0, 1; n_2=0, 1$
- $g_4 : C_2 \rightarrow C_2$ such that $g_4(x^{3n_1}a)=x^{3(n+1)}a$.

23
n=0,1,2,...
\[ g_5 \circ C_2 \rightarrow C_5 \] such that \[ g_5(x^{3n}a) = x^{3(n+1)}, \]
\[ n=0,1,2,... \]
\[ g_6 \circ C_3 \rightarrow C_3 \] such that \[ g_6(x^{4n}b) = x^{4(n+1)}b, \]
\[ n=0,1,2,... \]
\[ g_7 \circ C_3 \rightarrow C_5 \] such that \[ g_7(x^{4n}b) = x^{4(n+1)}, \]
\[ n=0,1,2,... \]
\[ g_8 \circ C_4 \rightarrow C_4 \] such that \[ g_8(x^{3n_1}a x^{2n_2}b) = x^{3n_1}a x^{2n_2}b, \]
\[ g_8(x^{3n_1}a x^{2n_2}b) = x^{3(n_1+s)}a x^{2(n_2+t)}b, \]
\[ n_1=0,1,2,...; s=0,1; n_2=0,1,2,...; t=0,1 \]
\[ g_9 \circ C_4 \rightarrow C_5 \] such that \[ g_9(x^{3n_1}a x^{2n_2}b) = x^{3(n_1+s)}a x^{2(n_2+t)}b, \]
\[ n_1=0,1,2,...; s=0,1; n_2=0,1,2,...; t=0,1 \]
\[ G = (C_1, C_2, C_3, C_4, C_5; g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8, g_9) \] and \( G \) is not uniquely readable since \( g_5(x^9a) = x^{12} \) and \( g_7(x^8b) = x^{12} \).
Note that \( L(G) = C_5 \) and that the following correspondences hold:

\[ g_i \leftrightarrow P_i, \quad i=1,2,...,7 \]
\[ V_T \leftrightarrow C_1 \]
\[ A \leftrightarrow C_2 \]
\[ B \leftrightarrow C_3 \]
\[ AB \leftrightarrow C_4 \]
We wish to note that since Engeler requires only that the syntactical classes be closed under the syntactical operations, the classes may also be enlarged as follows:

\[ C'_1 = \{ x \} \]
\[ C'_2 = \{ x^n_a : n = 0, 1, 2, \ldots \} \]
\[ C'_3 = \{ x^n_b : n = 0, 1, 2, \ldots \} \]
\[ C'_4 = \{ x^{n_1 a} x^{n_2 b} : n_1 = 0, 1, 2, \ldots ; n_2 = 0, 1, 2, \ldots \} \]
\[ C'_5 = \{ x^n : n \geq 2 \} \]

However, \( C'_5 \subseteq L(G) \). \( L(G) \) is a subset of \( C'_5 \) which consists of only the derivable words from the grammar \( G \).

We have shown with the examples above how an Engeler's formal grammar and a corresponding conventional formal grammar may be derived from each other by establishing a correspondence between the union of syntactical classes and the non-terminals and between the class of minimum derivable words from the grammar \( G \) and the language \( L(G) \).

Remark. We extend the definition of formal grammar to include the condition that for every \( Q \in V_N \), there exists a leftmost derivation
for some $\mu \in L(G)$, $a, b \in V_+^*$ and $c \in V_T^*$.

Let $L=L(G)$ be also denoted by $L(G, S)$ where $S$ is the starting symbol. Let $L(G, Q)$, where $Q \in V_N$, be the language generated by $G$ with $Q$ as the starting symbol.

Theorem. If $L(G)$ is unambiguous, then each $L(G, Q)$ is unambiguous.

Proof. If $L(G)$ is unambiguous, then there exists a unique leftmost derivation

$$S \rightarrow \ast aQ\beta \rightarrow \ast a\sigma\beta \rightarrow \ast a\sigma\delta = \mu$$

for some $\mu \in L(G)$, $a, \delta \in V_T^*$ and $\sigma \in V_T^*$. Suppose $L(G, Q)$ is ambiguous. There would exist two distinct leftmost derivations

$$Q \rightarrow \ast Z \rightarrow \ast X \rightarrow \ast \gamma$$

and

$$Q \rightarrow \ast Z \rightarrow \ast Y \rightarrow \ast \gamma$$

where $X \neq Y$ and $\gamma \in L(G, Q)$. Hence, the existence of two distinct leftmost derivations

$$S \rightarrow \ast aQ\beta \rightarrow \ast aZ\beta \rightarrow \ast aX\beta \rightarrow \ast a\gamma\beta \rightarrow \ast a\gamma\delta = \nu$$

and

$$Q \rightarrow \ast Z \rightarrow \ast X \rightarrow \ast \gamma$$

and

$$Q \rightarrow \ast Z \rightarrow \ast Y \rightarrow \ast \gamma$$
\[ S \vdash a_{Q\beta} \vdash a_{Z\beta} \vdash a_{Y\beta} \vdash a_{Y\delta} \vdash a_{Y\delta = \nu} \]

for \( \nu \in L(G) \). A contradiction!
Chapter 3
Partial Recursive Functions and Machines

3.1. Some Basic Definitions

Definition. Let $K$ be a class of similar relational structures and let $L$ be the language associated to it. An **algorithmic basis** $B$ for $K$ is a pair of sets $(\Sigma, \Theta)$ where $\Sigma$ is the set of formulas for $L$ called **conditions** and $\Theta$ is the set of **operations**. At least one of $\Sigma, \Theta$ is non-empty.

Definition. A **machine** confronted with a relational structure $R$ from $K$ consists of some **memory set** labelled with variables of the language $L$ and equipped with the basic capabilities namely: a **decision-making capability** and an **operating capability**.
To an algorithmic basis, an algorithmic language can be associated to formulate composite instructions or composite programs on the basic capabilities of the machine.

Definition. The expressions of the algorithmic language $\text{Alg}(B)$ associated with the algorithmic basis $B=(\Sigma, \theta)$ are divided into the following syntactical classes:

1) **Labels**: These are numerals $1, 2, 3, ...$

2) **Conditions**: These are elements of $\Sigma$.

3) **Operations**: These are elements of $\theta$.

4) **Conditional instructions**: These are expressions of the form

   $$i: \text{IF } \sigma \text{ THEN GO TO } j \text{ ELSE GO TO } k;$$

   where $\sigma$ is a condition and $i$, $j$, and $k$ are labels.

5) **Operational instructions**: These are expressions of the form

   $$i: \text{DO } \nu \text{ THEN GO TO } j;$$

   where $\nu$ is an operation and $i$ and $j$ are labels. If $j=i+1$, then we delete the phrase THEN GO TO $j$.

6) **Program**: This is a finite set of instructions such that no two different instructions have the same label.
Let \( \pi_i, i=1,2, \ldots \), be a program with one entrance and one exit. Given \( \pi_1 \) and \( \pi_2 \), the composition of \( \pi_1 \) and \( \pi_2 \) into \( \pi_3 \) is formulated by taking all the instructions of \( \pi_1 \) into \( \pi_3 \) unchanged and by taking all the instructions of \( \pi_2 \) into \( \pi_3 \) and making them distinct. If \( m \) is the largest label in \( \pi_1 \), then we add \( m \) to every label in \( \pi_2 \) except the entrance which is made to correspond to the exit of \( \pi_1 \). Hence, the entrance of \( \pi_1 \) becomes the entrance of \( \pi_3 \) and the exit of the altered \( \pi_2 \) becomes the exit of \( \pi_3 \). Composition of programs may be extended to more than two one-exit programs.

Actual programming languages use other notations and terminologies. Two such important terminologies are procedures and subroutines ranging over a program. Hence, we can speak of the composition of procedures and subroutines. In the following sections we shall deal with the composition of subroutines.

Programs or procedures or subroutines may have more than one exit. Consider for instance a subroutine containing a conditional instruction as follows:
i: IF v THEN GO TO j ELSE GO TO k;

j: DO α THEN GO TO EXIT-A;

k: DO β THEN GO TO EXIT-B;

Let such a subroutine be $π_1$ which is a two-exit subroutine. The subroutine $π_1$ may then be composed

$π_2$ where EXIT-A coincides with the entrance of $π_2$ or

$π_3$ where EXIT-B coincides with the entrance of $π_3$.

3.2. Partial Recursive Functions

Definition. A partial recursive function is a partial function that can be obtained from the

1) **Successor function**: $x' = x + 1$;

2) **Constant function**: For any $i=1,2,...,n$, $F(x_i) = k$ where $k$ is a constant;

3) **Projection function**: $G_k(x_1,x_2,...,x_k,...,x_n) = x_k$,

and by the repeated application of

4) **Composition**: If $g$ and $h$ are partial recursive functions, then so is $f:f(x) = h \circ g(x)$ where $h \circ g(x) = h(g(x))$;

5) **Primitive recursion**: If $g$ and $h$ are partial recursive functions, then so is $f$ which is defined by
\[ f(x,0) = g(x) \]
\[ f(x,y') = h(x,y,f(x,y)) \]

6) **Least number operator**: If \( g \) and \( h \) are partial recursive functions, then so is \( f \) if \( f(x) = \text{least } y \) such that \( g(x,y) = h(x,y) \), otherwise \( f \) is undefined.

Any machine that can effectively simulate all partial recursive functions can perform the following numerical operations, namely: addition, multiplication, exponentiation, magnitude, minimum, maximum, subtraction, division and modular division, since they are partial recursive functions as shown below:

1) **Addition**: \( f(x,y) \) means \( x + y \)
   \[ f(0,y) = y \]
   \[ f(x',y) = [f(x,y)]' \]

2) **Multiplication**: \( f(x,y) \) means \( x \times y \)
   \[ f(0,y) = 0 \]
   \[ f(x',y) = f(x,y) + y \]

3) **Exponentiation**: \( f(x,y) \) means \( y^x \)
   \[ f(0,y) = 1 \]
   \[ f(x',y) = y \times f(x,y) \]

4) **Magnitude**: \( f(x,y) \) means \( |x-y| \)
   Since \( |x-y| = \sqrt{(x+y)^2 - z^2} \), then
   \[ f(x,y) = \mu_z [z^2 = (x+y)^2] \]

32
5) Minimum: \( f(x,y) \) means \( \min(x,y) \)

The least \( z \) such that \((z-x)(z-y)=0\) will give the minimum between \( x \) and \( y \), but since subtraction has not been defined, then we restate the function as follows:

\[
 f(x,y) = \mu_z [z^2 + (x*y) = z*(x+y)]
\]

6) Maximum: \( f(x,y) \) means \( \max(x,y) \)

\[
 f(x,y) = [\min(x,y) + |x-y|]
\]

7) Subtraction: \( f(x,y) \) means \( x-y \)

\[
 f(x,y) = |x - \min(x,y)|
\]

8) Division: \( f(x,y) \) means \( x\div y \)

\[
 f(x,y) = \mu_z [(x\div (y*z)) = \min[(x\div (y*z)), y]=0]
\]

9) Modular division: \( f(x,y) \) means \( x \) mod \( y \)

\[
 f(x,y) = x\div [(x\div y)*y]
\]

3.3. Shepherdson-Sturgis or SS Machine

Definition. The SS Machine consists of the following: a memory set with variables \( x_0, x_1, x_2, \ldots \) with non-negative natural numbers for values (although real computers have bounded word size) and an algorithmic basis \( B=(\Sigma, \theta) \) where

\[
 \Sigma = \{ x_i = 0, i=0,1,2, \ldots \}
\]
\[ \theta = \{ \theta_1, \theta_2, \theta_3, \theta_4 \} \text{ where} \\
\theta_1 = \{ x_i = x_i + 1, \ i = 0, 1, 2, \ldots \} \\
\theta_2 = \{ x_i = x_i - 1, \ i = 0, 1, 2, \ldots \} \\
\theta_3 = \{ x_i = 0, \ i = 0, 1, 2, \ldots \} \\
\theta_4 = \{ x_i = x_j, \ i \neq j, \ j = 0, 1, 2, \ldots \} \]

**Theorem.** The SS Machine can compute all partial recursive functions.

**Proof.** To prove that the SS Machine can compute all partial recursive functions, it is sufficient to write subroutines for the following functions according to the definition.

a) **Successor function:** Define for \( x' = x + 1 \). Using \( \theta_1 \), we have the subroutine

\[ \text{MACRO SUCCESSOR (IN AND OUT } X1); \]
\[ 1: \text{DO } X1 := X1 + 1 \text{ THEN GO TO EXIT;} \]

In place of SUCCESSOR (XI), we shall refer to MACRO SUCCESSOR by writing \([XI := XI']\).

b) **Constant function:** Define for \( x = k \). Using \( \theta_3 \) and \([XI := XI']\), we have the subroutine for say \( k = 3 \).
MACRO CONSTANT3 (OUT X1);
1: DO X1 := 0;
2: DO [X1 := X1'];
3: DO [X1 := X1'];
4: DO [X1 := X1'] THEN GO TO EXIT;

c) Projection function: Define for \( G_k(x_1, x_2, \ldots, x_n) = x_k \). Using \( \theta_4 \), we have the subroutine

MACRO PROJECTION (IN X1; OUT X2);
1: DO X2 := X1 THEN GO TO EXIT;

d) Composition: Define for \( h \circ g(x) \). Let \( g \) and \( h \) be partial recursive functions. We have the subroutine

MACRO COMPOSITION (IN X1; OUT X3);
1: DO [X2 := G(X1)];
2: DO [X3 := H(X2)] THEN GO TO EXIT;

e) Primitive recursion: Define for \( f(x, 0) = g(x) \) and \( f(x, y') = h(x, y, f(x, y)) \). Let \( g \) and \( h \) be partial recursive functions. We have the subroutine

MACRO PRIMITIVE (IN X1, X2; OUT X4);
1: DO X0 := 0;
2: DO [X3 := G(X1)];
3: IF X0 = X2 THEN GO TO 6 ELSE GO TO 4;
4: DO [X3 := H(X1,X2,X3)];
5: DO [X2 := X2'] THEN GO TO 3;
6: DO X4 := X3 THEN GO TO EXIT;

f) Least number operator: Define for
f(x)=\mu_y[g(x,y)=h(x,y)]. Let g and h be partial recursive functions. We have the subroutine

MACRO LEAST (IN X1; OUT X5);
1: DO X2 := 0;
2: DO [X3 := G(X1,X2)];
3: DO [X4 := H(X1,X2)];
4: IF X3 = X4 THEN GO TO 6 ELSE GO TO 5;
5: DO [X2 := X2'] THEN GO TO 2;
6: DO X5 := X2 THEN GO TO EXIT;

Suppose we weaken the SS Machine into the SS' Machine by simplifying B into B'=(\Sigma',\theta') where
\Sigma' = \Sigma
\theta' = [\theta_1,\theta_2]
Let the SS' Machine have a memory set with variables x_0,x_1,x_2,... on B'.

Theorem. The SS' Machine can effectively simulate the SS Machine.
Proof. We have to show that the SS' Machine is capable of computing $\theta_3$ and $\theta_4$.

a) We compute $\theta_3$ with the subroutine

MACRO ZERO (IN AND OUT XI);
1: IF XI = 0 THEN GO TO EXIT ELSE GO TO 2;
2: DO XI := XI - 1 THEN GO TO 1;

In place of ZERO (XI), we shall refer to MACRO ZERO by writing $[XI := 0]$.

b) We compute $\theta_4$ with the subroutine

MACRO ASSIGN (IN X1; OUT X2);
1: DO [X2 := 0];
2: IF X1 = 0 THEN GO TO EXIT ELSE GO TO 3;
3: DO X2 := X2 + 1;
4: DO XI := XI - 1 THEN GO TO 2;

In place of ASSIGN (XI,XJ), we shall refer to MACRO ASSIGN by writing $[XJ := XI]$.

3.4. Tag Machine

The Tag Machine operates on a relational structure whose underlying set is the set of all words $x$ over $\{0,1\}$. 
Definition. The Tag Machine consists of the following: a memory set with one element $x$ and an algorithmic basis $B^t=(\Sigma^t, \theta^t)$ where

\[ \Sigma^t=\{b_0(x), b_1(x), e(x)\} \] such that

- $b_0(x)$ is true if the word $x$ begins with 0,
- $b_1(x)$ is true if the word $x$ begins with 1,
- $e(x)$ is true if the word $x$ is empty;

\[ \theta^t=\{\theta_1^t, \theta_2^t, \theta_3^t\} \] where

- $\theta_1^t=\{x_1=x_0\}$
  (tag 0 at the right of $x$)
- $\theta_2^t=\{x_1=x_1\}$
  (tag 1 at the right of $x$)
- $\theta_3^t=\{x_1=d(x)\}$
  (delete the first symbol of $x$).

Given Machine $A$ with program $\pi_A$, with initial data and final data. Given Machine $B$. Machine $A$ can be

**weakly but effectively simulated** on Machine $B$ if there exist effective encoding rules to transform program $\pi_A$ into a program $\pi_B$ for Machine $B$ and the initial data of Machine $A$ into initial data for Machine $B$, and the relevant arising words in the final data of Machine $B$.
into the final data form of Machine A such that both machines arrive at the same solution.

Theorem. The Tag Machine can weakly but effectively simulate the SS' Machine.

Proof. To every natural number n, we associate the word 1...1 of length n+1, which we shall denote by \( \bar{n} \). Suppose the SS' Machine has memory content variables \( x_0, x_1, x_2, \ldots, x_m \) with values \( n_0, n_1, n_2, \ldots, n_m \). We associate to it the Tag Machine memory content which is of the form \( \bar{n}_0 \bar{n}_1 \bar{n}_2 0 \ldots 0 \bar{n}_m \). The final data will also be in a similar form as the initial data. Hence, to encode the Tag Machine's final data back into the final data form of the SS' Machine, we simply associate the value \( n^i \), \( i=0,1,2,\ldots \), for each \( \bar{n}_i \).

Since the instructions of the Tag Machine can be performed on the beginning or the end of the word \( x \), to get to \( \bar{n}_1 \) we need the following subroutine which transforms \( \bar{n}_0 \bar{n}_1 \bar{n}_2 0 \ldots 0 \bar{n}_m \) to \( \bar{n}_1 \bar{n}_0 \bar{n}_2 0 \ldots 0 \bar{n}_m 0 \bar{n}_0 \). Let \( x=\bar{n}_0 \bar{n}_1 \bar{n}_2 0 \ldots 0 \bar{n}_m \).

MACRO SHIFT;
1: DO \( X := X0; \)

39
To get to \( \bar{n}_i \), we perform the above subroutine \( i-1 \) times. After performing the desired operations on \( \bar{n}_i \), perform the above subroutine until the word \( x \) is in its original form.

We also have to show how to simulate \( \Sigma' \) and \( \theta' \).

We shall denote a comment within a subroutine as follows:

\((* \text{COMMENT} *)\).

a) Is \( x_i = 0 \), where \( x_i \) is an SS' Machine word? For say, \( i=2 \), where \( x_2 = 3 \) which means \( \bar{n}_2 = 1111 \), and \( m = 6 \), we have the subroutine

MACRO TEST -2-6;

1: DO [SHIFT]; \((* \quad \bar{n}_1 0\bar{n}_2 0 \ldots 0\bar{n}_6 0\bar{n}_0 \quad *)\)
2: DO [SHIFT]; \((* \quad \bar{n}_2 0\bar{n}_3 0 \ldots 0\bar{n}_0 0\bar{n}_1 \quad *)\)
3: DO X := X0; \((* \quad 11110\bar{n}_3 0 \ldots 0\bar{n}_0 0\bar{n}_1 0 \quad *)\)
4: DO X := X1; \((* \quad 11110\bar{n}_3 0 \ldots 0\bar{n}_0 0\bar{n}_1 01 \quad *)\)
5: DO X := D(X); \((* \quad 1110\bar{n}_3 0 \ldots 0\bar{n}_0 0\bar{n}_1 01 \quad *)\)
6: IF \( \theta_0(X) \) THEN GO TO 15 ELSE GO TO 7;
7: DO X := X1;
103: DO [SHIFT]; (* $\tilde{n}_60\tilde{n}_00...0\tilde{n}_40\tilde{n}_5$ *)
104: DO [SHIFT] THEN GO TO EXIT;
(* $\tilde{n}_00\tilde{n}_10...0\tilde{n}_50\tilde{n}_6$ *)

c) Define $\tilde{n}_i = \tilde{n}_i - 1$ for say, $i=2$, where $m=6$. We have the subroutine

MACRO SUBTRACT1-2-6;
1: DO [SHIFT]; (* $\tilde{n}_10\tilde{n}_20...0\tilde{n}_60\tilde{n}_0$ *)
2: DO [SHIFT]; (* $\tilde{n}_20\tilde{n}_30...0\tilde{n}_00\tilde{n}_1$ *)
10: DO $X := D(X)$; (* $\tilde{n}_2,0\tilde{n}_30...0\tilde{n}_00\tilde{n}_1$ *)
101: DO [SHIFT]; (* $\tilde{n}_30\tilde{n}_40...0\tilde{n}_10\tilde{n}_2$ *)
102: DO [SHIFT]; (* $\tilde{n}_40\tilde{n}_50...0\tilde{n}_2,0\tilde{n}_3$ *)
103: DO [SHIFT]; (* $\tilde{n}_50\tilde{n}_60...0\tilde{n}_30\tilde{n}_4$ *)
104: DO [SHIFT]; (* $\tilde{n}_60\tilde{n}_00...0\tilde{n}_40\tilde{n}_5$ *)
105: DO [SHIFT] THEN GO TO EXIT;
(* $\tilde{n}_00\tilde{n}_10...0\tilde{n}_50\tilde{n}_6$ *)

3.5. Turing Machine

The Turing Machine operates on a relational structure whose underlying set is the set of all words of the form $#w_1qw_2#$ over $\{0,1,q,#\}$ where $w_1$ and $w_2$ are words over $\{0,1\}$, $q$ is the state and $#$ is a delimiter.
8: DO X := D(X);
9: IF B_0(X) THEN GO TO 11 ELSE GO TO 7;
10: DO X := D(X); (* \( \tilde{n}_30\tilde{n}_40\ldots0\tilde{n}_10\tilde{n}_2 \) *)
11: DO [SHIFT]; (* \( \tilde{n}_40\tilde{n}_50\ldots0\tilde{n}_20\tilde{n}_3 \) *)
12: DO [SHIFT]; (* \( \tilde{n}_50\tilde{n}_60\ldots0\tilde{n}_30\tilde{n}_4 \) *)
13: DO [SHIFT]; (* \( \tilde{n}_60\tilde{n}_00\ldots0\tilde{n}_40\tilde{n}_5 \) *)
14: DO [SHIFT] THEN GO TO EXIT-1;
15: DO X := D(X); (* \( \tilde{n}_00\tilde{n}_10\ldots0\tilde{n}_50\tilde{n}_6 \) *)
16: DO [SHIFT]; (* \( \tilde{n}_30\tilde{n}_40\ldots0\tilde{n}_10\tilde{n}_2 \) *)
17: DO [SHIFT]; (* \( \tilde{n}_40\tilde{n}_50\ldots0\tilde{n}_20\tilde{n}_3 \) *)
18: DO [SHIFT]; (* \( \tilde{n}_50\tilde{n}_60\ldots0\tilde{n}_30\tilde{n}_4 \) *)
19: DO [SHIFT] THEN GO TO EXIT-0;
(* \( \tilde{n}_60\tilde{n}_00\ldots0\tilde{n}_40\tilde{n}_5 \) *)

b) Define \( \tilde{n}_i = \tilde{n}_i + 1 \) for say, \( i = 2 \), where \( m = 6 \). We have the subroutine

MACRO ADD1-2-6;
1: DO [SHIFT]; (* \( \tilde{n}_10\tilde{n}_20\ldots0\tilde{n}_60\tilde{n}_0 \) *)
2: DO [SHIFT]; (* \( \tilde{n}_20\tilde{n}_30\ldots0\tilde{n}_00\tilde{n}_1 \) *)
3: DO [SHIFT]; (* \( \tilde{n}_30\tilde{n}_40\ldots0\tilde{n}_10\tilde{n}_2 \) *)
10: DO X := X1; (* \( \tilde{n}_30\tilde{n}_40\ldots0\tilde{n}_10\tilde{n}_2 \) *)
101: DO [SHIFT]; (* \( \tilde{n}_40\tilde{n}_50\ldots0\tilde{n}_20\tilde{n}_3 \) *)
102: DO [SHIFT]; (* \( \tilde{n}_50\tilde{n}_60\ldots0\tilde{n}_30\tilde{n}_4 \) *)
Definition. The Turing Machine consists of the following: a memory set with one element \( x \) and an algorithmic basis \( B^T = (\Sigma^T, \theta^T) \) where

\[
\Sigma^T = \{ s_0(x), s_1(x) \} \text{ where for } i = 0, 1,
\]

\( s_i(x) \) is the set of words in which \( w_2 \) begins with \( i \).

\[
\theta^T = \{ r(x), l(x), p_0(x), p_1(x) \} \text{ where}
\]

\[
r(#w_1q0w_2\#) = #w_10qw_2\# \text{ if } w_2 \neq \lambda
\]

\[
r(#w_1q1w_2\#) = #w_11qw_2\# \text{ if } w_2 \neq \lambda
\]

\[
r(#w_1q0\#) = #w_10q0\#
\]

\[
r(#w_1q1\#) = #w_11q0\#
\]

\[
l(#w_10qw_2\#) = #w_1q0w_2\#
\]

\[
l(#w_11qw_2\#) = #w_1q1w_2\#
\]

\[
l(#qw\#) = #q0w\#
\]

\[
p_1(#w_1qw_2\#) = #w_1qw_2\#
\]

where \( w'_2 \) results from \( w_2 \) by replacing the first symbol by \( i \), \( i = 0, 1 \).

Theorem. The Turing Machine can weakly but effectively simulate the Tag Machine that simulates the SS' Machine.

Proof. Since the Tag Machine can weakly but effectively simulate an SS' Machine, then it would be
sufficient to show that the Turing Machine can simulate the Tag Machine that simulates the SS' Machine.

To the Tag Machine memory element $x$, we associate the word $\#qx\#$. The final data of the Turing Machine simulating the Tag Machine that simulates the SS' Machine may be of the form $\#0\ldots 0qx0\ldots 0\#$ where $x$ ends with 1. To encode it into the final data form of the Tag Machine and further into the final data form of the SS' Machine, we simply pick out $x$ which is of the form $\tilde{n}_00\tilde{n}_10\tilde{n}_20\ldots 0\tilde{n}_m$ and drop everything else, and to every $\tilde{n}_i$ we associate $x_i$, for $i=0,1,2,\ldots,m$. Since we are not concerned here with simply simulating the Tag Machine on the Turing Machine, likewise, we are not concerned with translating any intermediary Turing Machine data into the corresponding Tag Machine data.

We also wish to recall that the length of the word on a Turing Machine may only be increased. Let $x=\#0\ldots 0q\tilde{n}_00\tilde{n}_10\tilde{n}_20\ldots 0\tilde{n}_m0\ldots 0\#$. We have to show how to simulate $\Sigma^t$, if necessary, and $\Theta^t$.

a) Clearly, $s_0(x)$ is equivalent to $b_0(x)$.

b) Clearly, $s_1(x)$ is equivalent to $b_1(x)$.

c) When the Tag Machine is simulating the SS' Machine, the empty word will never occur, therefore, the test $e(x)$ of the Tag Machine will always fail. Hence
there is no need to simulate $e(x)$ on the Turing Machine.

d) To compute $\theta_1^t$, that is,

$$#0...0q\bar{n}_0\bar{n}_10...\bar{n}_m0...0\#0...0q\bar{n}_0\bar{n}_10...\bar{n}_m00...0\#,$$

we have the subroutine

MACRO TAG-0;
1: DO R(X);
2: IF $S_0(X)$ THEN GO TO 3 ELSE GO TO 1;
3: DO R(X);
4: IF $S_0(X)$ THEN GO TO 5 ELSE GO TO 1;
5: DO L(X);
6: IF $S_0(X)$ THEN GO TO 7 ELSE GO TO 5;
7: DO L(X);
8: IF $S_0(X)$ THEN GO TO 9 ELSE GO TO 5;
9: DO R(X);
10: DO R(X) THEN GO TO EXIT;

Initially, $x=#q\bar{n}_0\bar{n}_10...\bar{n}_m\#$. To simulate $\theta_1^t$, we perform MACRO TAG-0. However, as soon as $x$ takes on the form $#0...0q\bar{n}_0\bar{n}_10...\bar{n}_m0...0\#$, the above subroutine becomes unnecessary since the first 0 in $0...0$ after the last 1 is equivalent to the result of performing $\theta_1^t$ in the Tag Machine.
e) To compute \( \theta_2^t \), that is,
\[
#0...0q_n00_n10...0n_m0...0#-#0...0q_n00_n10...0n_m10...0#,\]
we have the subroutine

MACRO TAG-1;
1: DO R(X);
2: IF \( S_0(X) \) THEN GO TO 3 ELSE GO TO 1;
3: DO R(X);
4: IF \( S_0(X) \) THEN GO TO 5 ELSE GO TO 1;
5: DO L(X);
6: DO \( P_1(X) \);
7: DO L(X);
8: IF \( S_0(X) \) THEN GO TO 9 ELSE GO TO 7;
9: DO L(X);
10: IF \( S_0(X) \) THEN GO TO 11 ELSE GO TO 7;
11: DO R(X);
12: DO R(X) THEN GO TO EXIT;

f) To compute \( \theta_3^t \), that is,
\[
#0...0q_n00_n10...0n_m0...0#-#0...0q_d(n_0)0...0n_m0...0#,\]
we have the subroutine

MACRO DELETE;
1: DO \( P_0(X) \);
2: DO R(X) THEN GO TO EXIT;
Chapter 4
Regular Expressions

4.1. Regular Expressions and Regular Sets

Definition. Let $V' = \{\cup, *, \emptyset, (, )\}$ and $V$ be a finite collection of symbols not in $V'$. A regular expression is the subset of $(VUV')^*$ defined by:

1) $\emptyset$ is a regular expression,
2) $\lambda$ is a regular expression,
3) For any $a_i \in V$, $i=1, 2, \ldots, n$, $a_i$ is a regular expression,
4) If $\alpha$ and $\beta$ are regular expressions, then so are the following:
   a) $(\alpha \cup \beta)$ (union)
   b) $(\alpha \beta)$ (catenation)
   c) $\alpha^*$ (catenation closure)
We shall refer to the three operations in regular expressions as *regular operations*. If we adapt the precedence rule for the regular operations as follows: catenation before union, and catenation closure before both union and catenation, then we may delete the use of parenthesis except when necessary.

The meaning or interpretation of a regular expression \( \gamma \) is that it denotes a set of words or a language \( L(\gamma) \) over an alphabet \( V \) according to the following conventions:

1) \( L(\emptyset) = \emptyset \),
2) \( L(\lambda) = \{\lambda\} \),
3) \( L(a_i) = \{a_i\}, \quad i=1,2,...,n \),
4) For regular expressions \( \alpha \) and \( \beta \),
   a) \( L(\alpha \cup \beta) = L(\alpha) \cup L(\beta) \)
   b) \( L(\alpha \beta) = L(\alpha)L(\beta) \)
   c) \( L(\alpha^*) = L(\alpha)^* \)

Such sets are called *regular sets*.

Example. \( (aa)^* \) is a regular expression over \( V=\{a\} \). The symbol \( a \) is a regular expression by (3) of the definition. \( (aa) \) is a regular expression by (4b) of the definition. If we let \( \alpha=(aa) \), then \( \alpha^*=(aa)^* \) is a regular
4.2. Finite State Machines and Finite Deterministic Automata

Definition. A machine is a finite state machine (FSM) if it consists of an input set, an output set and a memory set with one element each, say x, y and z respectively. The FSM operates on the relational structure \((W([a_1, a_2, \ldots, a_n]); L, B_1, B_2, \ldots, B_n; d)\) where \(W([a_1, a_2, \ldots, a_n])\) is the set of all words over the alphabet \(V = \{a_1, a_2, \ldots, a_n\}\); \(L\) is the set consisting of the empty word only; \(B_i\) is the set of words beginning with \(a_i\), \(i = 1, 2, \ldots, n\); \(d\) deletes the first letter, if any, of a word.

The algorithmic basis \(B=(\Sigma, \theta)\) for the FSM consists of the formulas \(\Sigma = \{B_1(z), B_2(z), \ldots, B_n(z)\}\) and the operations \(\theta = \{z := x; z := d(z); y := z\}\). The input program is

1: DO \(Z := X\) THEN GO TO EXIT;

The output program is
1: IF L(Z) THEN GO TO EXIT ELSE GO TO j;

The input program loads the memory. The FSM then tests the first character in z, branches accordingly and deletes the first character to make the next character accessible. The label of the instruction to be executed next is called the state of the FSM. The output program is accomplished if the program terminates and the memory is empty at termination.

For every program \( \pi \) for FSM, we single out the following items:
- \( Q \), which is the set of labels in \( \pi \),
- \( Q' \), which is the set of exits in \( \pi \),
- \( q_0 \), which is the entrance of \( \pi \).

In addition, we define the partial operations \( \tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_n \) on \( Q \) as follows: Let \( i \in Q \) and \( 1 \leq p \leq n \). Consider a word that begins with \( a_p \) and start the execution of \( \pi \) with \( i \) as the entrance label. Then we can find the first operational instruction of the form:

\[ k: \text{DO Z := D(Z) THEN GO TO j;} \]

If such an instruction is reached, we set \( \tilde{a}_p(i) = j \), otherwise \( \tilde{a}_p \) is undefined at \( i \).
This leads us to the definition of a finite
deterministic automaton.

Definition. A finite deterministic automaton (FDA)
over an alphabet $V$ is a quintuplet of the form
$F=(Q,W,q_0,Q',P_f)$ where $Q$ is a finite set of states; $W$ is
the finite input alphabet, $V=WUQ$, $Q$ and $W$ are disjoint;
$q_0$ is the start state, $q_0 \in Q$; $Q'$ is a finite set of
designated states, $Q' \subseteq Q$; $P_f$ is the set of productions of
the form $q_i a_k \rightarrow q_j$, $q_i, q_j \in Q$, $a_k \in V_T$.

Given a word $w_1, w_2, \ldots, w_m$, the letter $w_1$ is the
first scanned in the state $q_{s_0}$, (tested by $B_i$, $i=1,2,\ldots,
n$). It may delete $w_1$ (by the $d$ operation) and moves to
the next state $q_{s_1}$, scans $w_2$ in state $q_{s_1}$ and so on until
it reaches the rightmost letter in the word. The state
$q_{s_i}$ and the next scanned letter $w_{s_i}$ determines the next
state $q_{s_{i+1}}$. If the final state $q_{s_k}$ is in $Q'$, then the
word is accepted by the automaton; otherwise it is
rejected. The set of words accepted by an FDA
constitutes the language accepted by the automaton, which
we shall denote by $L(FDA)=\{\alpha \in W; q_0 \alpha \rightarrow^{*} q_1, q_1 \in Q'\}$
4.3. Regular Language

Definition. A grammar $G=(V_N, V_T, S, P)$ is a **regular grammar** if each production in $P$ is of one of the two forms $X-Ya$ or $X-a$ where $X, Y \in V_N$ and $a \in V_T$. The language generated by a regular grammar is referred to as a **regular language**.

Lemma. If $L_1$ and $L_2$ are regular languages on some alphabet $A$, then so are $L_1 \cup L_2$, $L_1L_2$ and $L_1^*$.

Proof. Let $G_1=(V_{N1}, V_{T1}, S_1, P_1)$ and $G_2=(V_{N2}, V_{T2}, S_2, P_2)$ where $V_{N1}$ and $V_{N2}$ are disjoint sets and $V_{T1}, V_{T2} \subseteq A$.

To show that $L_1 \cup L_2$ is a regular language, we introduce a new non-terminal $S$ and we define $G=(V_{N1} \cup V_{N2}, V_{T1} \cup V_{T2}, S, P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\})$. Clearly, $L(G)=L_1 \cup L_2$ is a regular language.

To show that $L_1L_2$ is a regular language, we replace each production of the form $X-a$ in $P_2$ by $X-S_1a$ and call the resulting set of productions $P'_1$. Define $G=(V_{N1} \cup V_{N2}, V_{T1} \cup V_{T2}, S, P_1 \cup P_2)$. Clearly, $L(G)=L_1L_2$ is a regular language.

To show that $L_1^*$ is a regular language, we add for
each production of the form X-α the production X=S₁α, and the additional production S₁-λ. Hence L₁* is a regular language.

Lemma. The following are equivalent:

i) L is acceptable by an FDA.

ii) L is a regular language.

Proof. A language accepted by an FDA is the union of all languages of the form \{α∈W: q₀α→*q₁\}, q₁∈Q'. And each such language is the language generated by the following regular grammar: V₉ is the set of states of the FDA; V₉ is the set of letters that composes the words acceptable by the FDA; S=q₁; P is the set of productions where for each production qᵢaⱼ→qₖ of the FDA, P has the production qₖ→qᵢaⱼ, and the additional production q₀→λ. Since the union of two regular languages is also a regular language, then L(FDA) is a regular language. Hence (i) implies (ii).

Consider a language L that is generated by a regular grammar G=(V₉,V₉,S,P). By definition, P may consists of the productions of the forms:

\[ X→Ya, \quad X,Y∈V₈, \quad α∈V₉⁺, \quad (4.3-1) \]

\[ Y→α, \quad Y∈V₈, \quad α∈V₉⁺, \quad (4.3-2) \]
\[ Y' - Y, \quad Y, Y' \in V'_N, \quad (4.3-3) \]

where \( \alpha = a_1 a_2 \ldots a_k, \quad a_i \in V_T, \quad k \geq 1 \). We wish to construct an equivalent regular grammar \( G' = (V'_N, V_T, S, P') \) whose productions are one of the following forms:

\[ Y \to Xa, \quad X, Y, \in V'_N, \quad a \in V_T, \quad (4.3-4) \]

\[ Y \to \lambda, \quad Y \in V'_N, \quad (4.3-5) \]

To eliminate the productions of the form (4.3-1), we introduce for each such production new non-terminals \( X_1, X_2, \ldots \) and \( X_{k-1} \) and replace it by the following productions:

\[
X \to X_{k-1} a_k \\
X_{k-1} \to X_{k-2} a_{k-1} \\
\vdots \\
X_2 \to X_1 a_2 \\
X_1 \to Y a_1
\]

which are of the form (4.3-4). To eliminate the productions of the form (4.3-2), we introduce for each such production new non-terminals \( Y_1, Y_2, \ldots \) and \( Y_k \) and replace it by the following productions:

\[
Y \to Y_k a_k \\
Y_k \to Y_{k-1} a_{k-1} \\
\vdots \\
Y_2 \to Y_1 a_1
\]
which are of the forms (4.3-4) and (4.3-5). To eliminate the productions of the form (4.3-3), for each such production of the form (4.3-4), we add the production $Y' \rightarrow Xa$, and for each production of the form (4.3-5), we add the production $Y' \rightarrow \lambda$. We now define an FDA $F=(Q,W,q_0,Q',P_f)$ such that $L(FDA)$ is evidently the same as $L(G')$. Let $Q$ be the set of all subsets of $V^*_N$ including the empty set; $W=V^*_T$; $q_0$ is the subset of $V^*_N$ consisting of all the $Y$'s such that the productions $Y \rightarrow \lambda$ are in $P'$; $Q'$ is the set of all subsets of $V^*_N$ which contain $S$; $P_f$ is the set of productions of the form $qa^*-q_1$ where $q_1=\{Y \in V^*_N : \text{there is a production } Y \rightarrow Xa \text{ in } P' \text{ for some } X \in Q \}$. To verify that $L(G')=L(FDA)$, let

$S \rightarrow Y_k a_k \rightarrow Y_{k-1} a_{k-1} a_k \rightarrow \ldots \rightarrow Y_1 a_1 a_2 \ldots a_k$ (4.3-6)

be a derivation according to $G'$ of a word $a_1 a_2 \ldots a_k$ in $L(G')$. Then

$q_0 a_1 a_2 \ldots a_k \rightarrow q_1 a_2 a_3 \ldots a_k \rightarrow \ldots \rightarrow q_{k-1} a_k \rightarrow q_k$ (4.3-7)

is the trace of the acceptance of the word $a_1 a_2 \ldots a_k$ by the FDA $F$, with $Y_1 \in q_0$, $Y_2 \in q_1$, $\ldots$, $Y_k \in q_{k-1}$, $S \in q_k$.

Conversely, to verify that $L(FDA)=L(G')$, let (4.3-7) be the trace of the acceptance of some word $a_1 a_2 \ldots a_k$ by $F$. 

55
Since \( q_k \in Q' \), then \( S \in q_k \); since \( q_{k-1} a_k q_k \) is a production in \( P_f \), we have that for some \( Y_k \in q_{k-1} \), there is a production \( S \rightarrow Y_k a_k \) in \( P' \); and so on, finally showing that (4.3-6) is a derivation of \( a_1 a_2 \ldots a_k \) in \( G' \). Hence \( L(G') = L(\text{FDA}) \), and (ii) implies (i).

Theorem. A language is a regular set if and only if it is a regular language.

Proof. Assume that the language \( L \) is a regular set over the alphabet \( \{a_1, a_2, \ldots, a_n\} \). By definition, this means that \( L \) may be obtained by a finite number of applications of the regular operations on the languages \( \emptyset, \{a_1\}, \{a_2\}, \ldots \) and \( \{a_n\} \). Since these languages are generated by grammars consisting of the productions of the form \( S \rightarrow \lambda \) or \( S \rightarrow a_i \), for \( i = 1, 2, \ldots, n \), clearly they are regular languages. By a previous lemma, \( L \) is a regular language.

Conversely, assume that \( L \) is a regular language with the grammar \( G=(V_N, V_T, S, P) \). We now proceed with an induction on the size of \( V_N \). For \( m = 1 \), \( V_N \) consists of \( S \) and of exactly one other non-terminal. Say, \( V_N = \{S, X_1\} \) and \( P \) consisted of the following productions:

\[
S \rightarrow \lambda
\]
S \rightarrow a_k, \ a_k \in A_1 \subseteq V_T,

S \rightarrow X_1a_k, \ a_k \in A_2 \subseteq V_T,

X_1 \rightarrow a_k, \ a_k \in A_3 \subseteq V_T,

X_1 \rightarrow X_1a_k, \ a_k \in A_4 \subseteq V_T.

L(G,X_1) is denoted by A_3A_4^* and L(G,S) is denoted by

\emptyset \cup A_1 \cup A_2A_4^*A_2. \ A_1, A_2, A_3 and A_4 are finite sets,

therefore, L(G) = L(G,S) is a regular set. We assume that

L(G) is a regular set for m=n, that is \( V_N \) consists of S

and of exactly n other non-terminals, say,

\( V_N = \{S,X_1,X_2,\ldots,X_n\} \). Consider m=n+1, where say,

\( V_N = \{S,X_1,X_2,\ldots,X_n,X_{n+1}\} \) and \( P \) consisted of the following

productions:

\begin{align*}
S & \rightarrow \lambda, \quad (4.3-8) \\
S & \rightarrow a_k, \quad (4.3-9) \\
S & \rightarrow X_i a_k, \ i \neq n+1, \quad (4.3-10) \\
X_i & \rightarrow a_k, \ i \neq n+1, \quad (4.3-11) \\
X_i & \rightarrow X_j a_k, \ i,j \neq n+1, \quad (4.3-12) \\
S & \rightarrow X_{n+1}a_k \quad (4.3-13) \\
X_{n+1} & \rightarrow a_k \quad (4.3-14) \\
X_{n+1} & \rightarrow X_j a_k, \ j \neq n+1, \quad (4.3-15) \\
X_i & \rightarrow X_{n+1}a_k, \ i \neq n+1, \quad (4.3-16) \\
X_{n+1} & \rightarrow X_{n+1}a_k. \quad (4.3-17)
\end{align*}

There are four subsets of L(G) that can be derived from

this set of productions. The first, which we shall
denote by $L_1$, has grammar $G_1$ where $V_{N_1} = V_N - \{X_{n+1}\}$ and $P$ consists of the productions of the forms (4.3-8), (4.3-9), (4.3-10), (4.3-11) and (4.3-12) only. Since the size of $V_{N_1}$ is $n$, clearly, $L_1$ is a regular set. The second, which we shall denote by $L_2$, has grammar $G_2$ where $V_{N_2} = (V_N - \{S, X_{n+1}\}) \cup \{S_2\}$ and $P$ consists of the productions of the forms (4.3-10), (4.3-11), $S_2 \rightarrow a_k$ in place of (4.3-13) and $X_i \rightarrow a_k$ in place of (4.3-16). Since the size of $V_{N_2}$ is $n$, clearly, $L_2$ is a regular set. The third, which we shall denote by $L_3$, has grammar $G_3$ where $V_{N_3} = (V_N - \{S, X_{n+1}\}) \cup \{S_3\}$ and $P$ consists of the productions of the forms (4.3-11), (4.3-12), $S_3 \rightarrow a_k$ in place of (4.3-14) and $S_3 \rightarrow X_j a_k$ in place of (4.3-15). Since the size of $V_{N_3}$ is $n$, clearly, $L_3$ is a regular set. The fourth, which we shall denote by $L_4$, has grammar $G_4$ where $V_{N_4} = (V_N - \{S, X_{n+1}\}) \cup \{S_4\}$ and $P$ consists of the productions of the forms (4.3-11), $S_4 \rightarrow a_k$ in place of (4.3-14), $S_4 \rightarrow X_j a_k$ in place of (4.3-15), $X_i \rightarrow a_k$ in place of (4.3-16), and $S_4 \rightarrow a_k$ also in place of (4.3-17). Since the size of $V_{N_4}$ is $n$, clearly, $L_4$ is a regular set. For every derivation in $L(G)$, if it uses (4.3-8), then we have the regular set $\emptyset$. If $X_{n+1}$ does not arise, then we have $L_1$. Otherwise, we have the derivation of the form

$$S \rightarrow \cdots X_{n+1} \rightarrow \cdots X_{n+1} \beta_1 \rightarrow \cdots X_{n+1} \beta_2 \beta_1 \rightarrow \cdots X_{n+1} \beta_j \beta_{j-1} \cdots$$
\[ \beta_2 \beta_1 \alpha^* \gamma \beta_j \beta_{j-1} \ldots \beta_2 \beta_1 \alpha \]
which is equivalent to \( L_2 L_3 L_4 \ldots L_j L_4 = L_2 L_3^* L_4 \). Therefore, 
\( L = L(G) = \emptyset U L_1 U L_2 L_3^* L_4 \) is a regular set.

Corollary. A regular set is acceptable by an FDA.

Proof. By a previous lemma, a regular language is acceptable by an FDA. By a previous theorem, a regular set is a regular language. Therefore, a regular set is acceptable by an FDA.
Chapter 5
Simple Precedence Grammars

5.1. Precedence Grammars

Definition. Let \( G=(V_N,V_T,S,P) \) be a grammar. Let \( \alpha=\delta xy \). Then \( x \) is called a simple phrase of the string \( \alpha \) for the non-terminal \( X \) if there exists a derivation \( S \Rightarrow^* \delta Xy \) such that there is a production \( X \rightarrow x \).

Definition. A handle of a string is the simple phrase resulting from the last step of a leftmost derivation.

Definition. Let \( G \) be a grammar and \( X \) a non-terminal in \( G \). Then \( a \in \text{FIRST}(X) \) if and only if there is a production \( X \rightarrow a\beta \). If there is a sequence of productions \( X \rightarrow X_1\beta_1, X_1 \rightarrow X_2\beta_2, \ldots, X_n \rightarrow a\beta, n \geq 0 \), then
a \in \text{FIRST}^+(X)$. Furthermore, \(a \in \text{FIRST}^+(X)\) if and only if \(a \in \text{FIRST}^+(X)\) or \(a = X\). Similarly, \(d \in \text{LAST}(X)\) if and only if there is a production \(X \rightarrow \beta d\). If there is a sequence of productions \(X \rightarrow \beta_1 X_1, X_1 \rightarrow \beta_2 X_2, \ldots, X_n \rightarrow \beta d, n \geq 0\), then \(d \in \text{LAST}^+(X)\). Also, \(d \in \text{LAST}^+(X)\) if and only if \(d \in \text{LAST}^+(X)\) or \(d = X\).

Given a word \(a = \delta \gamma \) where \(x\) is a handle, we say that we reduce the handle into \(\beta = \delta \gamma \) by production \(P_i = X \rightarrow X\) where \(P_i \in P\) and \(G = (V_N, V_T, S, P)\).

**Remark.** Observe that reduction is the reverse process of derivation. Hence, every letter becomes part of a handle at some point in the reduction of a string. And since we are considering leftmost derivations, then we are also concerned with rightmost reduction.

**Definition.** Given a grammar \(G = (V_N, V_T, S, P)\). The precedence relations between the letters in the vocabulary \(V = V_N \cup V_T\) are defined as follows:

1) \(x \preceq y\) if and only if there is a production \(Z \rightarrow \delta xy\) in \(G\).
2) \(x \succeq y\) if and only if there is a production \(Z \rightarrow \delta Uy\) in \(G\) such that \(x \in \text{LAST}^+(U)\).
3) \(x \succeq y\) if and only if \(x\) is a terminal and there is a
production $Z \rightarrow \delta U W Y$ such that $x \in \text{LAST}^*(U)$ and $y \in \text{FIRST}^+(W)$.

Note that in the presentation of precedence relations, we shall assume that all the productions of a given grammar are accessible; that is, for any non-terminal $Z$ there exists a leftmost derivation $S \rightarrow^* a Z B \rightarrow^* \delta, \delta \in V_T^*$.

Theorem. The precedence relation $x \triangleright y$ holds if and only if the substring $xy$ belongs to the handle of some string.

Proof. Suppose $x \triangleright y$. By definition, there exists a production $Z \rightarrow \delta x y y$. There also exists a leftmost derivation $S \rightarrow^* a Z B \rightarrow^* \delta x y y \beta$, for some $\alpha$ and $\beta$ where $S$ is the starting symbol. Clearly, $x y$ is in the handle of $\alpha \delta x y y \beta$.

Conversely, if $x y$ is part of the handle of some $\mu$, then by definition there exists a leftmost derivation $S \rightarrow^* a Z B$ such that $Z \rightarrow \delta x y y$ and $\mu = \alpha \delta x y y \beta$.

Theorem. The precedence relation $x \ll y$ holds if and only if there exists a string $\alpha x y \beta$ where $y$ is the head of the handle.
Proof. Suppose \( x \prec y \). By definition \( x \) is a terminal and there is a production \( Z \rightarrow \delta U W \gamma \) such that \( x \in \text{LAST}^*(U) \) and \( y \in \text{FIRST}^+(W) \). This means that there exists a leftmost derivation \( S \rightarrow \alpha_1 \beta_1 \cdots \beta_1 \delta U W \gamma \beta_1 \). Also \( x \in \text{LAST}^*(U) \) means there exists a leftmost derivation \( U \rightarrow \theta x \); and \( y \in \text{FIRST}^+(W) \) means there exists a leftmost derivation \( W \rightarrow \gamma \sigma \). Hence, \( S \rightarrow \alpha_1 \beta_1 \delta \theta x W \gamma \beta_1 \rightarrow^+ \alpha_1 \beta_1 \delta \theta x y \gamma \gamma \beta_1 = \alpha x y \beta \). Now, \( y \) is the head of the handle and \( x \) is not in the handle but precedes the handle.

Conversely, suppose there exists a string \( \alpha x y \beta \) where \( y \) is the head of the handle and is preceded by \( x \). This means that for some \( W \), there exists a leftmost derivation \( W \rightarrow^+ \gamma \sigma \). Perform reduction repeatedly until \( x \) becomes part of the handle. This gives rise to the string \( \alpha x W \beta_1 \).

1) If both \( x \) and \( W \) are in the handle then there must exists a production \( Z \rightarrow \delta x W \gamma \). Hence, \( Z \rightarrow \delta U W \gamma \) where \( U = x \) and \( W \rightarrow^+ \gamma \sigma \).

2) If \( x \) is the tail of the handle, then this means that for some \( U \), there exists a leftmost derivation \( U \rightarrow^+ \theta x \). Perform reduction until \( W \) is in the handle. This gives rise to the string \( \alpha_1 U W \beta_1 \). \( U \) must also be in the handle. If \( W \) were the head of the handle, then being concerned with rightmost reduction, everything
to the left of W must be terminals, but U€V_N. Thus, there exists a production Z-δUWy where U →⁺θx and W →⁺yσ.

Thus, there exists a production Z-δUWy where x€LAST⁺(U) and y€FIRST⁺(W). Hence, x<y.

Theorem. The precedence relation x>y holds if and only if there exists a string αxyβ where x is the tail of the handle.

Proof. Suppose x>y. By definition, there is a production Z-δUyy such that x€LAST⁺(U). This means that there exists a leftmost derivation S-α₁Zβ₁α₁δUyyβ₁. Also, x€LAST⁺(U) means there exists a leftmost derivation U→⁺θx. Hence, S-α₁δθxyγβ₁=αxyβ. Now, x is the tail of the handle and y is not in the handle but follows the handle.

Conversely, suppose there exists a string αxyβ where x is the tail of the handle and is followed by y. This means that for some U, there exists a leftmost derivation U→⁺θx. Perform reduction until y is in the handle. This gives rise to the string α₁Uyβ. U must also be in the handle. If y were the head of the handle, then being concerned with rightmost reduction, everything
to the left of $y$ must be terminals, but $U \in \mathcal{V}_N$. Thus, there exists a production $Z \rightarrow \delta U \gamma$ where $x \in \text{LAST}^+(U)$. Hence $x \preceq y$.

Definition. A grammar $G=(\mathcal{V}_N, \mathcal{V}_T, S, P)$ is called a simple precedence grammar or a (1,1) precedence grammar if

1) at most one relation holds between any two letters of the vocabulary,
2) no two productions have the same right side, and
3) no production of the form $Z \rightarrow \lambda$ exists.

The (1,1) refers to the fact that we use one symbol on each side of a possible handle to help decide if it is a handle or not.

A syntax tree is a diagram that describes the structure of a word by breaking it into its constituent parts. A parse of a word is the construction of a derivation and possibly a syntax tree for it.

Example. Consider the productions of the grammar $G$ which are as follows:

$Z \rightarrow E$
The syntax tree of the leftmost derivation of the word \( i + i \ast i \) which is in \( L(G) \) is as follows:

In the above example, \( E \) and \( T \) form branches of \( Z \). \( T \) on the other hand has its own branches formed by \( T \) and \( F \). The \( i \), \( + \) and \( \ast \) are referred to as leaves.

For the purpose of parsing, consider a given word as enclosed by the delimiter \( # \). Define the relation between \( # \) and the first letter in the word to be \( < \), and
Lemma. If $\delta xy \Rightarrow \delta xy$ is a step in a leftmost derivation, then if $x$ occupies positions $i$ through $j$ in $\delta xy$, then $S_{i-1} < S_i \Rightarrow S_{i+1} \Rightarrow \ldots \Rightarrow S_{j-1} \Rightarrow S_j \Rightarrow S_{j+1}$.

Proof. Since $\delta xy \Rightarrow \delta xy$, then there exists a production $X \rightarrow x$ and at some point $x$ is the handle. Since $x = S_i S_{i+1} \ldots S_j$, then $S_i \Rightarrow S_{i+1} \Rightarrow \ldots \Rightarrow S_j$. This means that $S_i$ is the head of the handle and $S_j$ is the tail of the handle. Since $S_{i-1}$ precedes the head of the handle, then $S_{i-1} < S_i$. Similarly, since $S_{j+1}$ follows the tail of the handle, then $S_j \Rightarrow S_{j+1}$.

Lemma. The handle of any string derived from a simple precedence grammar is the rightmost substring $S_i \ldots S_j$ such that

$$S_{i-1} < S_i \Rightarrow \ldots \Rightarrow S_j \Rightarrow S_{j+1} \quad (5.1-1)$$

and is therefore unique.

Proof. Suppose $u = S_k \ldots S_p$ is the handle in a leftmost derivation. Then $S_{k-1} < S_k \Rightarrow \ldots \Rightarrow S_p \Rightarrow S_{p+1}$. Suppose $v = S_i \ldots S_j$ is the rightmost substring satisfying (5.1-1). We know that at some point every letter becomes part of
the handle. Let \( S_t \) be the first letter to become part of
the handle in that leftmost derivation, \( i-1 \leq t \leq j+1 \). Then
one of the following must hold:

1) \( t = i-1 \), then \( S_t \geq S_{t+1} \) but \( S_t < S_{t+1} \) from \( (5.1-1) \).

2) \( t = j+1 \), then \( S_t < S_{t+1} \) but \( S_t \geq S_{t+1} \) from \( (5.1-1) \).

3) \( i \leq t \leq j \)

Let \( S_r \ldots S_w \), \( r \leq t \leq w \), be the handle.

a) If \( r < i \), then \( S_{r-1} \geq S_r \) but \( S_{r-1} < S_r \).

b) If \( r > i \), then \( S_{r-1} < S_r \) but \( S_{r-1} \geq S_r \).

c) If \( w < j \), then \( S_w \geq S_{w+1} \) but \( S_w < S_{w+1} \).

d) If \( w > j \), then \( S_w < S_{w+1} \) but \( S_w \geq S_{w+1} \).

Since at most one of the precedence relations may hold
between any two letters, then \( i=r \leq t \leq w=j \). Therefore, all
of \( v \) must have become part of the handle at the same
stage, that is \( v \) is a handle.

Since \( v \) is a handle at some stage, then there must
exist a production \( V \rightarrow v \) and a derivation \( \gamma V \beta \rightarrow \gamma v \beta \), \( \gamma \in \Gamma_\Sigma \).

Since \( u \) is the handle, then there must exist a production
\( U \rightarrow u \) and a derivation \( \gamma V \beta \rightarrow \alpha U \delta v \beta \rightarrow \alpha u \delta v \beta \). This means then
that \( \gamma \notin \Gamma_\Sigma \) which is a contradiction to this derivation
being leftmost.
Theorem. A simple precedence grammar is unambiguous.

Proof. Since the handle of any string derived from a simple precedence grammar is unique, this further means that at each step of the parse, the handle is uniquely determined. Since the handle may be the right side of only one production, there is only one symbol to which the handle can be reduced making each reduction unique. Hence, there is only one leftmost derivation for any word which proves that the grammar is unambiguous.

Remark. Observe that since a simple precedence grammar is unambiguous, simple phrases are either identical or disjoint.
5.2. Computation of the Precedence Relations

In the computation of the precedence relations from a given grammar, we shall use boolean matrices whose only elements are 1 (for when the relation holds true) and 0 (otherwise). Multiplication of boolean matrices is as in numerical matrix multiplication except that addition is replaced by "or" and multiplication by "and". Boolean matrix addition is also obviously associative and commutative while boolean matrix multiplication is only associative, but both operations satisfy the distributive law: $A(B+C)=AB+AC$, where $A$, $B$, and $C$ are matrices.

Specifically, choose an enumeration of the letters $S_1, S_2, \ldots, S_n$ of $V=V_T \cup V_N$.

Let the following matrices be defined:

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Relation Represented</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I$</td>
<td>identity</td>
</tr>
<tr>
<td>$EQ$</td>
<td>$\neq$ where $EQ_{ij}=1$ if and only if $S_i \neq S_j$</td>
</tr>
<tr>
<td>$LT$</td>
<td>$&lt;$ where $LT_{ij}=1$ if and only if $S_i &lt; S_j$</td>
</tr>
<tr>
<td>$GT$</td>
<td>$&gt;$ where $GT_{ij}=1$ if and only if $S_i &gt; S_j$</td>
</tr>
<tr>
<td>$FIRST$</td>
<td>$\in \text{FIRST}(X)$ where $FIRST_{ij}=1$ if and only if $S_j \in \text{FIRST}(S_i)$</td>
</tr>
</tbody>
</table>
Matrix Relation Represented

LAST $\in \text{LAST}(X)$ where $\text{LAST}_{ij}=1$ if and only if $S_i \in \text{LAST}(S_j)$

Definition. Let $M$ be an $n$ by $n$ boolean matrix representing a relation $R$ over a vocabulary $V$ of $n$ letters. Then the matrix $M^+$ is defined as follows:

$$M^+=M+M^2+M^3+\ldots+M^n.$$  

And $M^+=(I+M^+)$.

Remark. $M^+$ is called the transitive closure of $M$.

We shall prove that $GT=({\text{LAST}^+})^T(EQ)$ and $LT=({\text{LAST}^*})^T(EQ)(\text{FIRST}^+)$.

Definition. Given two relations $P$ and $Q$. For some $x$ and $y$, $xPQy$ if and only if there exists a $z$ such that $xPz$ and $zQy$. $PQ$ is called the product of $P$ and $Q$.

Lemma. The product of two relations over the same vocabulary is given by the product of the boolean matrix representing those relations.

Proof. Let $S_i \in V$, $i=1, \ldots, n$. Let $D=(d_{ij})$ be the
product of two matrices \( B=(b_{ij}) \) and \( C=(c_{ij}) \) which represent the relations \( P \) and \( Q \) respectively; that is, \( D=BC \). If \( d_{ij}=1 \), then by definition and for some \( k \), \( b_{ik}=1 \) and \( c_{kj}=1 \). Thus, \( S_i PS_k \) and \( S_k QS_j \) which means that \( (S_i, S_j) \) is in the product \( PQ \).

Conversely, if \( (S_i, S_j) \) is in \( PQ \), then there exists a \( k \) such that \( S_i PS_k \) and \( S_k QS_j \) and \( b_{ik}=1 \) and \( c_{kj}=1 \). Since \( D=BC \), then \( d_{ij}=1 \) which means that \( (S_i, S_j) \) is in \( D \) and \( D \) represents the relation \( PQ \).

Theorem. \( GT=(LAST^+)^T(EQ) \).

Proof. Recall that for any two letters \( x \) and \( y \), \( x\succ y \) if and only if there is a production \( Z\rightarrow_\alpha Uy\beta \) such that \( x\in LAST^+(U) \). This also means that \( U\nsucc y \).

Let \( g \) represent the relation \( \in LAST^+(X) \) and \( M=(LAST^+)^T(EQ) \). If for some \( U \), \( x \) and \( y \), \( U\nsucc y \) and \( U\nsucc x \), then \( xg^{-1}U \). Also, \( (x,y) \) is in the product of \( g^{-1} \) and \( \nsucc \), which by the previous lemma is represented by the matrix \( M \). But since \( M \) satisfies the definition of the relation \( \nsucc \), then \( M=GT \).
Theorem. \( LT = (\text{LAST}^*)^T (\text{EQ}) (\text{FIRST}^+) \).

Proof. Recall that for any two letters \( a \) and \( y \), \( x \preceq y \) if and only if \( x \) is a terminal and there is a production \( Z \rightarrow \alpha U W \beta \) such that \( x \in \text{LAST}^*(U) \) and \( y \in \text{FIRST}^+(W) \). This also means that \( U = W \).

Let \( f \) represent the relation \( \epsilon \text{FIRST}^+(X) \) and \( g \)
represent the relation \( \epsilon \text{LAST}^+(X) \). Let \( I \) be the identity
matrix (that is, if \( I = (a_{ij}) \), then \( a_{jj} = 1 \)) and
\( M = (\text{LAST}^*)^T (\text{EQ}) (\text{FIRST}^+) \). If for some \( U \), \( W \) and \( y \), \( U \preceq W \) and \( W \epsilon y \), then \((U, y)\) is in the product of \( \epsilon \) and \( f \), which we
shall represent by the matrix \( B \), that is, \( B = (\text{EQ}) (\text{FIRST}^+) \).

Let \( A = I + \text{LAST}^+ \). For some \( x \), \( x \epsilon g \) and \( U \epsilon x \). Hence, if
\( U \epsilon x \) (including \( U = x \)), then \( x \epsilon g^{-1} U \).

Let \( h \) be the relation represented by \( B \), then \( U \epsilon h y \).
If \( x \epsilon g^{-1} U \) and \( U \epsilon h y \), then \((x, y)\) is in the product of \( g^{-1} \) and
\( h \), which is \( AB \) and which is \( M \). But since \( M \) satisfies the
definition of the relation \( \prec \), then \( M = LT \).
To illustrate the computation of the precedence relations, consider the following grammar \( G=(V_N, V_T, S, P) \):

- \( V_N = \{Z, A, S\} \)
- \( V_T = \{a, b, (, )\} \)
- \( P_1 = Z \rightarrow (AS) \)
- \( P_2 = S \rightarrow (b) \)
- \( P_3 = A \rightarrow (SaA) \)
- \( P_4 = A \rightarrow (a) \)

First construct the matrices FIRST\(^+\) and LAST\(^+\):

<table>
<thead>
<tr>
<th>FIRST(^+)</th>
<th>Z</th>
<th>A</th>
<th>S</th>
<th>(</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

74
Construct \( \text{EQ} \) by going through the right side of each production and setting \( X=Y \) for each \( X \) and \( Y \) such that \( Z-aXY\beta \).

<table>
<thead>
<tr>
<th>( \text{LAST}^+ )</th>
<th>( Z )</th>
<th>( A )</th>
<th>( S )</th>
<th>( ( )</th>
<th>( ) )</th>
<th>( b )</th>
<th>( a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Z )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( A )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( S )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( ( )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( ) )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( b )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( a )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Construct \( \text{GT} \) by taking the product of \( (\text{LAST}^+)^T \) and
EQ, that is, \( GT=(LAST^*)^T(EQ) \).

<table>
<thead>
<tr>
<th>GT</th>
<th>Z</th>
<th>A</th>
<th>S</th>
<th>(</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Construct LT by taking the product of \((LAST^*)^T\), EQ and \(FIRST^+\), that is, \( LT=(LAST^*)^T(EQ)(FIRST^+) \).

<table>
<thead>
<tr>
<th>LT</th>
<th>Z</th>
<th>A</th>
<th>S</th>
<th>(</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

76
To construct the precedence matrix $PM$, take the matrices $EQ$, $LT$ and $GT$ and enter $\neq$ for the entry corresponding to every non-zero entry in $EQ$, $<$ for the entry corresponding to every non-zero entry in $LT$, $>$ for the entry corresponding to every non-zero entry in $GT$ and $e$ for error otherwise.

<table>
<thead>
<tr>
<th>$PM$</th>
<th>$Z$</th>
<th>$A$</th>
<th>$S$</th>
<th>$($</th>
<th>$)$</th>
<th>$b$</th>
<th>$a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Z$</td>
<td>$e$</td>
<td>$e$</td>
<td>$e$</td>
<td>$e$</td>
<td>$e$</td>
<td>$e$</td>
<td>$e$</td>
</tr>
<tr>
<td>$A$</td>
<td>$e$</td>
<td>$e$</td>
<td>$\neq$</td>
<td>$&lt;$</td>
<td>$&lt;$</td>
<td>$&lt;$</td>
<td>$e$</td>
</tr>
<tr>
<td>$S$</td>
<td>$e$</td>
<td>$e$</td>
<td>$e$</td>
<td>$e$</td>
<td>$\neq$</td>
<td>$e$</td>
<td>$\neq$</td>
</tr>
<tr>
<td>$( $</td>
<td>$e$</td>
<td>$\neq$</td>
<td>$\neq$</td>
<td>$&lt;$</td>
<td>$&lt;$</td>
<td>$&lt;$</td>
<td>$e$</td>
</tr>
<tr>
<td>$) $</td>
<td>$e$</td>
<td>$e$</td>
<td>$&gt;$</td>
<td>$&lt;$</td>
<td>$&lt;$</td>
<td>$&lt;$</td>
<td>$e$</td>
</tr>
<tr>
<td>$b$</td>
<td>$e$</td>
<td>$e$</td>
<td>$e$</td>
<td>$\neq$</td>
<td>$e$</td>
<td>$e$</td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>$e$</td>
<td>$\neq$</td>
<td>$e$</td>
<td>$&lt;$$&lt;$</td>
<td>$&lt;$</td>
<td>$e$</td>
<td>$e$</td>
</tr>
</tbody>
</table>

This also proves that the grammar is a precedence grammar since all conditions of the definition are satisfied.

Using the matrix $PM$, we indicate the relation of each letter with the next letter for the word

$#( ((b) a(a)) (b)) # \text{ to be } #\neq (\neq (\neq \neq) \neq \neq (\neq a a) \neq) \neq (\neq b b) \neq) \neq. #
We parse the word to determine if the word is a valid word for the grammar. Since we are concerned with leftmost derivations, we concern ourselves with rightmost parsing. Look for the handle by locating the rightmost < and the leftmost > to its right. By definition of a handle, this gives the rightmost simple phrase which we then reduce by substituting the lefthand side of the appropriate production and the relations according to PM.

Production

Projection

\[ \#<((v>b)avl<v(a)>)<(v>b)>\]  

\[ \#<((v>b)avl<v(a)>)vSv\]  

\[ \#<((v>b)avl<v(a)>)vSv\]  

\[ \#<((v>Savl<v(A)>)Sv)\]  

\[ \#<((v(A)Sv)\]  

And we find that the word is valid and is therefore in \( L(G) \).
Definition. Let \( G=(V_N,V_T,S,P) \) be a grammar. If there exists a production \( Z \rightarrow \alpha Z \beta \), then the grammar is said to be recursive in \( Z \). If \( \alpha = \lambda \), then \( G \) is left recursive. If \( \beta = \lambda \), then \( G \) is right recursive.

In the computation of precedence relations, a problem may arise with left recursive grammars. Suppose there is a production \( Z \rightarrow Z \beta \). If another production \( Y \rightarrow \alpha X Z \beta \) exists, then the relation \( X \preceq Z \) and \( X \prec Z \) exist which is a violation of condition one of the definition of precedence grammar. This conflict is resolved by introducing another non-terminal \( W \) and an intermediate production by changing \( Y \rightarrow \alpha X Z \beta \) to \( Y \rightarrow \alpha X W \beta \) and adding \( W \rightarrow \). This yields \( X \preceq W \) and \( X \prec Z \).

Consider the following example:

\[
\begin{align*}
P_1 &= E \rightarrow E + T \\
P_2 &= E \rightarrow T \\
P_3 &= T \rightarrow T \ast F \\
P_4 &= T \rightarrow F \\
P_5 &= F \rightarrow (E) \\
P_6 &= F \rightarrow i
\end{align*}
\]

Since no two productions have the same right side, condition two of the definition of a precedence grammar
is satisfied. Also, since there are no null productions, condition three of the definition is satisfied. Note, however, that the existence of both $P_1$ and $P_5$ and both $P_3$ and $P_1$ violates condition one as shown by the resulting PM matrix.

<table>
<thead>
<tr>
<th>PM</th>
<th>E</th>
<th>T</th>
<th>F</th>
<th>*</th>
<th>+</th>
<th>(</th>
<th>)</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>T</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>F</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>*</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>+</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>(</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>)</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>i</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
</tbody>
</table>

To remedy the conflict, introduce the following changes:

- $P_1 = E - E + T$
- $P_2 = E - T$
- $P_3 = T - T'$
- $P_4 = T' - T' * F$
- $P_5 = T' - F$
\[ P_6 = F \rightarrow (E') \]
\[ P_7 = F \rightarrow i \]
\[ P_8 = E' \rightarrow E \]

These changes lead to the following PM matrix.

<table>
<thead>
<tr>
<th>PM</th>
<th>E</th>
<th>E'</th>
<th>T</th>
<th>T'</th>
<th>F</th>
<th>*</th>
<th>+</th>
<th>( )</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>E'</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>F</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>*</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>+</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>(</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>)</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>i</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
</tbody>
</table>

Consider the word \#i*(i+i)#.

Production
Used

\[ \#i*i<(i+i)\# \]
\[ \uparrow \]

\[ \#i*i<(i+i+\#F)\# \]
\[ \uparrow \]
\[ T = \text{Production} \]

\[ P_3 \]

\[ T = \text{Used} \]

\[ P_2 \]

\[ E' \]

\[ P_8 \]
Bibliography


Vita

Carolina Paciencia Salas was born on February 15, 1949 in Manila, Philippines to Conrado Peredo Salas and Josefa Donato Paciencia. She comes from a family of three girls of which she is the eldest. She received her Bachelor of Arts in Mathematics from St. Scholatica's College in Manila in 1970. She was then awarded a graduate assistantship by the Mathematics Department of the Ateneo de Manila University to pursue a masters degree in mathematics. After a semester, she was employed by the Ateneo as an instructor of the Mathematics Department. At the same time, she changed her field of study from mathematics to statistics. She received her Master of Science in Statistics from the Ateneo de Manila University in 1976, after which she immediately went to the United States of America to pursue a masters degree in computer science at the
Lehigh University in Bethlehem, Pennsylvania where she was awarded a teaching assistantship. She received her Master of Science in Computer Science from Lehigh University in 1978.