7- Context Free Grammars

Context Free Grammar is a formal Notation for expressing Recursive Definitions of languages.

Definition: A CFG G is defined as G = (V, T, P, S) where

1. **T:** Finite set of terminals, i.e., set of symbols that form the strings of the language.
2. **V:** Finite set of variables called non-terminals or syntactic categories.
3. **S:** Start symbol defines (represents) the language. Other variables represent auxiliary classes of strings used to help define language.
4. **P:** Finite set of productions (rules) representing recursive definition of a language.

   Each production consists of

   - Variable being defined (Partially) called head of the production.
   - Production symbol $\rightarrow$
   - String of zero/more terminals and/or variables, called the body of the production.

Terminals remain unchanged.

Non-terminal (Variable) is replaced by its body.

Ex: Design Context Free Grammar to represent binary strings that are palindromes.

$$\left.\begin{array}{l} P \rightarrow \varepsilon \\ P \rightarrow 0 \\ P \rightarrow 1 \end{array}\right\} \textbf{BASIS}$$

$$\left.\begin{array}{l} P \rightarrow 0P0 \\ P \rightarrow 1P1 \end{array}\right\} \textbf{INDUCTION}$$

Ex: Design CFG to represent the strings of the type $\mathbf{a^*}$

$A \rightarrow \varepsilon$

$A \rightarrow aA$

Ex: Design CFG to represent the strings of the type $\mathbf{a^+}$

$A \rightarrow a$

$A \rightarrow aA$

Ex: Design CFG to represent the Language $L = \{a^n b^n \mid n >= 0\}$

$$\left.\begin{array}{l} A \rightarrow \varepsilon \\ A \rightarrow aAb \end{array}\right\} \quad \textbf{OR} \quad A \rightarrow \varepsilon \mid aAb$$

Ex: Design CFG to represent the Language $L = \{a^n b^n \mid n >= 1\}$

$$\left.\begin{array}{l} A \rightarrow ab \\ A \rightarrow aAb \end{array}\right\} \quad \textbf{OR} \quad A \rightarrow ab \mid aAb$$

Ex: Design CFG to represent the Language $L = \{a^m b^m c^n d^n \mid m,n >= 0\}$

$S \rightarrow AB$

$A \rightarrow \varepsilon$

$A \rightarrow aAb$

$B \rightarrow \varepsilon$

$B \rightarrow cBd$

Ex: Design CFG to represent the Language $L = \{a^m b^m c^n d^n \mid m, n >= 1\}$

$S \rightarrow AB$

$A \rightarrow ab$

$A \rightarrow aAb$

$B \rightarrow cd$

$B \rightarrow cBd$

7.1 Notational Conventions

1. Upper case letters early in the alphabet such as A, B, C represent Non terminals or Variables.
2. Lower case letters early in the alphabet such as a, b, c represent terminal symbols.
3. Upper case letters late in the alphabet such as X, Y, Z represent Grammar symbols. A grammar symbol is either a terminal or a Non terminal.
4. Lower case letters late in the alphabet such as w, x, y, z represent strings of terminal symbols only.
5. Normally S (Upper case) represents the start symbol.
6. Lower case greek letters such as α(alpha), β(beta), γ (gamma) represent strings of grammar symbols. If A --> β is a production then A is a non-terminal and β is a string of the type $X_1X_2X_3. . .X_n$ where each X_i is a grammar symbol.

7.2 Inference, Derivations and Derivation (Parse) trees

We use Inference and Derivations to infer that the strings are in the language of a Variable (Non Terminal).

7.2.1 Inference (Bottom up Approach):

1. Start with the terminal string.
2. At each step find the substring that matches the right hand side of a production and replace it by the non-terminal on the left hand side of the production.
3. Repeat step 2 till we are left with only the start symbol.

<u>Inference:</u> (Example)

$S \longrightarrow AC$
$A \longrightarrow aAb \mid \varepsilon$
$C \longrightarrow cCd \mid \varepsilon$

Infer the string ***aabbcccddd*** using the above grammar.

aa**ε**bbcccddd (Use the production $A \longrightarrow \varepsilon$)
aaAbbcccddd (Use the production $A \longrightarrow aAb$)

aAbcccddd (Use the production $A \longrightarrow aAb$)

Acccddd (Insert ε)

Accc**ε**ddd (Use the production $C \longrightarrow \varepsilon$)
AcccCddd (Use the production $C \longrightarrow cCd$)
AccCdd (Use the production $C \longrightarrow cCd$)
AcCd (Use the production $C \longrightarrow cCd$)
AC (Use the production $S \longrightarrow AC$)
S.

Starting with the terminal string the start symbol is reached. Hence the string aabbcccddd is valid.

7.2.2 Derivation (Top Down Approach):

1. Start with the start symbol.
2. At each step replace a Non Terminal by the right hand side of the production.
3. Repeat step 2 till we are left with only the terminal string.

$\Rightarrow$ This symbol is used to indicate a derivation step

Example:

Given the grammar $A \rightarrow aAb \mid \varepsilon$

(1) Derive the empty string $A \Rightarrow \varepsilon$

(2) Derive the string ab

	Production Used
$A \Rightarrow aAb$	$A \rightarrow aAb$
$\Rightarrow ab$	$A \rightarrow \varepsilon$

(3) Derive the string aabb

	Production Used
$A \Rightarrow aAb$	$A \rightarrow aAb$
$\Rightarrow aaAb$	$A \rightarrow aAb$
$\Rightarrow aabb$	$A \rightarrow \varepsilon$

Derivation (Top Down Approach): There are two types of derivation.

1. Left Most Derivation (LMD)
2. Right Most Derivation (RMD)

Left Most Derivation (LMD)

1. Start with the start symbol.
2. At each step replace the Left Most Non Terminal by the right hand side of the production.
3. Repeat step 2 till we are left with only the terminal string.

Right Most Derivation (RMD)

1. Start with the start symbol.
2. At each step replace the Right Most Non Terminal by the right hand side of the production.
3. Repeat step 2 till we are left with only the terminal string.

Leftmost Derivation (Example)

$S \longrightarrow AC$
$A \longrightarrow aAb \mid \varepsilon$
$C \longrightarrow cCd \mid \varepsilon$

Derive the string aabbccdd using LMD

$S \overset{lmd}{\Longrightarrow} AC \overset{lmd}{\Longrightarrow} aAbC \overset{lmd}{\Longrightarrow} aaAbbC \overset{lmd}{\Longrightarrow} aa\varepsilon bbC \overset{lmd}{\Longrightarrow} aabbC \overset{lmd}{\Longrightarrow} aabbcCd \overset{lmd}{\Longrightarrow} aabbccCdd \overset{lmd}{\Longrightarrow} aabbcc\varepsilon dd \overset{lmd}{\Longrightarrow} aabbccdd.$

Rightmost Derivation (Example)

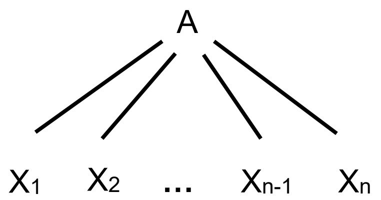
S ⟶ AC
A ⟶ aAb | ε
C ⟶ cCd | ε

Derive the string aaabbbccdd using the RMD.

$S \xRightarrow{rmd} AC \xRightarrow{rmd} AcCd \xRightarrow{rmd} AccCdd \xRightarrow{rmd} Acc\varepsilon dd \xRightarrow{rmd} Accdd \xRightarrow{rmd} aAbccdd \xRightarrow{rmd} aaAbbccdd$
$\xRightarrow{rmd} aaaAbbbccdd \xRightarrow{rmd} aaa\varepsilon bbbccdd \xRightarrow{rmd} aaabbbccdd.$

7.2.3 Derivation (Parse) trees

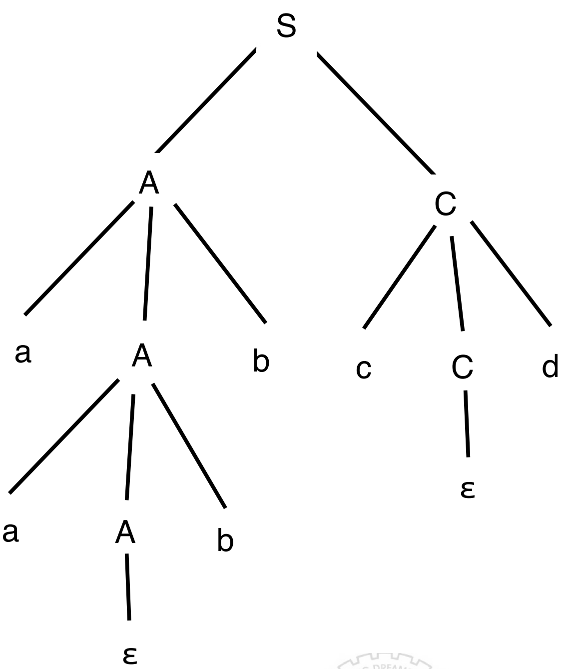
1. Start symbol is the root of the tree.
2. If A is some node in the tree and $A \longrightarrow X_1X_2X_3. . .X_n$ is a production in G, the corresponding tree looks like



3. All the interior nodes are Non terminals whereas all the leaf nodes are terminals.
4. The concatenation of leaf nodes from left to right is called the yield of the parse tree.

Parse Tree (Example)

Parse tree for the string aabbcd



Example: Design CFG to represent the Language

$L = \{ w \mid w \in \{a, b\}^* \}$

CFG: $A \to aA \mid bA \mid \varepsilon$

Example: Design CFG to represent all binary strings containing the substring 011 at least once.

CFG: $S \to A011A$

$A \to aA \mid bA \mid \varepsilon$

Example: Design CFG to represent strings of the type a*b*c*.

CFG: $S \to ABC$

$A \to aA \mid \varepsilon$

$B \to bB \mid \varepsilon$

$C \to cC \mid \varepsilon$

Example: Design CFG to represent the Language

$L = \{ a^m b^n \mid m > n \}$

CFG: $S \to AE$

$E \to aEb \mid \varepsilon$

$A \to aA \mid a$

Example: Design CFG to represent the Language

$L = \{ a^m b^n \mid m < n \}$

CFG: $S \to EB$

$E \to aEb \mid \varepsilon$

$B \to bB \mid b$

Example: Design CFG to represent the Language

$L = \{ a^m b^n \mid m \neq n \}$

CFG: $S \to AE \mid EB$

$E \to aEb \mid \varepsilon$

$A \to aA \mid a$

$B \to bB \mid b$

Example: Design CFG to represent the strings with balanced pairs of parentheses.

CFG: $S \to (S) \mid SS \mid \varepsilon$

Example: Design CFG to represent the Language

$L = \{w \mid n_0(w) = n_1(w) \}$

CFG: $S \to 0S1 \mid 1S0 \mid SS \mid \varepsilon$

Example: Design CFG to represent binary strings with alternating 0's and 1's.

CFG: $S \rightarrow A \mid B$
$A \rightarrow 0B \mid \varepsilon$
$B \rightarrow 1A \mid \varepsilon$

Example: Design CFG to represent the language
$L = \{ 0^i1^i2^j \mid i, j >= 1\}$

CFG: $S \rightarrow EA$
$E \rightarrow 0E1 \mid \varepsilon$
$A \rightarrow 2A \mid 2$

Example: Design CFG to represent the language
$L = \{ 0^i1^j2^k \mid i, j >= 0 \text{ and } i = j + k\}$

CFG: $S \rightarrow 0S2 \mid A$
$A \rightarrow 0A1 \mid \varepsilon$

Example: Design CFG to represent the language
$L = \{ 0^i1^j2^k \mid i, j >= 0 \text{ and } k = i + j\}$

CFG: $S \rightarrow 0S2 \mid A$
$A \rightarrow 1A2 \mid \varepsilon$

Example: Design CFG to represent the language
$L = \{ 0^i1^j2^k \mid i, j >= 0 \text{ and } j = i + k\}$

CFG: $S \rightarrow EF$
$E \rightarrow 0E1 \mid \varepsilon$
$F \rightarrow 1F2 \mid \varepsilon$

Example: Design CFGs to represent the strings consisting of a's and b's such that

1. First and last symbols are same.
2. First and last symbols are different.

1. $S \rightarrow aAa \mid bAb$
   $A \rightarrow aA \mid bA \mid \varepsilon$

2. $S \rightarrow aAb \mid bAa$
   $A \rightarrow aA \mid bA \mid \varepsilon$

Example: Design CFG to represent the strings consisting of a's and b's such that the first, last and the symbols are same.

CFG: $S \rightarrow aAa \mid bBb \mid a \mid b$
$A \rightarrow aAa \mid aAb \mid bAa \mid bAb \mid a$
$B \rightarrow aBa \mid aBb \mid bBa \mid bBb \mid b$

Example: Write CFG to represent arithmetic expressions with + and * as operators and identifiers as operands.
An identifier is the letter ‘a’ or ‘b’ followed by any sequence of a's, b's, 0's, and 1's.

CFG: $E \rightarrow E + E \mid E * E \mid (E) \mid I$
$I \rightarrow Ia \mid Ib \mid I0 \mid I1 \mid a \mid b$

7.3 Ambiguous Grammars:

A Grammar G is said to be ambiguous if for any valid string in the grammar there are

- Two or more Left Most Derivations **OR**
- Two or more Right Most Derivations **OR**
- Two or more Parse Trees

Example: Show that the below grammar is ambiguous.

$E \rightarrow E + E \mid E * E \mid (E) \mid id$

Consider the string id + id * id

LMD:

1. $E \overset{lmd}{\Longrightarrow} E + E \overset{lmd}{\Longrightarrow} id + E \overset{lmd}{\Longrightarrow} id + E * E \overset{lmd}{\Longrightarrow} id + id * E \overset{lmd}{\Longrightarrow} id + id * id$
2. $E \overset{lmd}{\Longrightarrow} E * E \overset{lmd}{\Longrightarrow} E + E * E \overset{lmd}{\Longrightarrow} id + E * E \overset{lmd}{\Longrightarrow} id + id * E \overset{lmd}{\Longrightarrow} id + id * id$

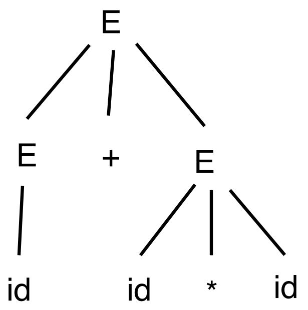
∵there are two left most derivations for the same string the grammar is ambiguous.

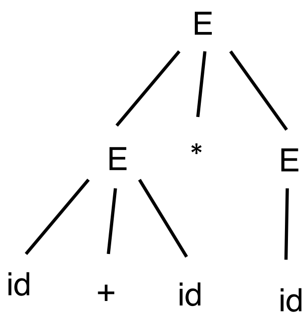
RMD:

1. $E \overset{rmd}{\Longrightarrow} E + E \overset{rmd}{\Longrightarrow} E + E * E \overset{rmd}{\Longrightarrow} E + E * id \overset{rmd}{\Longrightarrow} E + id * id \overset{rmd}{\Longrightarrow} id + id * id.$
2. $E \overset{lmd}{\Longrightarrow} E * E \overset{lmd}{\Longrightarrow} E + E * E \overset{lmd}{\Longrightarrow} id + E * E \overset{lmd}{\Longrightarrow} id + id * E \overset{lmd}{\Longrightarrow} id + id * id$

∵ There are two left most derivations for the same string the grammar is ambiguous.

Parse tree:





∵ There are two parse trees for the same string the grammar is ambiguous.

(**Note:** Any one of the above is enough to show that the grammar is ambiguous)

Example: Show that the below grammar is ambiguous.

$S \rightarrow (S) \mid SS \mid \varepsilon$

Consider the string ( )( )( )

1. $S \overset{lmd}{\Longrightarrow} SS \overset{lmd}{\Longrightarrow} (S)S \overset{lmd}{\Longrightarrow} (\varepsilon)S \overset{lmd}{\Longrightarrow} (\,)S \overset{lmd}{\Longrightarrow} (\,)SS \overset{lmd}{\Longrightarrow} (\,)(\varepsilon)S \overset{lmd}{\Longrightarrow} (\,)(\,)S \overset{lmd}{\Longrightarrow} (\,)(\,)(S) \overset{lmd}{\Longrightarrow} (\,)(\,)(\varepsilon) \overset{lmd}{\Longrightarrow} (\,)(\,)(\,)$.
2. $S \overset{lmd}{\Longrightarrow} SS \overset{lmd}{\Longrightarrow} SSS \overset{lmd}{\Longrightarrow} (S)SS \overset{lmd}{\Longrightarrow} (\varepsilon)SS \overset{lmd}{\Longrightarrow} (\,)(S)S \overset{lmd}{\Longrightarrow} (\,)(\varepsilon)S \overset{lmd}{\Longrightarrow} (\,)(\,)S \overset{lmd}{\Longrightarrow} (\,)(\,)(S) \overset{lmd}{\Longrightarrow} (\,)(\,)(\varepsilon) \overset{lmd}{\Longrightarrow} (\,)(\,)(\,)$.

∵ There are two Left Most Derivations for the same string the grammar is ambiguous.

Writing Unambiguous Grammar:

1. There is no algorithm which tells us if a CFG is ambiguous.
2. There is no algorithm to remove ambiguity from the CFG.
3. It is possible to write unambiguous grammars for the constructs in a programming language.(With the knowledge of how the construct is executed on the machine)

Example: Writing Unambiguous Grammar for arithmetic expressions with '+' and '*' as operators and identifiers as operands.

An arithmetic expression is evaluated based on the precedence (priority) and associativity of the operators.

1. An arithmetic expression is defined as the sum of products. (Multiplication has higher priority than addition)
   E → E + T | T (Sum of one or more products or terms)
2. A term (product) is defined as the product of one or more factors.
   T → T * F | F
3. A factor is an operand (identifier or number) or a parenthesised expression.
   F → id | (E)

∴The unambiguous grammar for arithmetic expressions is

E → E + T | T

T → T * F | F

T → T * F | F