UNIT-3. CONTEXT FREE GRAMMARS(CFG) AND LANGUAGES(CFL)

Topics to be covered :

Context Free Grammars and Languages : *Context free Grammars, Parse Tree, Applications of Context free Grammars, Ambiguity in the grammars and Languages. Normal Forms for Context free Grammar*

Upon completion you will be able to

- **Explain the concepts of Context free grammars and its Applications.**
- **Write the context free grammars for languages.**
- ***Demonstrate the ambiguity in the context free grammars.***
- **Explain the Normal forms for Context Free grammars**

CONTEXT FREE GRAMMARS(CFG) AND LANGUAGES(CFL)

1. Context Free Grammars

- 1.1. Introduction and Formal Defination
- 1.2. Notational Conventions
- 1.3. Derivations and Parse Tree.
  - 1.3.1. Language of a Grammar
  - 1.3.2. Sentential Forms
  - 1.3.3. Parse Tree
  - 1.3.4. Examples on derivations and Parse tree
- 1.4. Building Context free grammars for the Languages.
- 1.5. Ambiguity in Grammar with Examples
- 1.6. Applications of Context Free Grammars

2. Normal forms for Grammars

- 2.1. Introduction
- 2.2. Eliminating Ɛ-Productions
- 2.3. Eliminating Unit productions
- 2.4. Eliminating Useless Productions.
- 2.5. Chmosky Normal Forms - CNF

1. Context Free Grammar

1.1. Introduction and Formal Defination

- Up till now we studied the langauges, known as class of regular languages, that are very basic and primitives in nature. These languages are described by using the mechanism called Regular expressions and are being recognized by a finite automata (DFA).
- Also, With reference to the discussion from Pumping lemma, we know that there are certain languages which are NON Regular and these langauges are large class of languages called Context free Languages.
- These languages are very important and have very complex structure that involves recursive definiations.
- In order to describe these langauges, we have a very powerful mechanism called the Grammar and to recognize these languages we have a Automata called Push down Automata.
- The Grammar usually describes the strurcture of a sentence/construct by imposing the a heirachical structure .

- For example a grammar for the english language describes whether the perticular sentence is well formed or not.
- In otherwords, a heirachical structure of a simple sentence may be described as follows
- A **Simple <Sentence> is composed of** :
    - < Noun phrase > **followed by** < Predicate >.

  A < Noun phrase > **is made up of**

    - <Article > **followed by** <Noun>

  and Finally < Predicate > **is composed of**

    - < Verb >.
- if we associate the actual words like, **“a”**, and **“the”** with <Article>, **“boy”** and **“dog”** with < Noun > and **“runs”** and **“walks** “ with <Verbs> then the grammar tells the sentence **“ a boy runs”** and **“ the dog walks”** are well formed or not.

- The hierchical structure of the above sentence can be described by Backus Naur Form - BNF notation as follows :
  - **< Sentence > → < NounPhrase > < Predicate >**
  - **< NounPhrase > → < Article > < Noun >**
  - **< Predicate> → < Verb >**
  - **< Article > → a | the |..........**
  - **< Noun > → boy | dog |...........**
  - **< Verb > → runs | walks |.........**
- In the above notation, the symbols in angular brackets are called **syntactic variables** or **Non-terminals** namely, **< Sentence >, < NounPhrase > < Predicate >.**
- And the symbols without angular brackets are called **Terminals** or **Tokens(primitives)** namely, **a, the, boy, dog, runs, walks,......**
- The statements of the form :**< Sentence > → < NounPhrase > < Predicate > are** called **production rules**

Formal Defination Grammar

As discussed about the grammar, we know that the grammar is notation to describe the structure of a sentence and Formally it can be defined as follows :

A grammar **G** is defined as a Quadraple

G = { V, T, S, P }

Where

V is a Finite set of Variables or Non-terminals

→They denote the set of strings that forms the language

T is a Finite set of Terminal or Tokens

→ These are the basic symbols from which the strings are formed

S € V is a special Symbol called the start symbol

→ This symbol is used to genarate the sentence of a langauge

P is a finite set of Productions.

→ These are nothing but the rules to specify the manner in which the Terminals and Non Terminals can be combined to form strings

Each production rule is of the following form :

<HEAD> → <BODY> ***it can read as head derives body***

Here **<HEAD>** → is a **Non-terminal** or **the left side of a production** and **<BODY>** → is a **right side of a production** consisting of **zero or more terminals** and **Non-terminals**

Example -1 : $G = \{ V, T, S, P \}$

where $V = \{ E \}$

$T = \{ +, -, *, /, (,), id, digit \}$

$S = \{ E \}$

$$
\begin{aligned}
P = \{ \; E \rightarrow\; & E + E \\
& \mid E - E \\
& \mid E * E \\
& \mid E / E \\
& \mid (E) \\
& \mid id \\
& \mid digit \; \}
\end{aligned}
$$

Example -2 : $G = \{ V, T, S, P \}$

where $V = \{ S, A \}$

$T = \{ a, b \}$

$S = \{ S \}$

$$
\begin{aligned}
P = \{ \; & S \rightarrow AS \mid \varepsilon \\
& A \rightarrow aa \mid ab \mid ba \mid bb \\
\}
\end{aligned}
$$

1.2. Notational Conventions

Notation for CFG Derivations

There are a number of conventions in common use that help us remember the role of the symbols we use when discussing CFG's. Here are the conventions we shall use:

1. Lower-case letters near the beginning of the alphabet, a, b, and so on, are terminal symbols. We shall also assume that digits and other characters such as + or parentheses are terminals.

2. Upper-case letters near the beginning of the alphabet, A, B, and so on, are variables.

3. Lower-case letters near the end of the alphabet, such as w or z, are strings of terminals. This convention reminds us that the terminals are analogous to the input symbols of an automaton.

4. Upper-case letters near the end of the alphabet, such as X or Y, are either terminals or variables.

5. Lower-case Greek letters, such as α and β, are strings consisting of terminals and/or variables.

There is no special notation for strings that consist of variables only, since this concept plays no important role. However, a string named α or another Greek letter might happen to have only variables.

1.3. Derivations and Parse Tree

- In order to define the langauge assicaited with Grammar G we apply the productions rules to infer that certain strings are in the language.
- There are two approaches :
  - **Approach -1** → This is more conventional approach where the **production rules used from body to head**. That is, we take strings known to be in the language of each of the **variables** of the **body of production**, concatenate them in the proper order, with any terminals appearing in the body, and infer that the resulting strings is in the language of the varibale in the **head**.

  This is called recursive inference.

1.3. Derivations and Parse Tree

Approach -2 → In this approach we use the **productions from head to body**. We **expand the Start Symbol** using one of its productions. We further expand the **resulting string by replacing one of the variables by the body of one of the productions**, and so on, until we **derive a string consisting entirely of terminals**. The **langauge of the grammar is all strings of teminals,** that we can obtain in this way.

This is called as **derivation** process.

Example → Consider the grammar with following Productions :

$$E \rightarrow E + E \mid E - E$$
$$E \rightarrow (E)$$
$$E \rightarrow E * E \mid E / E$$
$$E \rightarrow - E$$
$$E \rightarrow id \mid digit$$

- **Let w = id + id,** and Consider the derivation step - **$E \Rightarrow E+E$** , which is the initial step of expansion of start symbol → E
  - Since, **E+E derives from E**
    - we can **replace E by E+E** to get a string - **E+E** in the derivation step**.**

      i.e. **$E \Rightarrow E+E$**
    - to able to do this, we have to have a production rule **E→E+E** in our grammar.
  - This is then continued until we **derive a string consisting entirely of terminals.**
    - In oherwords, in the next derivation step, We can **replace id by E** as we have a production rule **E→id** to get a **string - id + E.** .

      i. e. **$E \Rightarrow E+E \Rightarrow id+E$**
    - Finally, we can **replace id by E** again to get a **string-id+id**, consisting of eniterly of terminals to end the derivation Process.

      i. e. **$E \Rightarrow E+E \Rightarrow id+E \Rightarrow id+id$**

- Beginning with Start Non-terminal, we identify the sequence of **replacements for non-terminal symbols,** until we **derive a string consisting entirely of terminals** is called a **derivation** of string **id+id** from E.
- In general, **$\alpha A\beta \Rightarrow \alpha\gamma\beta$** is a derivation step, where **$A \rightarrow \gamma$** is a production in our grammar and **α and β** are arbitrary **strings of terminal and non-terminal symbols.**
- In the **derivation process,** we have the choices of **replacements of Non-terminal symbols and** to restrict the number of choices there are two types of derivations namely, Leftmost Derivation and Rightmost Derivation.

  - **Leftmost Derivation** : A derivation is said to be **Leftmost derivation** if, at each step the replacements are done by considering **LEFTMOST** non-terminal.
  - We indicate Leftmost derivation by a symbol $\rightarrow$ $\underset{lm}{\Rightarrow}$

- **Rightmost Derivation** : A derivation is said to be **Rightmost derivation** if, at each step the replacements are done by considering **RIGHTMOST** non-terminal.
- We indicate the **Rightmost derivation** by a symbol → $\underset{rm}{\Rightarrow}$

Example

1.	E	→	I
2.	E	→	$E+E$
3.	E	→	$E*E$
4.	E	→	(E)
5.	I	→	a
6.	I	→	b
7.	I	→	Ia
8.	I	→	Ib
9.	I	→	$I0$
10.	I	→	$I1$

Figure 5.2: A context-free grammar for simple expressions

Leftmost Derivation For the Input → a*(a+b00)

$$E \underset{lm}{\Rightarrow} E*E \underset{lm}{\Rightarrow} I*E \underset{lm}{\Rightarrow} a*E \underset{lm}{\Rightarrow}$$
$$a*(E) \underset{lm}{\Rightarrow} a*(E+E) \underset{lm}{\Rightarrow} a*(I+E) \underset{lm}{\Rightarrow} a*(a+E) \underset{lm}{\Rightarrow}$$
$$a*(a+I) \underset{lm}{\Rightarrow} a*(a+I0) \underset{lm}{\Rightarrow} a*(a+I00) \underset{lm}{\Rightarrow} a*(a+b00)$$

Rightmost Derivation For the Input → a*(a+b00)

$$E \underset{rm}{\Rightarrow} E*E \underset{rm}{\Rightarrow} E*(E) \underset{rm}{\Rightarrow} E*(E+E) \underset{rm}{\Rightarrow}$$
$$E*(E+I) \underset{rm}{\Rightarrow} E*(E+I0) \underset{rm}{\Rightarrow} E*(E+I00) \underset{rm}{\Rightarrow} E*(E+b00) \underset{rm}{\Rightarrow}$$
$$E*(I+b00) \underset{rm}{\Rightarrow} E*(a+b00) \underset{rm}{\Rightarrow} I*(a+b00) \underset{rm}{\Rightarrow} a*(a+b00)$$

1.3.1 Language of a Grammar

5.1.5 The Language of a Grammar

If $G(V, T, P, S)$ is a CFG, the *language* of G, denoted $L(G)$, is the set of terminal strings that have derivations from the start symbol. That is,

$$L(G) = \{w \text{ in } T^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w\}$$

1.3.2 Sentential Form

5.1.6 Sentential Forms

Derivations from the start symbol produce strings that have a special role. We call these "sentential forms." That is, if $G = (V, T, P, S)$ is a CFG, then any string α in $(V \cup T)^*$ such that $S \overset{*}{\Rightarrow} \alpha$ is a *sentential form*. If $S \underset{lm}{\overset{*}{\Rightarrow}} \alpha$, then α is a *left-sentential form*, and if $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha$, then α is a *right-sentential form*. Note that the language $L(G)$ is those sentential forms that are in T^*; i.e., they consist solely of terminals.

Example 5.8: Consider the grammar for expressions from Fig. 5.2. For example, $E * (I + E)$ is a sentential form, since there is a derivation

$$E \Rightarrow E * E \Rightarrow E * (E) \Rightarrow E * (E + E) \Rightarrow E * (I + E)$$

However this derivation is neither leftmost nor rightmost, since at the last step, the middle E is replaced.

As an example of a left-sentential form, consider $a * E$, with the leftmost derivation

$$E \underset{lm}{\Rightarrow} E * E \underset{lm}{\Rightarrow} I * E \underset{lm}{\Rightarrow} a * E$$

Additionally, the derivation

$$E \underset{rm}{\Rightarrow} E * E \underset{rm}{\Rightarrow} E * (E) \underset{rm}{\Rightarrow} E * (E + E)$$

shows that $E * (E + E)$ is a right-sentential form. □

1.3.3. Parse Tree

- Parse Tree is **graphical representation of a derivation process** which clearly shows how the symbols of a terminal strings are grouped into a substring belonging to a language.
- Formaly it is defined as follows :

Let us fix on a grammar $G = (V, T, P, S)$. The *parse trees* for G are trees with the following conditions:

1. Each interior node is labeled by a variable in V.
2. Each leaf is labeled by either a variable, a terminal, or ϵ. However, if the leaf is labeled ϵ, then it must be the only child of its parent.
3. If an interior node is labeled A, and its children are labeled

$$X_1, X_2, \ldots, X_k$$

respectively, from the left, then $A \rightarrow X_1 X_2 \cdots X_k$ is a production in P. Note that the only time one of the X's can be ϵ is if that is the label of the only child, and $A \rightarrow \epsilon$ is a production of G.

Example of Parse Tree on input string - → a*(a+b00)

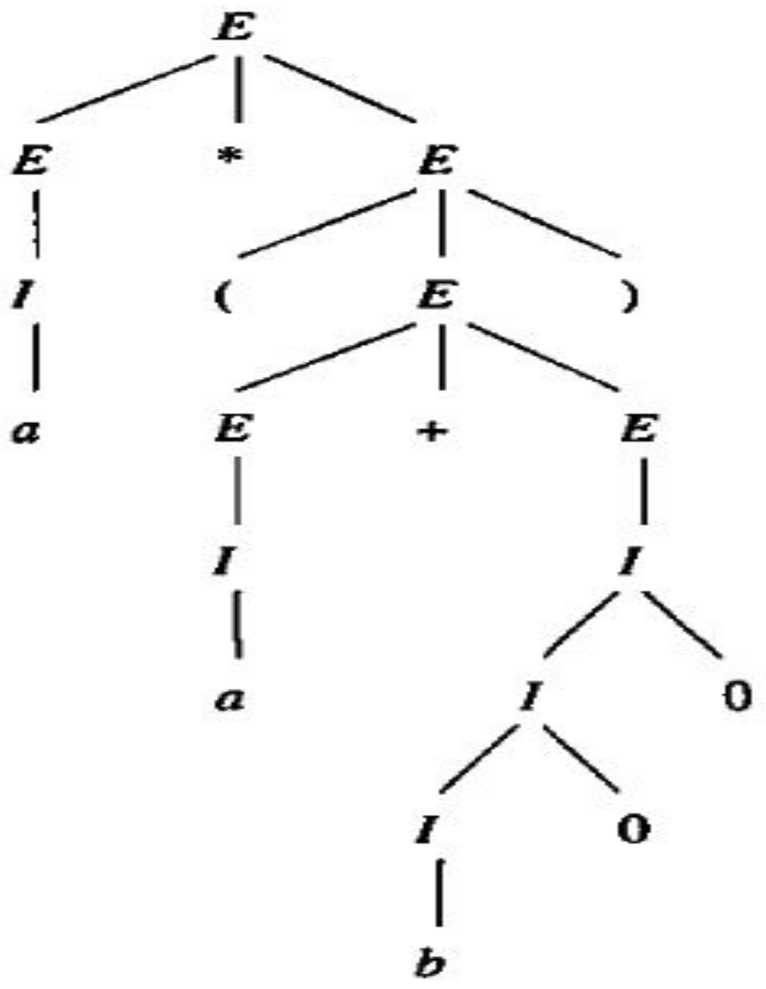


Figure 5.6: Parse tree showing $a * (a + b00)$ is in the language of our expression grammar

1.3.4. Examples on Derivations and Parse Tree

Example -1

Consider the grammar G = { {S, A, B}, {a, b}, {S}, P }

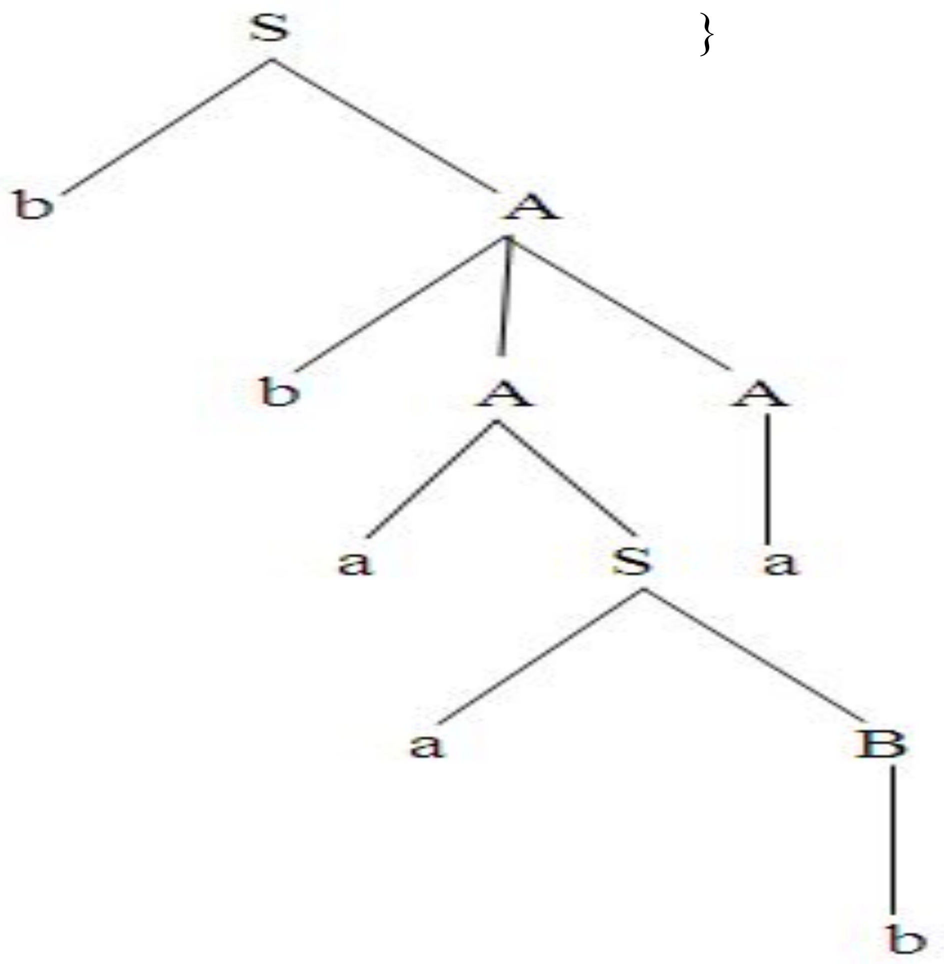
$$
\begin{aligned}
P - \{\ & S \rightarrow b\,A \mid a\,B \\
& A \rightarrow b\,A\,A \mid a\,S \mid a \\
& B \rightarrow a\,B\,B \mid b\,S \mid b \\
\}\ &
\end{aligned}
$$

Write leftmost and rightmost derivation for the following sentences along with Parse tree.

i. bbaaba **ii. bbbaaaba**

i) bbaaba

$$P - \{ S \rightarrow b\,A \mid a\,B$$
$$A \rightarrow b\,A\,A \mid a\,S \mid a$$
$$B \rightarrow a\,B\,B \mid b\,S \mid b$$
$$\}$$



ii) bbbaaaaba (Home Work)

Leftmost Derivation

$$\begin{aligned} S &\Rightarrow b\,A \\ &\Rightarrow b\,b\,\underline{A}\,A \\ &\Rightarrow b\,b\,a\,\underline{S}\,A \\ &\Rightarrow b\,b\,a\,a\,\underline{B}\,A \\ &\Rightarrow b\,b\,a\,a\,b\,A \\ &\Rightarrow b\,b\,a\,a\,b\,a \end{aligned}$$

Rightmost derivation

$$\begin{aligned} S &\Rightarrow b\,\underline{A} \\ &\Rightarrow b\,b\,A\,\underline{A} \\ &\Rightarrow b\,b\,\underline{A}\,a \\ &\Rightarrow b\,b\,a\,\underline{S}\,a \\ &\Rightarrow b\,b\,a\,a\,\underline{B}\,a \\ &\Rightarrow b\,b\,a\,a\,b\,a \end{aligned}$$

Fig. 3.5 Parse Tree for the string bbaaba

Example -2

Exercise 5.4.7: The following grammar generates *prefix* expressions with operands x and y and binary operators +, −, and $*$:

$$E \rightarrow +EE \mid *EE \mid -EE \mid x \mid y$$

a) Find leftmost and rightmost derivations, and a derivation tree for the string +*-xyxy.

Example -3

$$\begin{array}{lcl} S & \rightarrow & A1B \\ A & \rightarrow & 0A \mid \epsilon \\ B & \rightarrow & 0B \mid 1B \mid \epsilon \end{array}$$

Give leftmost and rightmost derivations of the following strings:

* a) 00101.

b) 1001.

c) 00011.

Example -4

$S \rightarrow AS \mid \varepsilon$

$A \rightarrow aa \mid ab \mid ba \mid bb$

Give Leftmost and rightmost derivations and a parse tree for following Srings

i. aabbba ii. baabab iii. aaabbb

Example -5

$$S \rightarrow aAS \mid a$$
$$A \rightarrow SbA \mid ba$$

Give Leftmost and rightmost derivations and Parse tree for following Srings - aabbba

Example -6

$$S \rightarrow AaAb \mid BbBa$$
$$A \rightarrow aAb \mid bAB \mid d$$
$$B \rightarrow aB \mid bB \mid a$$

Give Leftmost and Rightmost derivations and Parse Tree for following Srings

i. aabbba   ii. badbabaadb

Example -7

$$E \rightarrow ET+ \mid T$$
$$T \rightarrow TF* \mid F$$
$$F \rightarrow FP\uparrow \mid P$$
$$P \rightarrow E \mid id$$

Give Leftmost and Rightmost derivations and Parse Tree for following Srings

i. idid id*+id+   ii. idid id↑id*id*+

1.4. Building Context free grammars for the Languages.

- In order to learn the writing of context free grammars,We can start with following easiest steps :

  - **A. Building Context free grammar for Regular Languages**
    i. Context free grammar from finite automata.
    ii. From Regular Expressions
  - **B. Building Context free grammar for Other Languages**

A. Building Context free grammar for Regular Languages

i. Context free grammar from finite automata.

Method : Let M = { Q, ∑, δ, q0, F} be a DFA accepting the language L. The Grammar G = {V, T, S, P} can be constructed as follows :

V = { q0, q1...qn} → The states of DFA will be Non-Terminals

T = { ∑ } → The inputs are the Terminal Symbols.

S = q0 → The start state of DFA will be the Start Symbol

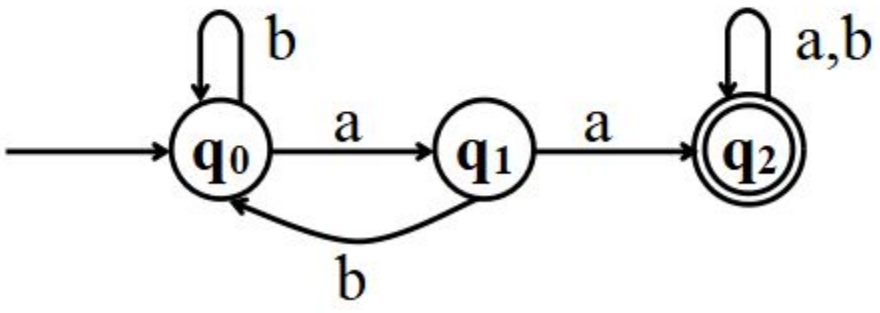
P = The productions can obtained as follows :

If δ(qi, a) = qj then introduce the productions as - qi → aqj

if qi € F then introduce the productions as - qi → ε

Examples :

Example -1



G = { V, T, S, P}

V= {q_0, q_1, q_2}   S = { q_0}

T = { a, b }

P { Refer Following table }

δ→Transitions	Productiuons
$\delta(q_0, a) = q_1$	$q_0 \rightarrow aq_1$
$\delta(q_0, b) = q_0$	$q_0 \rightarrow bq_0$
$\delta(q_1, a) = q_2$	$q_1 \rightarrow aq_2$
$\delta(q_1, b) = q_0$	$q_1 \rightarrow bq_0$
$\delta(q_2, a) = q_2$	$q_2 \rightarrow aq_2$
$\delta(q_2, b) = q_2$	$q_2 \rightarrow bq_2$
q_2 € F	$q_2 \rightarrow \varepsilon$

B. Building Context free grammar for Regular Languages

ii. Context free grammar from Regular Expressions

Method : The grammar can obtained by breaking the given regular expression into smaller one and introducing the new variables and associated productions for the smaller Regular expressions. expressions.

Example - 1. Regular Expression = a*

Productions $S \rightarrow aS$

Example - 2. Regular Expression = ab(a + b)*

Productions $S \rightarrow S_1S_2$

$S_1 \rightarrow ab$

$S_2 \rightarrow aS_2 \mid bS_2 \mid \varepsilon$

B. Building Context free grammar for other Languages

i. Obtain a grammar to genarate the following Languages

1. $L = \{ a^n \ b^n \mid n >= 0 \}$
2. $L = \{ a^{n+1} \ b^n \mid n >= 0 \}$
3. $L = \{ a^n \ b^{n+1} \mid n >= 0 \}$
4. $L = \{ a^{2n} \ b^n \mid n >= 0 \}$
5. $L = \{ a^n \ b^{2n} \mid n >= 0 \}$
6. $L = \{ ww^R \mid w \in (a+b)^* \}$
7. $L = \{ w \mid n_a(w) = n_b(w)$ and $w \in (a+b)^* \}$
8. $L = \{ w \mid n_a(w) > n_b(w)$ and $w \in (a+b)^* \}$
9. $L = \{ w \mid n_a(w) < n_b(w)$ and $w \in (a+b)^* \}$
10. $L = \{ 0^m 1^m \ 2^n \mid m, n >= 0 \}$
11. $L = \{ a^n \ b^{n-3} \mid n >= 3 \}$

B. Building Context free grammar for other Languages

12. $L = \{ a^n \ b^{n+3} \mid n >= 0 \}$
13. $L = \{ a^i \ b^j \ c^k \mid j, k >= 0 \text{ and } i=j+k \}$
14. $L = \{ a^i \ b^j \ c^k \mid i, k >= 0 \text{ and } j=i+k \}$
15. $L = \{ a^n \ b^m \ c^k \mid m, n >= 0 \text{ and } n+2m=k \}$
16. $L = \{ a^m \ b^n \mid m > n, m, n >=0 \}$
17. $L = \{ a^m \ b^n \mid m < n, m, n >=0 \}$
18. $L = \{ a^m \ b^n \mid m <> n, m, n >=0 \}$
19. $L = \{ a^n \ ww^R \ b^n \mid n >= 0, w \text{ €} \ (a+b)^* \}$

B. Building Context free grammar for other Languages

i. Obtain a grammar to genarate the following Languages

1. L = $\{ a^n\ b^n \mid n >= 0 \}$

   Answer = $S \rightarrow a\,S\,b \mid \varepsilon$

2. L = $\{ a^{n+1}\ b^n \mid n >= 0 \}$

   Answer = $S \rightarrow a\,S\,b \mid a$

3. L = $\{ a^n\ b^{n+1} \mid n >= 0 \}$

   Answer = $S \rightarrow a\,S\,b \mid b$

4. L = $\{ a^{2n}\ b^n \mid n >= 0 \}$

   Answer = $S \rightarrow aa\,S\,b \mid \varepsilon$

5. L = $\{ a^n\ b^{2n} \mid n >= 0 \}$

   Answer = $S \rightarrow a\,S\,bb \mid \varepsilon$

B. Building Context free grammar for other Languages

6. L = { ww^R | w € (a+b)* }

   Answer = $S \rightarrow a\,S\,a \mid b\,S\,b \mid a \mid b \mid \varepsilon$

7. L = { w | $n_a(w) = n_b(w)$ and w € (a+b)* }

   Answer = $S \rightarrow a\,S\,b \mid b\,S\,a \mid \varepsilon$

8. L = { w | $n_a(w) > n_b(w)$ and w € (a+b)* }

   Answer = $S \rightarrow a\,S_1\,b \mid b\,S_1\,a \mid S_1$

   $S_1 \rightarrow a \mid a\,S_1$

9. L = { w | $n_a(w) < n_b(w)$ and w € (a+b)* }

   Answer = $S \rightarrow a\,S_1\,b \mid b\,S_1\,a \mid S_1$

   $S_1 \rightarrow b \mid b\,S_1$

10. L = { $0^m 1^m\ 2^n$ | m, n>= 0 }

    Answer = $S \rightarrow S_1\,S_2$

    $S_1 \rightarrow 0\ S_1 1 \mid \varepsilon$

    $S_2 \rightarrow 2\,S_2 \mid \varepsilon$

B. Building Context free grammar for other Languages

11. $L = \{ a^n \; b^{n-3} \mid n >= 3 \}$

    Answer = $S \rightarrow a\,S\,b \mid aaa$

12. $L = \{ a^n \; b^{n+3} \mid n >= 0 \}$

    Answer = $S \rightarrow a\,S\,b \mid bbb$

13. $L = \{ a^i \, b^j \, c^k \mid j, k >= 0 \text{ and } i=j+k \}$

    Answer = Since $i = j+k$

    $L = \{ a^{j+k} \, b^j \, c^k \}$

    $L = \{ a^j \, a^k \, b^j \, c^k \}$

    $L = a^k \, a^j \, b^j \, c^k$

    $S \rightarrow a\,S\,c \mid S_1$

    $S_1 \rightarrow a\,S_1\,b \mid \varepsilon$

14. $L = \{ a^i \, b^j \, c^k \mid i, k >= 0 \text{ and } j=i+k \}$

15. $L = \{ a^n \, b^m \, c^k \mid m, n >= 0 \text{ and } n+2m=k \}$

B. Building Context free grammar for other Languages

16. $L = \{ a^m \ b^n | \ m > n, m, n >= 0 \}$
17. $L = \{ a^m \ b^n | \ m < n, m, n >= 0 \}$
18. $L = \{ a^m \ b^n | \ m <> n, m, n >= 0 \}$

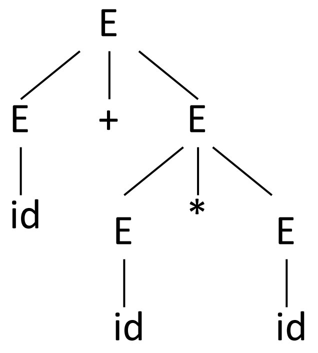
1.5. Ambiguity in the Grammar

- A grammar produces more than one parse tree for a sentence is called as an ***Ambiguous* Grammar.** In other words there exists more than one leftmost or more than one rightmost derrivations for some sentence S.

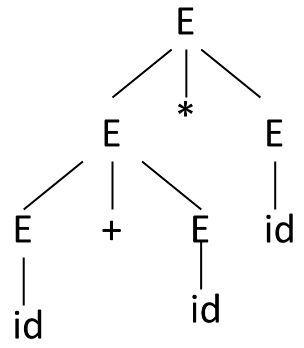
Example : $E \rightarrow E + E \mid E - E$
$E \rightarrow (E)$
$E \rightarrow E * E \mid E / E$
$E \rightarrow - E$
$E \rightarrow id \mid digit$

Two Leftmost Derivations for → S= id+id*id

$E \Rightarrow E+E \Rightarrow id+E \Rightarrow id+E*E \Rightarrow id+id*E \Rightarrow id+id*id$



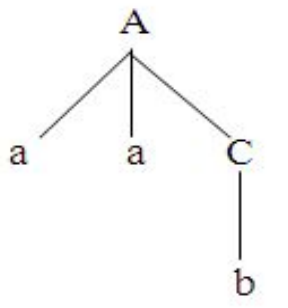
$E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow id+E*E \Rightarrow id+id*E \Rightarrow id+id*id$



Example: Consider the following grammar

A → B C | a a C
B → a | B a
C → b
Prove that the grammar is ambiguous

Tree 1

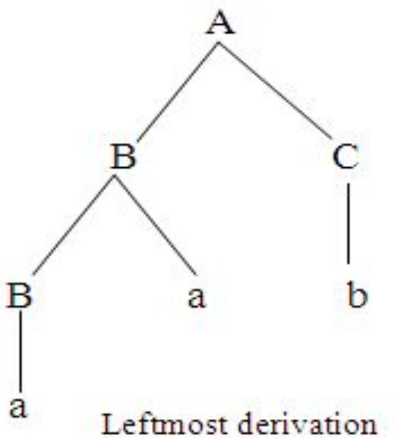


Leftmost derivation

$A \Rightarrow a\ a\ \underline{C}$

$\Rightarrow a\ a\ b$

Tree 2



Leftmost derivation

$A \Rightarrow \underline{B}\ C$

$\Rightarrow \underline{B}\ a\ c$

$\Rightarrow a\ a\ c \Rightarrow a\ a\ b$

Fig. 3.20 Two leftmost derivation for string a a b

Examples on Ambiguity

Example - 1 : $S \rightarrow AS \mid aaa$

$A \rightarrow a \mid Aa$

$B \rightarrow a$

Example - 2 : $S \rightarrow aS \mid X$

$X \rightarrow aX \mid a$

Example - 3 : $S \rightarrow aSbS \mid bSaS \mid \varepsilon$

Example - 4 : $S \rightarrow SS \mid aSb \mid bSa \mid \varepsilon$

Example - 5 : $S \rightarrow AB \mid aaB$

$A \rightarrow a \mid Aa$

$B \rightarrow b$

Example - 6 : $S \rightarrow aS \mid aSbS \mid \varepsilon$

Example - 7 $S \rightarrow aB \mid bA$

$A \rightarrow aS \mid bAA \mid a$

$B \rightarrow bS \mid aBB \mid b$

$w = aaabbabbba$

5.4.4 Inherent Ambiguity

A context-free language L is said to be *inherently ambiguous* if all its grammars are ambiguous. If even one grammar for L is unambiguous, then L is an unambiguous language. We saw, for example, that the language of expressions generated by the grammar of Fig. 5.2 is actually unambiguous. Even though that grammar is ambiguous, there is another grammar for the same language that is unambiguous — the grammar of Fig. 5.19.

We shall not prove that there are inherently ambiguous languages. Rather we shall discuss one example of a language that can be proved inherently ambiguous, and we shall explain intuitively why every grammar for the language must be ambiguous. The language L in question is:

$$L = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$$

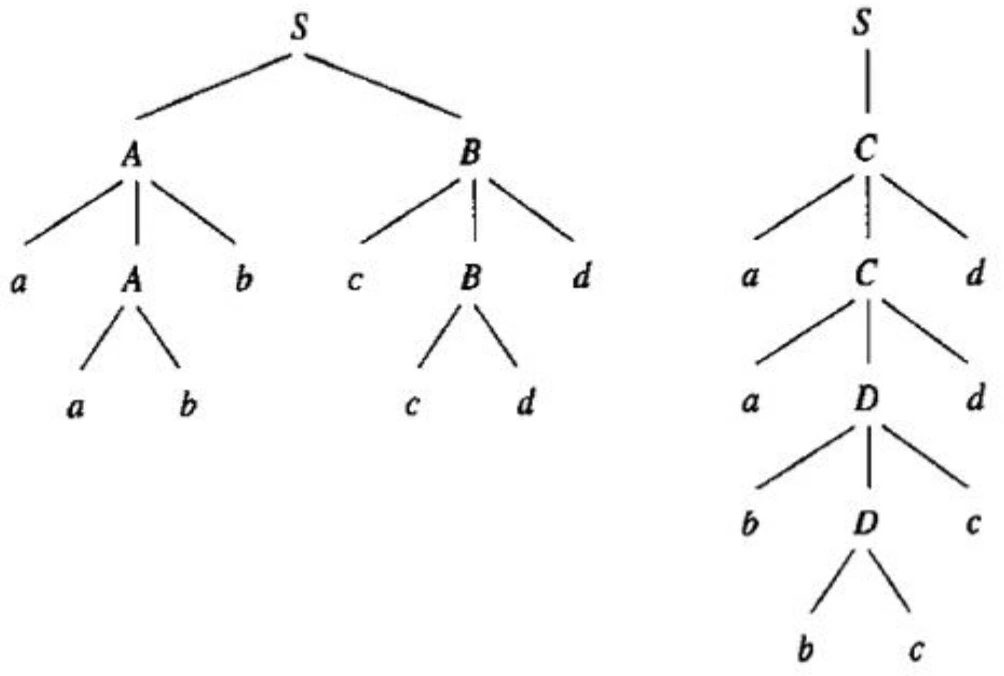
$$\begin{array}{lcl} S & \rightarrow & AB \mid C \\ A & \rightarrow & aAb \mid ab \\ B & \rightarrow & cBd \mid cd \\ C & \rightarrow & aCd \mid aDd \\ D & \rightarrow & bDc \mid bc \end{array}$$

Figure 5.22: A grammar for an inherently ambiguous language

This grammar is ambiguous. For example, the string *aabbccdd* has the two leftmost derivations:

1. $S \underset{lm}{\Rightarrow} AB \underset{lm}{\Rightarrow} aAbB \underset{lm}{\Rightarrow} aabbB \underset{lm}{\Rightarrow} aabbcBd \underset{lm}{\Rightarrow} aabbccdd$

2. $S \underset{lm}{\Rightarrow} C \underset{lm}{\Rightarrow} aCd \underset{lm}{\Rightarrow} aaDdd \underset{lm}{\Rightarrow} aabDcdd \underset{lm}{\Rightarrow} aabbccdd$



(a) (b)

Figure 5.23: Two parse trees for *aabbccdd*

1.6. Applications of Comtext free grammar

- **Parsers :**

  The **Parsers** are nothing but the **program modules** written for the **Compiler** that **discovers the structure of source program** and **represents it in a Parse tree**. The **Context free grammars** are used in **describing the structure of source program** for many programming languages.

- **The YACC Parser Generator :**

  YACC Parser Genarator is a **special tool on UNIX platform** that uses Context free grammars as a part of **Rule section of YACC specifications** along with actions to genarate the Parser program module for a typical Compiler.

1.6. Applications of Comtext free grammar

- **Markup Languages :**

  In the Markup language like **HTML -HyperText Markup Language** has **two major functions** : one is to **create a link between documents** and second is to **describe the format of the document**. The **Context free grammars** are used describe **the structure of legal HTML documents** and guides the **processing of documents** with respect the display of the document on a monitor or printer.

- **XML and Document Type Definitions (DTD) :**

  In the development **XML -Extensible Markup Language**, as a part of **Document Type Definitions (DTD)**, the **Context Free Grammars** are used to **describe the allowable tags** and **how these tags can be nested**, where each **tag is deals with formating of text** and along **with its meaningfullness.**

2. Normal Forms of Context Free Grammar

2.1. Introduction

- We know that **Context Free Grammars** are more powerful enough to describe t**he syntax of most of the programming languages.**
- We also know that the languages that are described by the **Context free grammars** are called **Context free langauges**.
- in order to take the advantage during the **implementation of these context free langauges** we ensure that the **Conetxt free grammars are** in **one of the Normal froms** where we impose **certain restrictions** in writing the productions for **Context free grammar.**
- There are Two types of normal forms for the context **free grammars** :
  1. **Chomsky Normal Form - CNF**
  2. **Greibach Normal Form - GNF**

- In **Chomsky Normal form**, all the productions are of the form : **A→BC** or **A → a**, where **A, B,** and **C** are **all variables** and **a** is terminal symbol
- In **Greiback Normal Form** all the productions are of the form :

  A→ax where **A** is a **variable, a € T and x € V***
- To get these normal forms for the grammmars, we need to make number of preliminary simplifications. Following are the simplications:
  1. We must **eliminate Ɛ-productions**, those of the from **A →Ɛ** for some **variable A**
  2. We must **eliminate Unit-productions**, those of the from **A → B** for some **variables A** and **B.**
  3. We must **eliminate useless varibales**, those **variables** that **do not appear in any derivation and genarating a terminal string** from the **start symbol** of the **grammar.**

2.2. Eliminating Ɛ-Productions

Let G=(V,T,S,P) be a context free grammar. A production of the form **A → Ɛ** for some **A € V** is called **Ɛ- production**.

A **variable A** is said to be **Nuallable,**
if A ⇒*ε, in zero or more derivation steps are posssible. i.e in every derivation step, we come across only ‘Ɛ ‘ in the sentential form.

2.2. Eliminating Ɛ-Productions

1.We first find all **set V_n of Nullable variables of G** using the following steps:

a. All productions of the form A→Є put A into V_n

b. Repeat the following steps until no further variables are added to V_n

- For all productions **$A \rightarrow B_1,B_2,B_3.....B_n$** where **$B_i$ € V_n** then **add A to V_n.**

2. We then construct **P1(new Production set)** by looking at productions in P of the form **$A \rightarrow x_1,x_2,x_3......x_n$ for n>1,** where **x_i** € to **VUT** for each such production of **P.**

- we put into **P1 that productions** as well as **all those productions that are generated by replacing null able variables with Є** in all possible combinations.

Working with Example :

$S \rightarrow ABA$

$A \rightarrow aA \mid \varepsilon$,

$B \rightarrow bB \mid \varepsilon$

Step -1. Finding Nullable Variables V_n :

$V_n = \{ A, B, S \} \rightarrow$ **A, B are Nullable as they have** Ɛ- productions and **S is Nullable** as its **production body** contains all **Nullable variables**

Step -2. Remove all **Ɛ- productions** from given **grammar G** and add new productions by **replacing null able variables with Ɛ ie. NULL** in all possible combinations. The resulting grammar is follows :

$S \rightarrow ABA \mid BA \mid AA \mid AB \mid A \mid B$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

Examples on Elimination of **ε-productions**

Example -1

$$S \to AB$$
$$A \to aAA \mid \epsilon$$
$$B \to bBB \mid \epsilon$$

Example -2

$$S \to ASB \mid \epsilon$$
$$A \to aAS \mid a$$
$$B \to SbS \mid A \mid bb$$

Example -3

$$S \to 0A0 \mid 1B1 \mid BB$$
$$A \to C$$
$$B \to S \mid A$$
$$C \to S \mid \epsilon$$

Example -4

$$S \to 0A0 \mid 1B1 \mid BB$$
$$A \to C$$
$$B \to S \mid A$$
$$C \to S \mid \epsilon$$

2.3. Eliminating UNIT-Productions

Let **G= { V, T, S, P}** be a context -free -grammar.

A production of the form **A → B** for some variable **A and B € V** is called **Unit production.**

They can be eliminated using the following steps :

1. **Add all Non unit productions to P1 where P1 is a new Production set.**
2. **For Each variable A find all variables such that A ⇒*B is possible that is in the derivation process from A we encounter only single variables in the sentential form to B.(no other terminal symbols). This is obtained by constructing the dependency graph for unit productions only.**
3. **By substitution to unit productions we add new productions to P1 for each variable that is if A⇒*B is possible then add all non-unit productions of B to variable A.**
4. **Resulting grammar with P1 productions generates the same language as accepted by the original grammar G.**

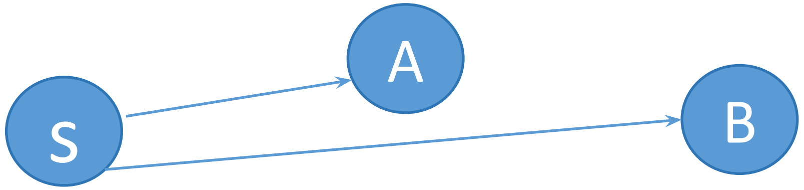
Working with Example :

S → ABA | BA | AA | AB | A | B
A → aA | a
B → bB | b

Step -1. The unit productions are S → A, S →B. Now List all Non Unit productions of Given grammar G

S → ABA | BA | AA | AB
A → aA | a
B → bB | b

Step -2. Write dependency graph for only **UNIT - productions and for Each variable A find all variables such that A ⇒*B is possible.**



Possible Derivation are of the from

A⇒*B for variables A and B are :

S ⇒*A and S ⇒*B

Add **all productions of Variable A** and **B** to the **Varibale S** and resulting grammar is follows :

$$S \rightarrow ABA \mid BA \mid AA \mid AB \mid aA \mid a \mid bB \mid b$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

Examples on Elimination of **Unit-productions**

Example -1

$$S \rightarrow AB$$
$$A \rightarrow aAA \mid \epsilon$$
$$B \rightarrow bBB \mid \epsilon$$

Example -2

$$\begin{array}{lcl} S & \rightarrow & ASB \mid \epsilon \\ A & \rightarrow & aAS \mid a \\ B & \rightarrow & SbS \mid A \mid bb \end{array}$$

Example -3

$$\begin{array}{lcl} S & \rightarrow & 0A0 \mid 1B1 \mid BB \\ A & \rightarrow & C \\ B & \rightarrow & S \mid A \\ C & \rightarrow & S \mid \epsilon \end{array}$$

Example -4

$$\begin{array}{lcl} S & \rightarrow & 0A0 \mid 1B1 \mid BB \\ A & \rightarrow & C \\ B & \rightarrow & S \mid A \\ C & \rightarrow & S \mid \epsilon \end{array}$$

Example - 5

$$\begin{array}{lcl} S & \rightarrow & Aa \mid B \\ A & \rightarrow & a \mid bc \mid B, \\ B & \rightarrow & A \mid bb \end{array}$$

Example - 6

$$\begin{array}{lcl} S & \rightarrow & a \mid aA \mid B \mid C \\ A & \rightarrow & aB \\ B & \rightarrow & aA \mid a \\ C & \rightarrow & cCD \\ D & \rightarrow & ddd \end{array}$$

2.4. Eliminating Useless Productions

Let **G = { V, T, S P}** be a **context free grammar**, a **variable A € V** is said to be **useful**, if and only if there exits, atleast **one W € L(G)**,

such that **S ⇒....xAy⇒ ....⇒ W** is possible.

In otherwords a **variable A € V** is **useful** if it appears **atleast once** in the **derivation process** and evetually leads to a **terminal string w € L(G)**.

A **variable 'A'** is not useful is said to be **Useless** and associated **productions are called as Useless productions**.

Procedure to Eliminate Useless variables and Productions

1. **First we identify the set of variables that can lead to a terminal string by the following steps:**
   A. Set **V1 to NULL** where V_1 is **set of useful variables.**
   B. Repeat the following steps until **no more variables are added to V1**:
      - For every **A € V** for which **P has a production of the form** $A \rightarrow x_1x_2x_3....x_n$ with all x_i € $(V_1 \cup T)$, add **A to** V_1.
      - Take all the productions into **P1 whose symbols** are all in $(V_1 \cup T)$, where **P1** is set of useful productions.

2. We eliminate the variables from Productions P1, that cannot be reached from the start variable. For this we draw the dependency graph for the variable set V1 found in step 1.

i.e. Dependency Graph is a graph with vertices labeled with variables with an edge between vertices C and D iff there is a production of the form c->xDy.

From the dependency graph. we obtain the useful variables as follows :

- if there is a direct or indirect path from **vertex labeled S** to the **vertex labeled A** then **A is a useful variable**. ie. the variables are reachable from **vertex 'S'** where **S is Start Symbol** of the Grammar.
- Any **vertex 'X'** if there is **no direct or indirect path** from **Vertex 'S'** then the **Vertex 'X"** is ignored.
- Finally, the grammar G is modified that contains only useful variables and its associated productions

Working with the Example : $S \rightarrow a \mid aA$

$A \rightarrow aB$

$B \rightarrow aA \mid a$

$D \rightarrow ddd$

1. First we identify the set of variables that can lead to a terminal string by the following steps:

Steps	V1- Set of Useful variables	New Set of useful variables	Productions	Remarks
1	Ø	**S, B and D**	**S → a** **B → a** **D → ddd**	**S, B and D** derive only Terminal strings, so Add **S, B and D to new set of V1**
2.	**S, B and D** **Update w.r.t New Set of variables**	**A**	S → a B → a D → ddd **A → aB**	Body of Production **A €{S, B, D } Ụ{ T }**, Add the **variable 'A' to new Set of V1**
3	**S, B D and A** **Update w.r.t New Set of Variables**	**S, B**	S → a **| aA** B → a D → ddd A → aB **B → aA**	Body of Productions **S and B €{S, B, D, A } Ụ{ T }.** Add the variables **S and B to the new Set of V1**
4.	**S, B, D, A** **No update**	NiL	NIL	All productions are Considered

2. We eliminate the variables from Productions P1, that cannot be reached from the start variable. For this we draw the dependency graph for the variable set V1 = { S, A, B and D } found in step 1. Following is the dependency Graph :

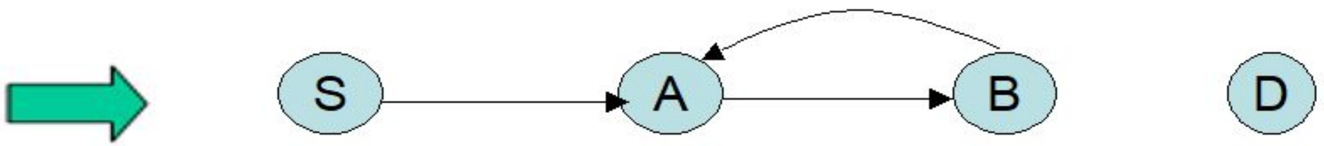
Productions :

$S \rightarrow a \mid aA$

$B \rightarrow a \mid aA$

$D \rightarrow ddd$

$A \rightarrow aB$



From the above Dependency graph, it is obsereved that the Vertex D is Not reachable from Vertex S, hence, Vertex D is ignored. So we have only the Verices S, A and B.

Finally, modified Grammar with Useful Varibales S, A and B and their associated productions are as follows :

$$S \rightarrow a \mid aA$$

$$A \rightarrow aB$$

$$B \rightarrow aA \mid a$$

Examples on Elimination of **Useless-productions**

Example -1

$$S \to AB$$
$$A \to aAA \mid \epsilon$$
$$B \to bBB \mid \epsilon$$

Example -2

$$S \to ASB \mid \epsilon$$
$$A \to aAS \mid a$$
$$B \to SbS \mid A \mid bb$$

Example -3

$$S \to 0A0 \mid 1B1 \mid BB$$
$$A \to C$$
$$B \to S \mid A$$
$$C \to S \mid \epsilon$$

Example -4

$$S \to 0A0 \mid 1B1 \mid BB$$
$$A \to C$$
$$B \to S \mid A$$
$$C \to S \mid \epsilon$$

Example - 5

$$S \to Aa \mid B$$
$$A \to a \mid bc \mid B,$$
$$B \to A \mid bb$$

Example - 6

$$S \to a \mid aA \mid B \mid C$$
$$A \to aB$$
$$B \to aA \mid a$$
$$C \to cCD$$
$$D \to ddd$$

2.5. Chomsky Normal Forms

A Grammar G is said to be in **Chomsky Normal form**, if all of its productions are of the form : **$A \rightarrow BC$** or **$A \rightarrow a$**, where **A, B,** and **C** are **all variables** and **a** is terminal symbol

Chomsky Normal Form

- The construction of CNF is performed through:

1. Arrangement of all bodies of length 2 or more to contain only variables.

2. Breaking bodies of length 3 or more into a cascade productions, where each one has a body consisting of 2 variables.

Note : Before the **Grammar is transferred to CNF notation**, the grammar must be **Simplified** interms of **Ɛ-productions, Unit productions and Useless productions.**

Algorithm to produce a grammar in CNF:

1. Eliminate **ε-productions, Unit productions** and **Useless symbols,** from the grammar Given grammar.

2. Elimination of terminals on the body of a productions.

   a) Add all productions of the form $A \rightarrow BC$ or $A \rightarrow a$ to **P1**, where **P1** is an intermediate set of productions b) Consider a production $A \rightarrow X_1X_2\ldots X_n$ from P.

   If **X_i is a terminal 'a'** then

   - Introduce a **new variable B_a for 'a'** and **replace each X_i** in **A** by B_a
   - Also Add **new production $B_a \rightarrow a$ to P1**.

c) Consider each production **$A \rightarrow X_1X_2\ldots X_n$,** where **$n >= 3$** and all **$X_i$'s are variables** from P1 -the intermedaite set of productions and introduce new variables and productions as per the following order to reduce the **production's body length to 2** to P1

$A \rightarrow X_1C_1$

$C_1 \rightarrow X_2C_2$

$C_2 \rightarrow X_3C_3$

$C_{n-2} \rightarrow X_{n-1}X_n$ ⇨ **Here C_1, C_2,.... C_{n-2} are all New variables**

Working with Example :

$$S \rightarrow aAD$$
$$A \rightarrow aB \mid bAB$$
$$B \rightarrow b$$
$$D \rightarrow d$$

Step -1. Simplification of the Grammar.

Grammar is already simplified.

Step -2. Elimination of terminals on the body of a productions. i.e arrangements of production bodies of length to 2 or more to contain only variables.

$$S \rightarrow aAD$$
$$A \rightarrow aB \mid bAB$$
$$B \rightarrow b$$
$$D \rightarrow d$$

$\Rightarrow$

P1 :

$$S \rightarrow B_aAD$$
$$A \rightarrow B_aB \mid B_bAB$$
$$B_a \rightarrow a$$
$$B_b \rightarrow b$$
$$B \rightarrow b$$
$$D \rightarrow d$$

Step -3. Breaking the bodies of length 3 or more into a cascade of productions.

P1:

$S \rightarrow B_a AD$

$A \rightarrow B_a B \mid B_b AB$

$B_a \rightarrow a$

$B_b \rightarrow b$

$B \rightarrow b$

$D \rightarrow d$

Final Grammar in CNF Notation

P1 :

$S \rightarrow B_a C_1$

$C_1 \rightarrow AD$

$A \rightarrow B_a B \mid B_b C2$

$C_2 \rightarrow AB$

$B_a \rightarrow a$

$B_b \rightarrow b$

$B \rightarrow b$

$D \rightarrow d$

Examples on **Chomsky Normal From - CNF**

Example -1

$$S \to AB$$
$$A \to aAA \mid \epsilon$$
$$B \to bBB \mid \epsilon$$

Example -2

$$S \to ASB \mid \epsilon$$
$$A \to aAS \mid a$$
$$B \to SbS \mid A \mid bb$$

Example -3

$$S \to 0A0 \mid 1B1 \mid BB$$
$$A \to C$$
$$B \to S \mid A$$
$$C \to S \mid \epsilon$$

Example -4

$$S \to 0A0 \mid 1B1 \mid BB$$
$$A \to C$$
$$B \to S \mid A$$
$$C \to S \mid \epsilon$$

Example - 5

$$S \to Aa \mid B$$
$$A \to a \mid bc \mid B,$$
$$B \to A \mid bb$$

Example - 6

$$S \to a \mid aA \mid B \mid C$$
$$A \to aB$$
$$B \to aA \mid a$$
$$C \to cCD$$
$$D \to ddd$$

END of UNIT-3