UNIT- III

CONTEXT FREE GRAMMARS

5

CONTEXT FREE GRAMMARS

After going through this chapter, you should be able to understand :

- Context free grammars
- Left most and Rightmost derivation of strings
- Derivation Trees
- Ambiguity in CFGs
- Minimization of CFGs
- Normal Forms (CNF & GNF)
- Pumping Lemma for CFLs
- Enumeration properties of CFLs

5.1 CONTEXT FREE GRAMMARS

A grammar $G = (V, T, P, S)$ is said to be a CFG if the productions of G are of the form :

$$A \rightarrow \alpha, \text{ where } \alpha \in (V \cup T)^*$$

The right hand side of a CFG is not restricted and it may be null or a combination of variables and terminals. The possible length of right hand sentential form ranges from 0 to ∞ i.e., $0 \leq |\alpha| \leq \infty$.

As we know that a CFG has no context neither left nor right. This is why, it is known as CONTEXT - FREE. *Many programming languages have recursive structure that can be defined by CFG's.*

Example 1 : Consider the grammar $G = (V, T, P, S)$ having productions :
$S \rightarrow aSa \mid bSb \mid \in$. Check the productions and find the language generated.

Solution :

Let $P_1 : S \rightarrow aSa$ (RHS is terminal variable terminal)

$P_2 : S \rightarrow bSb$ (RHS is terminal variable terminal)

$P_3 : S \rightarrow \in$ (RHS is null string)

Since, all productions are of the form $A \rightarrow \alpha$, where $\alpha \in (V \cup T)^*$, hence G is a CFG.

Language Generated :

$S \Rightarrow aSa \text{ or } bSb$

$\Rightarrow a^nSa^n \text{ or } b^nSb^n$ (Using n step derivation)

$\Rightarrow a^nb^mSb^ma^n \text{ or } b^na^mSa^mb^n$ (Using m step derivation)

$\Rightarrow a^nb^mb^ma^n \text{ or } b^na^ma^mb^n$ (Using $S \rightarrow \in$)

So, $L(G) = \{ww^R : w \in (a+b)^*\}$

Example 2 : Let G = (V, T, P, S) where V = { S , C } , T = { a , b }

P = { S → aCa
C → aCa | b
} S is the start symbol

What is the language generated by this grammar ?

Solution : Consider the derivation

$S \Rightarrow aCa \Rightarrow$ aba (By applying the 1st and 3rd production)

So, the string aba $\in L(G)$

Consider the derivation

$S \Rightarrow aCa$	By applying	$S \rightarrow aCa$
$\Rightarrow aaCaa$	By applying	$C \rightarrow aCa$
$\Rightarrow aaaCaaa$	By applying	$C \rightarrow aCa$
....		
....		
$\Rightarrow a^nCa^n$	By applying	$C \rightarrow aCa$ n times
$\Rightarrow a^nba^n$	By applying	$C \rightarrow b$

So, the language L accepted by the grammar G is $L(G) = \{ a^n ba^n \mid n \geq 1 \}$

i. e., the language L derived from the grammar G is "The string consisting of n number of a's followed by a 'b' followed by n number of a's.

Example 3 : What is the language generated by the grammar

$S \rightarrow 0A \mid \in$

$A \rightarrow 1S$

Solution : The null string $\in$ can be obtained by applying the production $S \rightarrow \in$ and the derivation is shown below :

$S \Rightarrow \in$ (By applying $S \rightarrow \in$)

Consider the derivation

$S \Rightarrow 0A$ (By applying $S \rightarrow 0A$)

$\Rightarrow 01S$ (By applying $A \rightarrow 1S$)

$\Rightarrow 010A$ (By applying $S \rightarrow 0A$)

$\Rightarrow 0101S$ (By applying $A \rightarrow 1S$)

$\Rightarrow 0101$ (By applying $S \rightarrow \in$)

So, alternatively applying the productions $S \rightarrow 0A$ and $A \rightarrow 1S$ and finally applying the production $S \rightarrow \in$, we get string consisting of only of 01's. So, both null string i.e., $\in$ and string consisting 01's can be generated from this grammar. So, the language generated by this grammar is

$$L = \{w \mid w \in \{01\}^*\} \; or \; L = \{(01)^n \mid n \geq 0\}$$

Example 4 : Show that the language $L = \{a^m b^n \mid m \neq n\}$ is context free.

Solution :

If it is possible to construct a CFG to generate this language then we say that the language is context free. Let us construct the CFG for the language defined. Assume that m = n i. e., m number of a's should be followed by m number of b's. The CFG for this can be

$$S \rightarrow aSb \mid \in \quad ....(1)$$

But, $L = \{a^m b^n \mid m \neq n\}$ means, a's should be followed by b's and number of a's should not be equal to number of b's i. e., $m \neq n$.

Let us see the different cases when m > n and when m < n.

Case 1 :

m > n : This case occurs if the number of a's are more compared to number of b's. The extra a's can be generated using the production

$$A \rightarrow aA \mid a$$

and the extra a's generated from this production should be appended towards left of the string generated from the production shown in production 1. This can be achieved by introducing one more production.

$$S_1 \rightarrow AS$$

So, even though from S we get n number of a's followed by n number of b's since it is preceded by a variable A from which we could generate extra a's, number of a's followed by number of b's are different.

Case 2 :

m < n : This case occurs if the number of b's are more compared to number of a's. The extra b's can be generated using the production.

$$B \rightarrow bB|b$$

and the extra b's generated from this production should be appended towards right of the string generated from the production shown in production (1). This can be achieved by introducing one more production

$$S_1 \rightarrow SB$$

The context free grammar $G = (V, T, P, S)$ where

$$V = \{S_1, S, A, B\} \;,\; T = \{a, b\}$$

$P = \{$

$$S_1 \rightarrow AS|SB$$
$$S \rightarrow aSb \mid \in$$
$$A \rightarrow aA|a$$
$$B \rightarrow bB|b$$

$\}$ S_1 is the start symbol

generates the language $L = \{ a^m b^n \mid m \neq n \}$. Since a CFG exists for the language, the language is context free.

Example 5 : Draw a CFG to generate a language consisting of equal number of a's and b's.

Solution : Note that initial production can be of the form

$$S \rightarrow aB \mid bA$$

If the first symbol is **'a'**, the second symbol should be a non - terminal from which we can obtain either **'b'** or one more **'a'** followed by two **B's** denoted by **aBB** or a **'b'** followed by S denoted by **bS**.

Note that from all these symbols definitely we obtain equal number of a's and b's. The productions corresponding to these can be of the form

$$B \rightarrow b|aBB|bS$$

On similar lines we can write A - productions as

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

from which we obtain a 'b' followed by either

1. 'a' or
2. a 'b' followed by AA's denoted by b AA or
3. symbol 'a' followed by S denoted by aS

The context free grammar G = (V, T, P, S) where

$$V = \{ S, A, B \},\ T = \{ a, b \}$$

$$P = \{ S \to aB \mid bA$$

$$A \to aS \mid bAA \mid a$$

$$B \to bS \mid aBB \mid b$$

$\}$ S is the start symbol

generates the language consisting of equal number of a's and b's.

Example 6 : Construct CFG for the language L which has all the strings which are all palindromes over $T = \{a, b\}$

Solution : As we know the strings are palindrome if they posses same alphabets from forward as well as from backward.

For example the string "LIRIL" is palindrome because

LIRIL	
→ read	← read
⇓	
It is the same !	

Since the language L is over $T = \{a, b\}$. We want the production rules to be build a's and b's. As $\in$ can be the palindrome, a can be palindrome even b can be palindrome. So we can write the production rules as

$$G = (\{S\}, \{a, b\}, P, S)$$

P can be
$$S \to a\ S\ a$$
$$S \to b\ S\ b$$
$$S \to a$$
$$S \to b$$
$$S \to \in$$

The string **abaaba** can be derived as

$$S \to aSa$$
$$\to ab\,Sba$$
$$\to ab\,a\,Sa\,ba$$
$$\to ab\,a \in a\,ba$$
$$\to ab\,a\,a\,ba$$

which is a palindrome.

Example 7 : Obtain a CFG to generate integers .

Solution :
The sign of a number can be '+' or '-' or $\in$. The production for this can be written as

$$S \rightarrow +|-|\in$$

A number can be formed from any of the digits 0, 1, 2, ...9. The production to obtain these digits can be written as $D \rightarrow 0|1|2|...|9$

A number N can be recursively defined as follows .

1. A number N is a digit D (i. e., $N \rightarrow D$)
2. The number N followed by digit D is also a number (i. e., $N \rightarrow ND$)

The productions for this recursive definition can be written as

$$N \rightarrow D$$
$$N \rightarrow ND$$

An integer number I can be a number N or the sign S of a number followed by number N. The production for this can be written as $I \rightarrow N|SN$

So, the grammar G to obtain integer numbers can be written as $G = (V, T, P, S)$ where

V = { D, S, N, I } , T = { +, -, 0, 1, 2, ... 9 }

p = {

$I \rightarrow N \mid SN$
$N \rightarrow D \mid ND$
$S \rightarrow + \mid - \mid \in$
$D \rightarrow 0 \mid 1 \mid 2 \mid .... \mid 9$

}

S = I which is the start symbol

Example 8 : Obtain the grammar to generate the language

$$L = \{ 0^m 1^m 2^n \mid m \geq 1 \ and \ \ n \geq 0 \}.$$

Solution : In the language $L = \{ 0^m 1^m 2^n \}$, if n = 0, the language L contains m number of 0's and m number of 1's. The grammar for this can be of the form

$$A \rightarrow 01 \mid 0A1$$

If n is greater than zero, the language L should contain m number of 0's followed by m number of 1's followed by one or more 2's i. e., the language generated from the non - terminal A should be followed by n number of 2's. So, the resulting productions can be written as

$$S \rightarrow A \mid S2$$
$$A \rightarrow 01 \mid 0A1$$

Thus, the grammar G to generate the language

$$L = \{ 0^m 1^m 2^n \mid m \geq 1 \text{ and } n \geq 0 \}$$

can be written as G = (V, T, P, S) where

V = { S, A } , T = { 0, 1, 2 }

P = {

$S \to A \mid S2$

$A \to 01 \mid 0A1$

} S is the start symbol

Example 9 : Obtain a grammar to generate the language $L = \{0^n 1^{n+1} \mid n \geq 0\}$.

Solution :

Note : It is clear from the language that total number of 1's will be one more than the total number of 0's and all 0's precede all 1's. So, first let us generate the string $0^n 1^n$ and add the digit 1 at the end of this string.

The recursive definition to generate the string $0^n 1^n$ can be written as

$$A \to 0A1 \mid \in$$

If the production $A \to 0A1$ is applied n times we get the sentential form as shown below.

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow \quad \ldots\ldots\ldots 0^n A1^n$$

Finally if we apply the production

$$A \to \in$$

the derivation starting from the start symbol A will be of the form

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 0^n A 1^n \Rightarrow 0^n 1^n$$

Thus, using these productions we get the string $0^n 1^n$. But, we should get the string $0^n 1^{n+1}$ i. e., an extra 1 should be placed at the end. This can be achieved by using the production

$$S \to A1$$

Note that from A we get string $0^n 1^n$ and 1 is appended at the end resulting in the string $0^n 1^{n+1}$.

So, the final grammar G to generate the language $L = \{ 0^n 1^{n+1} \mid n \geq 0 \}$ will be G = (V, T, P, S)

where V = { S, A } , T = { 0, 1 }

P = {

$S \to A1$

$A \to 0A1 \mid \in$

} S is the start symbol

Example 10 : Obtain the grammar to generate the language

$$L = \{w \mid n_a(w) = n_b(w)\}$$

Solution :

Note : $n_a(w) = n_b(w)$ means, number of a's in the string w should be equal to number of b's in the string w. To get equal number of a's and b's, we know that there are three cases :

1. There are no a's and b's present in the string w .
2. The symbol 'a' can be followed by the symbol 'b'
3. The symbol 'b' can be followed by the symbol a'

The corresponding productions for these three cases can be written as

$$S \to \in$$
$$S \to aSb$$
$$S \to bSa$$

Using these productions the strings of the form $\in$, ab, ba, abab, baba etc., can be generated. But, the stirngs such as abba, baab, etc., where the string starts and ends with the same symbol, can not be generated from these productions (even though they are valid strings).

So, to obtain in the producitons to generate such strings, let us divide the string into two substrings. For example, let us take the string 'abba'. This string can be split into two substrings 'ab' and 'ba'. The substring 'ab' can be generated from S and the derivation is shown below :

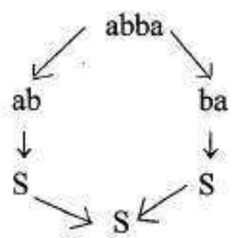
$$S \Rightarrow aSb \qquad \text{(By applying } S \to aSb \text{)}$$
$$\Rightarrow ab \qquad \text{(By applying } S \to \in \text{)}$$

Similarly, the substring 'ba' can be generated from S and the derivation is shown below :

$$S \Rightarrow bSa \qquad \text{(By applying } S \to bSa \text{)}$$
$$\Rightarrow ba \qquad \text{(By applying } S \to \in \text{)}$$

i. e., the first sub string 'ab' can be generated from S as shown in the first derivation and the second sub string 'ba' can also be generated from S as shown in second derivation.

So, to get the string 'abba' from S, perform the derivation in reverse order as shown below :



So, to get a string such that it starts and ends with the same symbol, the production to be used is

$$S \to SS$$

So, the final grammar to generate the language $L = \{ w \mid n_a(w) = n_b(w) \}$ is G = (V, T, P, S) where

$$V = \{ S \} \quad , \quad T = \{ a, b \}$$
$$P = \{ \; S \rightarrow \in$$
$$S \rightarrow aSb$$
$$S \rightarrow bSa$$
$$S \rightarrow SS$$
$$\}$$ S is the start symbol

5 . 2 LEFTMOST AND RIGHTMOST DERIVATIONS

Leftmost derivation :

If $G = (V, T, P, S)$ is a CFG and $w \in L(G)$ then a derivation $S \overset{*}{\underset{L}{\Rightarrow}} w$ is called leftmost derivation if and only if all steps involved in derivation have leftmost variable replacement only.

Rightmost derivation :

If $G = (V, T, P, S)$ is a CFG and $w \in L(G)$, then a derivation $S \overset{*}{\underset{R}{\Rightarrow}} w$ is called rightmost derivation if and only if all steps involved in derivation have rightmost variable replacement only.

Example 1 : Consider the grammar $S \rightarrow S + S | \; S * S | \, a | \, b$. Find leftmost and rightmost derivations for string $w = a * a + b$.

Solution :
Leftmost derivation for $w = a * a + b$

$S \underset{L}{\Rightarrow} S * S$ (Using $S \rightarrow S * S$)

$\underset{L}{\Rightarrow} a * S$ (The first left hand symbol is a, so using $S \rightarrow a$)

$\underset{L}{\Rightarrow} a * S + S$ (Using $S \rightarrow S + S$, in order to get $a + b$)

$\underset{L}{\Rightarrow} a * a + S$ (Second symbol from the left is a, so using $S \rightarrow a$)

$\underset{L}{\Rightarrow} a * a + b$ (The last symbol from the left is b, so using $S \rightarrow b$)

Rightmost derivation for $w = a * a + b$

$S \underset{R}{\Rightarrow} S * S$ (Using $S \to S * S$)

$\underset{R}{\Rightarrow} S * S + S$ (Since, in the above sentential form second symbol from the right is * so, we can not use $S \to a|b$. Therefore, we use $S \to S + S$)

$\underset{R}{\Rightarrow} S * S + b$ (Using $S \to b$)

$\underset{R}{\Rightarrow} S * a + b$ (Using $S \to a$)

$\underset{R}{\Rightarrow} a * a + b$ (Using $S \to a$)

Example 2 : Consider a CFG $S \to bA| aB,\ A \to aS| aAA| a,\ B \to bS| aBB| b$. Find leftmost and rightmost derivations for $w = aaabbabbba$.

Solution :

Leftmost derivation for $w = aaabbabbba$:

$S \Rightarrow aB$ (Using $S \to aB$ to generate first symbol of w)

$\Rightarrow aaBB$ (Since, second symbol is a, so we use $B \to aBB$)

$\Rightarrow aaaBBB$ (Since, third symbol is a, so we use $B \to aBB$)

$\Rightarrow aaabBB$ (Since fourth symbol is b, so we use $B \to b$)

$\Rightarrow aaabbB$ (Since, fifth symbol is b, so we use $B \to b$)

$\Rightarrow aaabbaBB$ (Since, sixth symbol is a, so we use $B \to aBB$)

$\Rightarrow aaabbabB$ (Since, seventh symbol is b, so we use $B \to b$)

$\Rightarrow aaabbabbS$ (Since, eighth symbol is b, so we use $B \to bS$)

$\Rightarrow aaabbabbbA$ (Since, ninth symbol is b, so we use $S \to bA$)

$\Rightarrow aaabbabbba$ (Since, the tenth symbol is a, so using $A \to a$)

Rightmost derivation for $w = aaabbabbba$

$S \Rightarrow aB$ (Using $S \to aB$ to generate first symbol of w)

$\Rightarrow aaBB$ (We need a as the rightmost symbol and second symbol from the left side, so we use $B \to aBB$)

$\Rightarrow aaBbS$ (We need a as rightmost symbol and this is obtained from A only, we use $B \to bS$)

$\Rightarrow aaBbbA$ (Using $S \to bA$)

$\Rightarrow aaBbba$ (Using $A \to a$)

$\Rightarrow aaaBBbba$ (We need b as the fourth symbol from the right)

$\Rightarrow aaaBbbba$ (Using $B \to b$)

$\Rightarrow aaabSbbba$ (Using $B \to bS$)

$\Rightarrow aaabbAbbba$ (Using $S \rightarrow bA$)

$\Rightarrow aaabbabbba$ (Using $A \rightarrow a$)

5.3 DERIVATION TREES

Let $G = (V, T, P, S)$ is a CFG. Each production of G is represented with a tree satisfying the following conditions:

1. If $A \rightarrow \alpha_1\alpha_2\alpha_3\ldots\alpha_n$ is a production in G, then A becomes the parent of nodes labeled $\alpha_1, \alpha_2, \alpha_3, \ldots \alpha_n$, and
2. The collection of children from left to right yields $\alpha_1\alpha_2\alpha_3\ldots\alpha_n$

Example : Consider a CFG $S \rightarrow S + S \mid S * S \mid a \mid b$ and construct the derivation trees for all productions.

Solution :

For the production $S \rightarrow S + S$ $\Rightarrow$

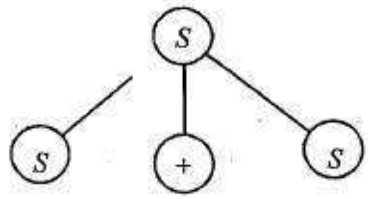


Figure (a)

For the production $S \rightarrow S * S$ $\Rightarrow$

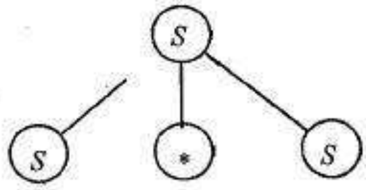
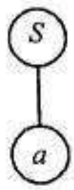


Figure (b)

For the production $S \rightarrow a$ $\Rightarrow$



For the production $S \rightarrow b$ $\Rightarrow$

Figure (c) **Figure (d)**

If $w \in L(G)$ then it is represented by a tree called **derivation tree or parse tree** satisfying the following conditions :

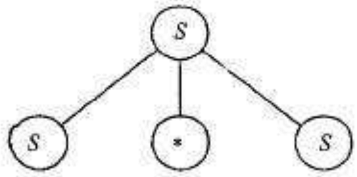
1. The root has label S (the starting symbol),
2. The all internal vertices (or nodes) are labeled with variables,
3. The leaves or terminal nodes are labeled with $\in$ or terminal symbols,
4. If $A \rightarrow \alpha_1\alpha_2\alpha_3 \ldots \alpha_n$ is a production in G, then A becomes the parent of nodes labeled $\alpha_1, \alpha_2, \alpha_3, \ldots \alpha_n$, and
5. The collection of leaves from left to right yields the string w.

Example 1 : Consider the grammar $S \rightarrow S + S | S * S | a | b$. Construct derivation tree for string $w = a * b + a$.

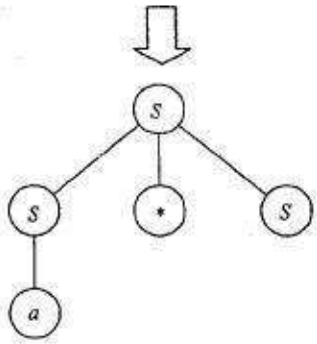
Solution : The derivation tree or parse tree is shown in below figure.

Leftmost derivation for $w = a * b + a$:

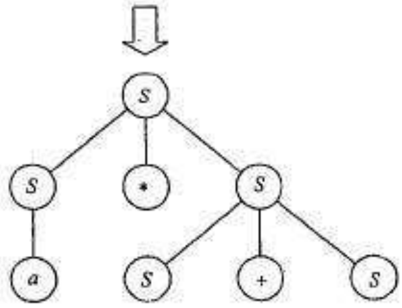
$S \Rightarrow S * S$ (Using $S \rightarrow S * S$)



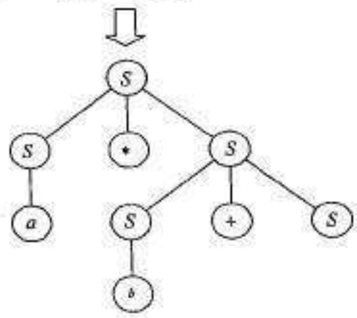
$\Rightarrow a * S$ (The first left hand symbol is a, so using $S \rightarrow a$)



$\Rightarrow a * S + S$ (Using $S \rightarrow S + S$, in order to get $b + a$)



$\Rightarrow a * b + S$ (Second symbol from the left is b, so using $S \rightarrow b$)



$\Rightarrow a * b + a$ (The last symbol from the left is a, so using $S \rightarrow a$)

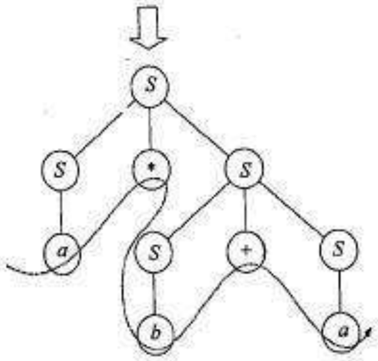


Figure : Parse tree for $a * b + a$

Example 2 : Consider a grammar G having productions $S \rightarrow aAS| a,\ A \rightarrow SbA| SS| ba$.

Show that $S \overset{*}{\Rightarrow} aabbaa$ and construct a derivation tree whose yield is aabbaa.

Solution :

$S \Rightarrow aAS$

$\Rightarrow aSbAS$

$\Rightarrow$ aabAS

$\Rightarrow aabbaS$

$\Rightarrow aabbaa$

Hence, $S \overset{*}{\Rightarrow} aabbaa$

Parse tree is shown in figure .

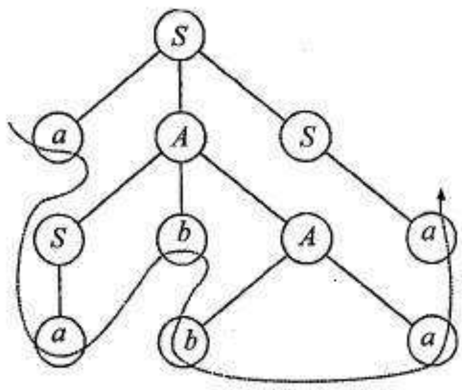


Figure : Parse tree yielding $aabbaa$

Example 3 : Consider the grammar G whose productions are

$$S \rightarrow 0B | 1A,\ A \rightarrow 0 | 0S | 1AA,\ B \rightarrow 1 | 1S | 0BB.$$ Find

(a) Leftmost and (b) Rightmost derivation for string 00110101, and construct derivation tree also.

Solution :

(a) Leftmost derivation :

$S \Rightarrow 0B \Rightarrow 00BB$

$\Rightarrow 001B \Rightarrow 0011S$

$\Rightarrow 00110B \Rightarrow 001101S$

$\Rightarrow 0011010B \Rightarrow 00110101$

(b) Rightmost derivation :

$S \Rightarrow 0B \Rightarrow 00BB$

$\Rightarrow 00B1 \Rightarrow 001S1$

$\Rightarrow 0011A1 \Rightarrow 00110S1$

$\Rightarrow 001101A1 \Rightarrow 00110101$

(c) Derivation tree :

Derivation tree is shown in below figure.

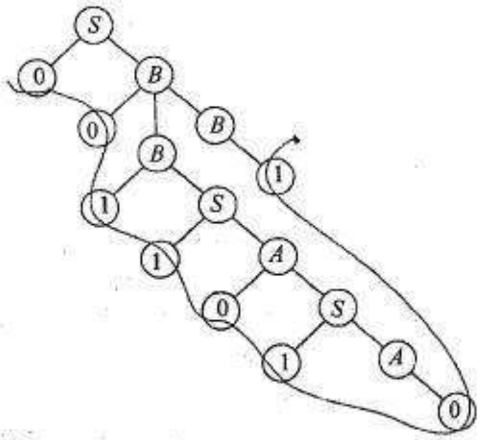


Figure : Derivation tree for 00110101

5.4 AMBIGUITY IN CFGs

A grammar G is *ambiguous if there exists some string* $w \in L(G)$ *for which there are two or more distinct derivation trees, or there are two or more distinct leftmost derivations.*

Example 1 : Consider the CFG $S \to S + S \mid S * S \mid a \mid b$ and string $w = a * a + b$, and derivations as follows:

Solution :

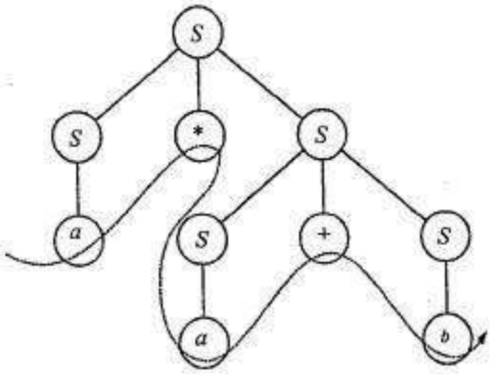
First leftmost derivation for $w = a * a + b$

$S \Rightarrow S * S$ (Using $S \to S * S$)
$\Rightarrow a * S$ (Using $S \to a$)
$\Rightarrow a * S + S$ (Using $S \to S + S$)
$\Rightarrow a * a + S$ (Using $S \to a$)
$\Rightarrow a * a + b$ (Using $S \to b$)

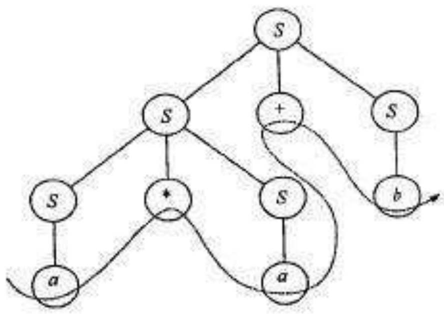
Second leftmost derivation for $w = a * a + b$

$S \Rightarrow S + S$ (Using $S \to S + S$)
$\Rightarrow S * S + S$ (Using $S \to S * S$)
$\Rightarrow a * S + S$ (Using $S \to a$)
$\Rightarrow a * a + S$ (Using $S \to a$)
$\Rightarrow a * a + b$ (Using $S \to b$)

Two distinct parse trees are shown in figure (a) and figure (b)



Figure(a) Parse tree for $a * a + b$



Figure(b) Parse tree for $a * a + b$

Since, there are two distinct leftmost derivations (two parse trees) for string w, hence w is ambiguous and there is ambiguity in grammar G.

Example 2 : Show that the following grammars are ambiguous.

(a) $S \to SS \mid a \mid b$

(b) $S \to A \mid B \mid b,\ A \to aAB \mid ab,\ B \to abB \mid \in$

Solution :

(a) Consider the string $w = bbb$, two leftmost derivations are as follows :

$S \underset{L}{\Rightarrow} SS$	(Using $S \rightarrow SS$)	$S \underset{L}{\Rightarrow} SS$	(Using $S \rightarrow SS$)
$S \underset{L}{\Rightarrow} bS$	(Using $S \rightarrow b$)	$\underset{L}{\Rightarrow} SSS$	(Using $S \rightarrow SS$)
$\underset{L}{\Rightarrow} bSS$	(Using $S \rightarrow SS$)	$\underset{L}{\Rightarrow} bSS$	(Using $S \rightarrow b$)
$\underset{L}{\Rightarrow} bbS$	(Using $S \rightarrow b$)	$\underset{L}{\Rightarrow} bbS$	(Using $S \rightarrow b$)
$\underset{L}{\Rightarrow} bbb$	(Using $S \rightarrow b$)	$\underset{L}{\Rightarrow} bbb$	(Using $S \rightarrow b$)

Two parse trees are shown in figure(a) and figure(b).

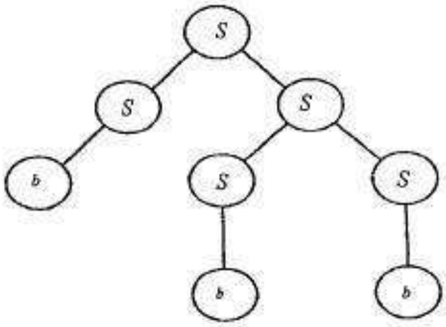


Figure (a) Parse tree for bbb

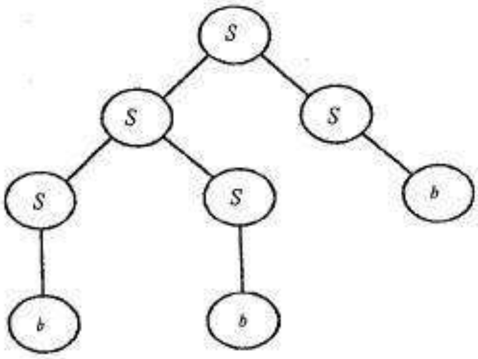


Figure (b) Parse tree for bbb

So, the given grammar is ambiguous.

(b) Consider the string $w = ab$, we get two leftmost derivations for w as follows :

$S \underset{L}{\Rightarrow} A$		$S \underset{L}{\Rightarrow} B$	
$\underset{L}{\Rightarrow} ab$	(Using $A \rightarrow ab$)	$\underset{L}{\Rightarrow} abB$	(Using $B \rightarrow abB$)
		$\underset{L}{\Rightarrow} ab$	(Using $B \rightarrow \in$)

Two parse trees are shown in figure (c) and figure (d).

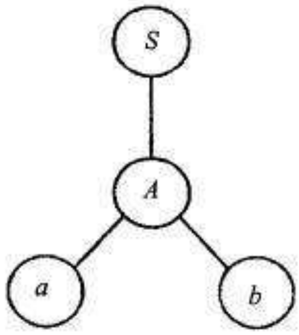


Figure (c) Parse tree for $w = ab$

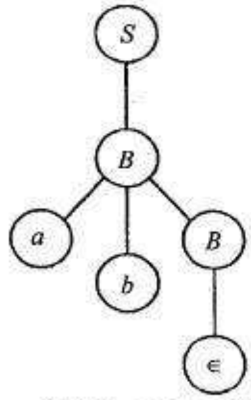


Figure (d) Parse tree for $w = ab$

So, the given grammar is ambiguous.

5.4.1 Removal of Ambiguity

5.4.1.1 Left Recursion

A grammar can be changed from one form to another accepting the same language. If a grammar has left recursive property, it is undesirable and left recursion should be eliminated. The left recursion is defined as follows.

Definition : A grammar G is said to be left recursive if there is some non terminal A such that $A \Rightarrow^{+} A\alpha$. In other words, in the derivation process starting from any non - terminal A, if a sentential form starts with the same non - terminal A, then we say that the grammar is having left recursion.

Elimination of Left Recursion

The left recursion in a grammar G can be eliminated as shown below. Consider the A - production of the form $A \rightarrow A\alpha_1 | A\alpha_2 | A\alpha_3 \ldots\ldots\ldots A\alpha_n | \beta_1 | \beta_2 | \beta_3 \ldots\ldots \beta_m$

where β_i's do not start with A. Then the A productions can be replaced by

$$A \rightarrow \beta_1 A^1 | \beta_2 A^1 | \beta_3 A^1 | \ldots\ldots\ldots \beta_m A^1$$

$$A^1 \rightarrow \alpha_1 A^1 | \alpha_2 A^1 | \alpha_3 A^1 | \ldots\ldots\ldots | \alpha_n A^1 | \in$$

Note that α_i's do not start with A^1.

Example 1 : Eliminate left recursion from the following grammar

$$E \rightarrow E + T | T$$

$$T \rightarrow T * F | F$$

$$F \rightarrow (E) \ | id$$

Solution : The left recursion can be eliminated as shown below :

Given	Substitution	Without left recursion
$A \to A\alpha_i \| \beta_i$		$A \to \beta_i A^1$ and $A^1 \to \alpha_i A^1 \| \in$
$E \to E+T \| T$	$A = E$ $\alpha_1 = +T$ $\beta_1 = T$	$E \to TE^1$ $E^1 \to +TE^1 \| \in$
$T \to T*F \| F$	$A = T$ $\alpha_1 = *F$ $\beta_1 = F$	$T \to FT^1$ $T^1 \to *FT^1 \| \in$
$F \to (E) \| id$	Not applicable	$F \to (E) \| id$

The grammar obtained after eliminating left recursion is

$$E \to TE^1$$
$$E^1 \to +TE^1 \mid \in$$
$$T \to FT^1$$
$$T^1 \to *FT^1 \mid \in$$
$$F \to (E) \mid id$$

Example 2 : Eliminate left recursion from the following grammar

$$S \to Ab \mid a$$
$$A \to Ab \mid Sa$$

Solution :

The non terminal S, even though is not having immediate left recursion, it has left recursion because $S \Rightarrow Ab \Rightarrow Sab$ i. e., $S \Rightarrow^+ Sab$. Substituting for S in the A - production can eliminate the indirect left recursion from S. So, the given grammar can be written as

$$S \to Ab|a$$
$$A \to Ab \mid Aba \mid aa$$

Now, A - production has left recursion and can be eliminated as shown below :

Given	Substitution	Without left recursion
$A \to A\alpha_i \mid \beta_i$		$A \to \beta_i A^1$ *and* $A^1 \to \alpha_i A^1 \mid \in$
$S \to Ab \mid a$	Not applicable	$S \to Ab \mid a$
$A \to Ab \mid Aba \mid aa$	A=A $\alpha_1 = b$ $\alpha_2 = ba$ $\beta_1 = aa$	$A \to aaA^1$ $A^1 \to bA^1 \mid baA^1 \mid \in$

The grammar obtained after eliminating left recursion is

$$S \to Ab \mid a$$
$$A \to aaA^1$$
$$A^1 \to bA^1 \mid baA^1 \mid \in$$

5.4.1.2 Left Factoring

Definition :

Two or more productions of a variable A of the grammar $G = (V, T, P, S)$ are said to have left factoring if A - productions are of the form $A \to \alpha\beta_1 \mid \alpha\beta_2 \mid .... \mid \alpha\beta_n$, where $\beta_i \in (V \cup T)^*$ and does not start (prefix) with α . All these A - productions have common left factor α .

Elimination of Left Factoring

Let the variable A has (left factoring) productions as follows :

$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid ..... \mid \alpha\beta_n \mid \gamma_1 \mid \gamma_2 \mid ... \mid \gamma_m$, where $\beta_1, \beta_2, \beta_3 ..... \beta_n$ and $\gamma_1, \gamma_2, ... \gamma_m$ do not contain α as a prefix, then we replace A - productions by :

$A \to \alpha A' \mid \gamma_1 \mid \gamma_2 \mid ..... \mid \gamma_m$, where $A' \to \beta_1 \mid \beta_2 \mid ..... \mid \beta_n$.

Example : Consider the grammar $S \to aSa \mid aa$ and remove the left factoring (if any).

Solution :

$S \to aSa$ and $S \to aa$ have $\alpha = a$ as a left factor, so removing the left factoring, we get the productions : $S \to aS'$, $S' \to Sa \mid a$.

The problem associated with left factoring and left recursive grammars is back - tracking. We can find α as a prefix in RHS in many ways and a string having α as a prefix can create problem. In worst condition, to get appropriate remaining part of the string we have to search the entire production list. We take the first production, if it is not suitable then take second production and so on. This situation is known as back - tracking . For example, consider the above S - productions $S \to aSa \mid aa$ and a string w=aa. We have choice of the both productions looking at the first symbol on the RHS.

Iteration First :

$S \Rightarrow aSa$

$\Rightarrow aaaa \neq w$

Iteration Second :

$S \Rightarrow aa = w$

So, if we follow the iteration first, then we can not get the string w and we will have to return to the iteration second i. e. the starting symbol. The problem, in which we proceed further and do not get the desired string and we come to the previous step, is known as back - tracking. This problem is a fundamental problem in designing of compilers (parser).

Procedure for Removal of Ambiguity :

We have no obvious rule or method defined for removing ambiguity as we have for left recursion and left factoring. So, we will have to concentrate on heuristic approach most of the time.

Let us consider the ambiguous grammar $S \to S+S \mid S*S \mid a \mid b$. Now, if we analyze the productions, then we find that two productions are left recursive. So, first we try to remove the left recursion.

$S \to S+S$ and $S \to S*S$ is replaced by $S \to aS' \mid bS'$, $S' \to +SS' \mid *SS' \mid \in$

Now, we check the derivation for ambiguous string $w = a*a+a$. We have only one left most derivation or only one parse tree given as follows :

$$
\begin{aligned}
S &\Rightarrow aS' \\
&\Rightarrow a*SS' \\
&\Rightarrow a*aS'S' \\
&\Rightarrow a*a+SS'S' \\
&\Rightarrow a*a+aS'S'S' \\
&\Rightarrow a*a+a\in S'S' \\
&\Rightarrow a*a+a\in S' \\
&\Rightarrow a*a+a\in \; (\equiv a*a+a)
\end{aligned}
$$

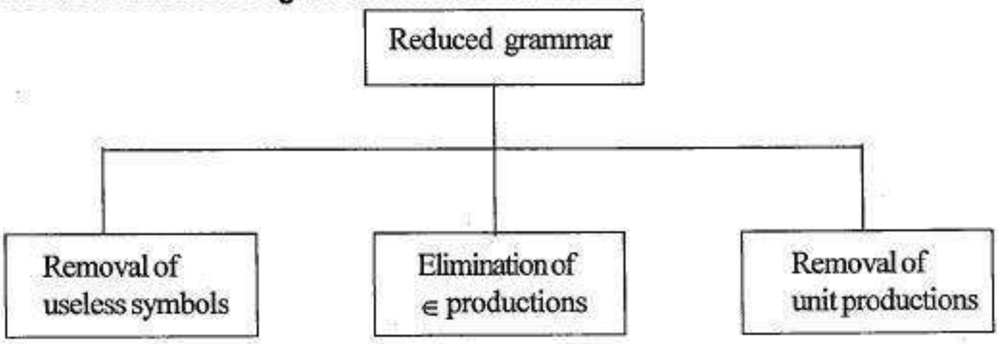
So, we conclude that removal of left recursion (and left factoring also) helps in removal of ambiguity of the ambiguous grammars.

5.5 MINIMIZATION OF CFGs

As we have seen various languages can effectively be represented by context free grammar. All the grammars are not always optimized. That means grammar may consists of some extra symbols (non - terminals). Having extra symbols unnecessary increases the length of grammar. Simplification of grammar means reduction of grammar by removing useless symbols. The properties of reduced grammar are given below :

1. Each variable (i. e. non - terminal) and each terminal of G appears in the derivation of some word in L.
2. There should not be any production as $X \to Y$ where X and Y are non - terminals.
3. If $\in$ is not in the language L then there need not be the production $X \to \in$.

We see the reduction of grammar as shown below :



5.5.1 Removal of useless symbols

Definition : A symbol X is useful if there is a derivation of the form

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* w$$

Otherwise, the symbol X is useless. Note that in a derivation, finally we should get string of terminals and all these symbols must be reachable from the start symbol S. Those symbols and productions which are not at all used in the derivation are useless.

Theorem 5.5.1 : Let G = (V, T, P, S) be a CFG. We can find an equivalent grammar $G_1 = (V_1, T_1, P_1, S)$ such that for each A in $(V_1 \cup T_1)$ there exists α and β in $(V_1 \cup T_1)^*$ and x in T^+ for which $S \Rightarrow^* \alpha A \beta \Rightarrow^* x$.

Proof : The grammar G_1 can be obtained from G in two stages.

STAGE 1 :

Obtain the set of variables and productions which derive only string of terminals i. e., Obtain a grammar $G_1 = (V_1, T_1, P_1, S)$ such that V_1 contains only the set of variables A for which $A \Rightarrow^* x$ where $x \in T^+$.

The algorithm to obtain a set of variables from which only string of terminals can be derived is shown below.

Step 1 : [Initialize old _ variables denoted by ov to ϕ]

$$ov = \phi$$

Step 2 : Take all productions of the form $A \to x$ where $x \in T^+$ i. e., if the R. H. S of the production contains only string of terminals consider those productions and corresponding non terminals on L. H. S are added to new _ variables denoted by nv. This can be expressed using the following statement :

$$nv = \{ A \mid A \to x \quad and \quad x \in T^+ \}$$

Step 3 : Compare ov and nv. As long as the elements in ov and nv are not equal, repeat the following statements. Otherwise goto step 4.

a. [Copy new_varialbes to old _variables]

ov = nv

b. Add all the elements in ov to nv. Also add the variables which derive a string consisting of terminals and non terminals which are in ov.

$$nv = ov \cup \{ A | A \to y \ and \ \ y \in (ov \cup T)^* \}$$

Step 4 : When the loop is terminated, nv (or ov) contains all those non terminals from which only the string of terminals are derived and add those variables to V_1.

i. e., $V_1 = ov$

Step 5 : [Terminate the algorithm]

return V_1

Note that the variable V_1 contains only those variables from which string of terminals are obtained. The productions used to obtain V_1 are added to P_1 and the terminals in these productions are added to T_1. The grammar $G_1 = (V_1, T_1, P_1, S)$ contains those variables A in V_1 such that $A \Rightarrow^* x$ for some x in T^+. Since each derivation in G_1 is a derivation of G, L $(G_1) = L(G)$.

STAGE 2 :

Obtain the set of variables and terminals which are reachable from the start symbol and the corresponding productions. This can be obtained as shown below :

Given a CFG $G = (V, T, P, S)$, we can find an equivalent grammar $G_1 = (V_1, T_1, P_1, S)$ such that for each X in $V_1 \cup T_1$ there exists α such that $S \Rightarrow^* \alpha$ and X is a symbol in α i. e., if X is a variable $X \in V_1$ and if X is terminal $X \in T_1$. Each symbol X in $V_1 \cup T_1$ is reachable from the start symbol S. The algorithm for this is shown below.

$V_1 = \{S\}$
For each A in V_1
    if $A \rightarrow \alpha$ then
        Add the variables in $A \ to \ V_1$
        Add the terminals in $\alpha \ to \ T_1$
    Endif
Endfor

Using this algorithm all those symbols (whether variables or terminals) that are not reachable from the start symbol are eliminated. The grammar G_1 does not contain any useless symbol or production. For each $X \in L(G_1)$ there is a derivation.

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* x$$

Using these two steps we can effectively find G_1 such that $L(G) = L(G_1)$ and the two grammars G and G_1 are equivalent.

Example 1 : Eliminate the useless symbols in the grammar

$S \rightarrow aA \mid bB$
$A \rightarrow aA \mid a$
$B \rightarrow bB$
$D \rightarrow ab \mid Ea$
$E \rightarrow aC \mid d$

Solution :

Stage 1 : Applying the algorithm shown in stage 1 of the theorem 5.5.1, we can obtain a set of variables from which we get only string of terminals and is shown below.

oV	nV	Productions
ϕ	A, D, E	A → a D → ab E → d
A, D, E	A, D, E, S	S → aA A → aA D → Ea
A, D, E, S	A, D, E, S	

The resulting grammar $G_1 = (V_1, T_1, P_1, S)$ where

$V_1 = \{A, D, E, S\}$

$T_1 = \{a, b, d\}$

$P_1 = \{$

A → a | aA
D → ab | Ea
E → d
S → aA

} S is the start symbol

contains all those variables in V_1 such that $A \Rightarrow^+ w$ where $w \in T^+$.

Stage 2 :

Applying the algorithm given in stage 2 of the theorem 5.5.1, we obtain the symbols such that each symbol X is reachable from the start symbol S as shown below.

P_1	T_1	V_1
-	-	S
$S \to aA$	a	S, A
$A \to a \mid aA$	a	S, A

The resulting grammar $G_1 = (V_1, T_1, P_1, S)$ where $V_1 = \{S, A\}$, $T_1 = \{a\}$

$P_1 = \{$

$S \to aA$

$A \to a \mid aA$

} S is the start symbol

such that each symbol X in $(V_1 \cup T_1)$ has a derivation of the form $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$.

Example 2 : Eliminate the useless symbols in the grammar

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

Solution :

Stage 1 :

Applying the algorithm shown in stage1 of theorem 5.5.1 , we can obtain a set of variables from which we get only string of terminals and is shown below.

ov	nv	Productions
ϕ	S, B, D	S → a B → a D → ddd
S, B, D	S, B, D, A	S → Bb A → aB
S, B, D, A	S, B, D, A	S → aA B → Aa

The resulting grammar $G_1 = (V_1, T_1, P_1, S)$ where

$V_1 = \{S, B, D, A\}$

$T_1 = \{a, b, d\}$

$P_1 = \{$

$$\begin{array}{lcl} S & \rightarrow & a \mid Bb \mid aA \\ B. & \rightarrow & a \mid Aa \\ D & \rightarrow & ddd \\ A & \rightarrow & aB \end{array}$$

$\}$

S is the start symbol contains all those variables in V_1 such that $A \Rightarrow^+ w$.

Stage 2 :

Applying the algorithm given in stage 2 of the theorem 5.5.1, we obtain the symbols such that each symbol X is reachable from the start symbol S as shown below.

P_1	T_1	V_1
-	-	S
S → a | Bb | Aa	a, b	S, A, B
A → a B	a, b	S, A, B
B → a | A a	a, b	S, A, B

The resulting grammar $G_1 = (V_1, T_1, P_1, S)$ where

V_1 = { S, A, B }

T_1 = { a, b }

P_1 = {

S → a | Bb | aA

A → aB

B → a | Aa

} S is the start symbol

such that each symbol X in $(V_1 \cup T_1)$ has a derivation of the form $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$.

5.5.2 Eliminating $\in$ - productions

A production of the form $A \to \in$ is undesirable in a CFG, unless an empty string is derived from the start symbol. Suppose, the language generated from a grammar G does not derive any empty string and the grammar consists of $\in$ - productions. Such $\in$ - productions can be removed. An $\in$ - production is defined as follows :

Definition 1 : Let G = (V, T, P, S) be a CFG. A production in P of the form

$$A \to \in$$

is called an $\in$ - production or NULL production. After applying the production the variable A is erased. For each A in V, if there is a derivation of the form

$$A \Rightarrow^* \in$$

then A is a nullable variable.

Example : Consider the grammar

S → ABCa | bD

A → BC | b

B → b | $\in$

$C \rightarrow c \mid \in$
$D \rightarrow d$

In this grammar, the productions

$$B \rightarrow \in$$
$$C \rightarrow \in$$

are $\in$ - productions and the variables B, C are nullable variables. Because there is a production

$$A \rightarrow BC$$

and both B and C are nullable variables, then A is also a nullable variable.

Definition 2 : Let G = (V, T, P, S) be a CFG where V is set of variables, T is set of terminals, P is set of productions and S is the start symbol. A nullable variable is defined as follows.

1. If $A \rightarrow \in$ is a production in P, then A is a nullable variable.
2. If $A \rightarrow B_1\ B_2........B_n$ is a production in P, and if $B_1\ B_2........B_n$ are nullable variables, then A is also a nullable variable
3. The variables for which there are productions of the form shown in step 1 and step 2 are nullable variables.

Even though a grammar G has some $\in$ - productions, the language may not derive a language containing empty string. So, in such cases, the $\in$ - productions or NULL productions are not needed and they can be eliminated.

Theorem 5.5.2 : Let G = (V, T, P, S) where $L(G) \neq \in$. We can effectively find an equivalent grammar G_1 with no $\in$- productions such that $L(G_1) = L(G) - \in$.

Proof : The grammar G_1 can be obtained from G in two steps.

Step 1 : Find the set of nullable variables in the grammar G using the following algorithm.

```
ov = ϕ
nv = {A|A→∈ }
while ( ov ! = n v)
{
        ov = nv
        nv = ov ∪ { A | A→ α and  α ∈ ov* }
}
V = ov
```

Once the control comes out of the while loop, the set V contains only the nullable variables.

Step 2 : Construction of productions P_1 . Consider a production of the form

$$A \rightarrow X_1X_2X_3 \text{..........} X_n, \; n \geq 1$$

where each X_i is in $(V \cup T)$. In a production, take all possible combinations of nullable variables and replace the nullable variables with $\in$ one by one and add the resulting productions to P_1 . If the given production is not an $\in$ – production, add it to P_1 .

Suppose, A and B are nullable variables in the production, then

1. First add the production to P_1 .
2. Replace A with $\in$ and add the resulting production to P_1
3. Replace B with $\in$ and the resulting production to P_1 .
4. Replace A and B with $\in$ and add the resulting production to P_1 .
5. If all symbols on right side of production are nullable variables, the resulting production is an $\in$ production and do not add this to P_1 .

Thus, the resulting grammar G_1 obtained, generates the same language as generated by G without $\in$ and the proof is straight forward .

Example 1 : Eliminate all $\in$ - productions from the grammar

S → ABCa | bD
A → BC | b
B → b | $\in$
C → c | $\in$
D → d

Solution :

Step 1 :

Obtain the set of nullable variables from the grammar. This can be done using step 1 of theorem 5.5.2 as shown below.

OV	nv	Productions
ϕ	B, C	B → $\in$ C → $\in$
B, C	B, C, A	A → BC
B, C, A	B, C, A	-

V = { B, C, A } are all nullable variables.

Step 2 : Construction of productions P_1.

Productions	Resulting productions (P_1)
$S \rightarrow ABCa$	$S \rightarrow ABCa$ | BCa |ACa | ABa | Ca | A a | Ba | a
$S \rightarrow bD$	$S \rightarrow bD$
$A \rightarrow BC \mid b$	$A \rightarrow BC$| B | C | b
$B \rightarrow b \mid \in$	$B \rightarrow b$
$C \rightarrow c \mid \in$	$C \rightarrow c$
$D \rightarrow d$	$D \rightarrow d$

The grammar $G_1 = (V_1, T_1, P_1, S)$ where

$$V_1 = \{ S, A, B, C, D \}$$

$$T_1 = \{ a, b, c, d \}$$

$$P_1 = \{ \quad S \rightarrow ABCa \mid BCa \mid ACa \mid ABaCa \mid Aa \mid Ba \mid a \mid bD$$
$$A \rightarrow BC \mid B \mid C \mid b$$
$$B \rightarrow b$$
$$C \rightarrow c$$
$$D \rightarrow d$$
$$\} \quad \text{S is the start symbol}$$

Example 2 : Eliminate all $\in$- productions from the grammar

$$S \rightarrow BAAB$$
$$A \rightarrow 0A2 \mid 2A0 \mid \in$$
$$B \rightarrow AB \mid 1B \mid \in$$

Solution :

Step 1 : Obtain the set of nullable variables from the grammar. This can be done using step 1 of theorem 5.5.2 as shown below.

ov	nv	Productions
ϕ	A, B	$A \rightarrow \in$ $B \rightarrow \in$
A, B	A, B, S	$A \rightarrow BAAB$
A, B, S	A, B, S	-

V = { S, A, B } are all nullable variables.

Step 2 : Construction of productions P_1. Add a non $\in$- production in P to P_1. Take all the combinations of nullable variables in a production, delete subset of nullable variables one by one and add the resulting productions to P_1.

Productions			Resulting productions (P_1)
S	$\rightarrow$	BAAB	S $\rightarrow$ BAAB | AAB | BAB | BAA | AB | BB | BA | AA | A | B
A	$\rightarrow$	0A 2	A $\rightarrow$ 0A2 | 02
A	$\rightarrow$	2A0	A $\rightarrow$ 2A0 | 20
B	$\rightarrow$	AB	B $\rightarrow$ AB | B | A
B	$\rightarrow$	1 B	B $\rightarrow$ 1 B | 1

We can delete the productions of the form A $\rightarrow$ A. In P_1, the production $B \rightarrow B$ can be deleted and the final grammar obtained after eliminating $\in$ -productions is shown below.

The grammar $G_1 = (V_1, T_1, P_1, S)$ where

V_1 = { S, A, B, C, D }

T_1 = { a, b, c, d }

P_1 = { S $\rightarrow$ BAAB | AAB | BAB | BAA | AB | BB | BA | AA | A | B
A $\rightarrow$ 0A2 | 02 | 2A0 | 20
B $\rightarrow$ AB | A | 1B | 1
} S is the start symbol

5.5.3 Eliminating unit productions

Consider the production $A \rightarrow B$. The left hand side of the production and right hand side of the production contains only one variable. Such productions are called unit productions. Formally, a unit production is defined as follows.

Definition : Let G = (V, T, P, S) be a CFG. Any production in G of the form

$$A \rightarrow B$$

where A, $B \in V$ is a unit production.
In any grammar, the unit productions are undesirable. This is because one variable is simply replaced by another variable.

Example : Consider the productions.

$$A \rightarrow B$$
$$B \rightarrow aB \mid b$$

In this example,

$$B \rightarrow aB$$
$$B \rightarrow b$$

are non unit productions. Since B is generated from A, whatever is generated by B, the same things can be generated from A also. So, we can have

$A \rightarrow aB$

$A \rightarrow b$ and the production $A \rightarrow B$ can be deleted.

Theorem 5.5.3 : Let G = (V, T, P, S) be a CFG and has unit productions and no $\in$– productions. An equivalent grammar G_1 without unit productions can be obtained such that $L(G) = L(G_1)$ i. e., any language generated by G is also generated by G_1. But , the grammar G_1 has no unit productions.

Proof :

A unit production in grammar G can be eliminated using the following steps :

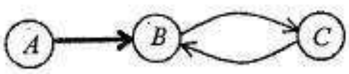
1. Remove all the productions of the form $A \rightarrow A$
2. Add all non unit productions to P_1.
3. For each variable A find all variables B such that

   $$A \Rightarrow^* B$$

   i. e., in the derivation process from A, if we encounter only one variable in a sentential form say B (no terminals should be there), obtain all such variables.
4. Obtain a dependency graph. For example, if we have the productions

   $$A \rightarrow B$$
   $$B \rightarrow C$$
   $$C \rightarrow B$$

   the dependency graph will be of the form



5. Note from the dependency graph that

   a. $A \Rightarrow^* B$ i. e., B can be obtained from A
   So, all non - unit productions generated from B can also be generated from A

   b. $A \Rightarrow^* C$ i. e., C can be obtained from A
   So, all non - unit productions generated from C can also be generated from A

c. $B \Rightarrow^* C$ i. e., C can be obtained from B

So, all non - unit productions generated from C can also be generated from B

d. $C \Rightarrow^* B$ i. e., B can be obtained from C

So , all non - unit productions generated from B can also be generated from C.

6. Finally, the unit productions can be deleted from the grammar G.

7. The resulting grammar G_1 , generates the same language as accepted by G.

Example1 : Eliminate all unit productions from the grammar

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow C \mid b \\ C &\rightarrow D \\ D &\rightarrow E \mid bC \\ E &\rightarrow d \mid Ab \end{aligned}$$

Solution : The non unit productions of the grammar G are shown below :

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow b \\ D &\rightarrow bC \\ E &\rightarrow d \mid Ab \end{aligned} \qquad \text{............... (1)}$$

The unit productions of the grammar G are shown below :

$$\begin{aligned} B &\rightarrow C \\ C &\rightarrow D \\ D &\rightarrow E \end{aligned}$$

The dependency graph for the unit productions is shown below :

$$B \longrightarrow C \longrightarrow D \longrightarrow E$$

It is clear from the dependency graph that all non unit productions from E can be generated from D. The non unit productions from E are

$$E \rightarrow d \mid Ab \qquad \text{.............. (2)}$$

Since $D \Rightarrow^* E$,

$$D \rightarrow d \mid Ab$$

The resulting D productions are

$D \to bC$ (from production(1))

$D \to d | Ab$ ....(3)

From the dependency graph it is clear that, $C \Rightarrow^* E$. So, the non unit productions from E shown in (production(2)) can be generated from C. Therefore,

$C \to d \mid Ab$

From the dependency graph it is clear that, $C \Rightarrow^* D$. So, the non unit productions from D shown in (production(3)) can be generated from C. Therefore,

$C \to bC$

$C \to d \mid Ab$ ....(4)

From the dependency graph it is clear that $B \Rightarrow^* C$, $B \Rightarrow^* D$, $D \Rightarrow^* E$. So, all the productions obtained from B can be obtained using (productions (1), (2), (3) and (4)) and the resulting productions are :

$B \to b$

$B \to d \mid Ab$

$B \to bC$ .... (5)

The final grammar obtained after eliminating unit productions can be obtained by combining the productions (Productions (1), (2), (3), (4), and (5)) and is shown below :

V_1 = { S, A, B, C, D, E }

T_1 = { a, b, d }

P_1 = {

S	→	AB
A	→	a
B	→	b | d | Ab | bC
C	→	bC | d | Ab
D	→	bC | d | Ab
E	→	d | Ab

} S is the start symbol

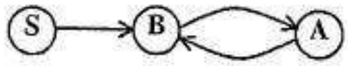
Example 2 : Eliminate unit productions from the grammar

S	→	A 0 | B
B	→	A | 11
A	→	0 | 12 | B

Solution : The unit productions of the grammar G are shown below :

$$\begin{aligned} S &\rightarrow B \\ B &\rightarrow A \\ A &\rightarrow B \end{aligned}$$

The dependency graph for the unit productions is shown below.



The non unit productions are :

$$\begin{aligned} S &\rightarrow A0 \\ B &\rightarrow 11 \\ A &\rightarrow 0 \mid 12 \qquad \text{.............(1)} \end{aligned}$$

It is clear from the dependency graph that $S \Rightarrow^* B$, $S \Rightarrow^* A$, $B \Rightarrow^* A$ and $A \Rightarrow^* B$. So, the new productions from S, A and B are

$$\begin{aligned} S &\rightarrow 11 \mid 0 \mid 12 \\ B &\rightarrow 0 \mid 12 \\ A &\rightarrow 11 \qquad \text{................(2)} \end{aligned}$$

The resulting grammar without unit productions can be obtained by combining Productions (1) and (2) and is shown below :

$$\begin{aligned} V_1 &= \{ S, A, B \} \;, \; T_1 = \{ 0, 1, 2 \} \\ P_1 &= \{ \quad S \rightarrow A0 \mid 11 \mid 0 \mid 12 \\ &\qquad\; A \rightarrow 0 \mid 12 \mid 11 \\ &\qquad\; B \rightarrow 11 \mid 0 \mid 12 \\ &\quad \} \quad S \text{ is the start symbol} \end{aligned}$$

Note : Given any grammar, all undesirable productions can be eliminated by removing

1. $\in$ - productions using theorem 6.5.2
2. unit productions using theorem 6.5.3.
3. useless symbols and productions using theorem 6.5.1

in sequence. The final grammar obtained does not have any undesirable productions.

5.6 NORMAL FORMS

As we have seen the grammar can be simplified by reducing the $\in$ production, removing useless symbols, unit productions. There is also a need to have grammar in some specific form. As you have seen in CFG at the right hand of the production there are any number of terminal or non - terminal symbols in any combination. We need to normalize such a grammar. That means we want the grammar in some specific format. That means there should be fixed number of terminals and non - terminals, in the context free grammar.

In a CFG, there is no restriction on the right hand side of a production. The restrictions are imposed on the right hand side of productions in a CFG resulting in normal forms. The different normal forms are :

1. Chomsky Normal Form (CNF)
2. Greiback Normal Form (GNF)

5.6.1 Chomsky Normal Form (CNF)

Chomsky normal form can be defined as follows.

Non - terminal → Non - terminal.Non - terminal Non - terminal → terminal

The given CFG should be converted in the above format then we can say that the grammar is in CNF. Before converting the grammar into CNF it should be in reduced form. That means remove all the useless symbols, $\in$ productions and unit productions from it. Thus this reduced grammar can be then converted to CNF.

Definition :
Let G = (V, T, P, S) be a CFG. The grammar G is said to be in CNF if all productions are of the form

$$A \rightarrow BC$$
or
$$A \rightarrow a$$

where A, B and $C \in V$ and $a \in T$.

Note that if a grammar is in CNF, the right hand side of the production should contain two symbols or one symbol. If there are two symbols on the right hand side those two symbols must be non - terminals and if there is only one symbol, that symbol must be a terminal.

Theorem 5.6.1 : Let G = (V, T, P, S) be a CFG which generates context free language without $\in$. We can find an equivalent context free grammar $G_1 = (V_1, T, P_1, S)$ in CNF such that $L(G) = L(G_1)$ i. e., all productions in G_1 are of the form

$$A \rightarrow BC$$
or
$$A \rightarrow a$$

Proof : Let the grammar G has no $\in$ - productions and unit productions. The grammar G_1 can be obtained using the following steps.

Step 1 : Consider the productions of the form

$$A \rightarrow X_1 X_2 X_3 \ldots\ldots\ldots\ldots X_n$$

where $n \geq 2$ and each $X_i \in (V \cup T)$ i. e., consider the productions having more than two symbols on the right hand side of the production. If X is a terminal say a, then replace this terminal by a corresponding non terminal B_a and introduce the production

$$B_a \rightarrow a$$

The non - terminals on the right hand side of the production are retained. The resulting productions are added to P_1. The resulting context free grammar $G_1 = (V_1, T, P_1, S)$ where each production in P_1 is of the form

$$A \rightarrow A_1 \ A_2 \ldots\ldots\ldots\ldots A_n$$

or

$$A \rightarrow a$$

generates the same language as accepted by grammar G. So, $L(G) = L(G_1)$.

Step 2 : Restrict the number of variables on the right hand side of the production. Add all the productions of G_1 which are in CNF to P_1. Consider a production of the form

$$A \rightarrow A_1 \ A_2 \ \ldots\ldots\ldots\ldots A_n$$

where $n \geq 3$ (Note that if n = 2 , the production is already in CNF and n can not be equal to 1. Because if n = 1, there is only one symbol and it is a terminal which again is in CNF). The A - production can be written as

$$\begin{array}{lcl} A & \rightarrow & A_1 \ D_1 \\ D_1 & \rightarrow & A_2 \ D_2 \\ D_2 & \rightarrow & A_3 \ D_3 \\ & \vdots & \\ D_{n-2} & \rightarrow & A_{n-1} D_{n-1} \end{array}$$

These productions are added to P_1 and new variables are added to V_1. The grammar thus obtained is in CNF. The resulting grammar $G_1 = (V_1, T, P_1, S)$ generates the same language as accepted by G i. e. $L(G) = L(G_1)$.

Example 1 : Consider the grammar

$S \rightarrow 0A \mid 1B$
$A \rightarrow 0AA \mid 1S \mid 1$
$B \rightarrow 1BB \mid 0S \mid 0$ Obtain the grammar in CNF :

Solution :

Step 1 : All productions which are in CNF are added to P_1. The productions which are in standard form and added to P_1 are :

$$A \rightarrow 1$$
$$B \rightarrow 0 \qquad \text{...... (1)}$$

Consider the productions, which are not in CNF. Replace the terminal a on right hand side of the production by a non - terminal A and introduce the production $A \rightarrow a$. This step has to be carried out for each production which are not in CNF.

The table below shows the action taken indicating which terminal is replaced by the corresponding non - terminal and what is the new production introduced. The last column shows the resulting productions.

Given Productions	Action	Resulting productions
$S \rightarrow 0A \mid 1B$	Replace 0 by B_0 and introduce the production $B_0 \rightarrow 0$ Replace 1 by B_1 and introduce the production $B_1 \rightarrow 1$	$S \rightarrow B_0 A \mid B_1 B$ $B_0 \rightarrow 0$ $B_1 \rightarrow 1$
$A \rightarrow 0AA/1S$	Replace 0 by B_0 and introduce the production $B_0 \rightarrow 0$ Replace 1 by B_1 and introduce the production $B_1 \rightarrow 1$	$A \rightarrow B_0\ AA / B_1 S$ $B_0 \rightarrow 0$ $B_1 \rightarrow 1$
$B \rightarrow 1BB/0S$	Replace 0 by B_0 and introduce the production $B_0 \rightarrow 0$ Replace 1 by B_1 and introduce the production $B_1 \rightarrow 1$	$B \rightarrow B_1 BB / B_0 S$ $B_1 \rightarrow 1$ $B_0 \rightarrow 0$

The grammar $G_1 = (V_1, T, P_1, S)$ can be obtained by combining the productions obtained from the last column in the table and the productions shown in (1).

$$V_1 = \{S, A, B, B_0, B_1\}$$
$$T_1 = \{0, 1\}$$
$$P_1 = \{ \quad S \rightarrow B_0 A | B_1 B$$
$$A \rightarrow B_0 A A | B_1 S | 1$$
$$B \rightarrow B_1 B B | B_0 S | 0$$
$$B_0 \rightarrow 0$$
$$B_1 \rightarrow 1$$
$\}$ S is the start symbol

Step 2 :

Restricting the number of variables on the right hand side of the production to 2. The productions obtained after step 1 are :

$$S \rightarrow B_0 A | B_1 B$$
$$A \rightarrow B_0 A A | B_1 S | 1$$
$$B \rightarrow B_1 B B | B_0 S | 0$$
$$B_0 \rightarrow 0$$
$$B_1 \rightarrow 1$$

In the above productions, the productions which are in CNF are

$$S \rightarrow B_0 A | B_1 B$$
$$A \rightarrow B_1 S | 1$$
$$B \rightarrow B_0 S | 0$$
$$B_0 \rightarrow 0$$
$$B_1 \rightarrow 1 \qquad \text{............. (2)}$$

and add these productions to P_1. The productions which are not in CNF are

$$A \rightarrow B_0 A A$$
$$B \rightarrow B_1 B B$$

The following table shows how these productions are changed to CNF so that only two variables are present on the right hand side of the production.

Given Productions	Action	Resulting productions
$A \to B_0AA$	Replace AA on R.H.S with variable D_1 and introduce the production $D_1 \to AA$	$A \to B_0D_1$ $D_1 \to AA$
$B \to B_1BB$	Replace BB on R. H. S with variable D_2 and introduce the production $D_2 \to BB$	$B \to B_1D_2$ $D_2 \to BB$ ....(3)

The final grammar which is in CNF can be obtained by combining the productions in (2) and (3).

The grammar $G_1 = (V_1, T, P_1, S)$ is in CNF where

$$V_1 = \{ S, A, B, B_0, B_1, D_1, D_2 \}$$
$$T_1 = \{ 0, 1 \}$$
$$P_1 = \{ \begin{array}{lll} S & \to & B_0A \mid B_1B \\ A & \to & B_1S \mid 1 \mid B_0D_1 \\ B & \to & B_0S \mid 0 \mid B_1D_2 \\ B_0 & \to & 0 \\ B_1 & \to & 1 \\ D_1 & \to & AA \\ D_2 & \to & BB \end{array}$$
$\}$ S is the start symbol

Example 2 : Find a grammar in CNF equivalent to the grammar :

$$S \to -S \mid [\, S \uparrow S \,] \mid a \mid b$$

Solution : Given, grammar is :

$$S \to -S \mid [\, S \uparrow S \,] \mid a \mid b \qquad .....(A)$$

where, terminals are :

$$-, [\,, \uparrow,], \; a \; and \; b$$

In the given grammar (A) there is no any $\in$–production, no any unit - production and no any useless symbols .

Now, in the given grammar (A), following are the productions which is already in the form of CNF:

$$S \to a$$
$$S \to b$$

Also, in the given grammar (A), following are the productions which are not in the form of CNF:

$$S \rightarrow -S$$

$$S \rightarrow [S \uparrow S]$$

Thus: (a) Considering the production:

$$S \rightarrow -S$$

We can write this production as:

$$S \rightarrow V_1 S \quad \text{.... (1)}$$

$$V_1 \rightarrow - \quad \text{.... (2)}$$

where V_1 is a new variable.

(b) Now, considering the production:

$$S \rightarrow [S \uparrow S]$$

We can write this production as:

$$S \rightarrow V_2\ SV_3\ SV_4 \quad \text{.... (3)}$$

$$V_2 \rightarrow [\quad \text{.... (4)}$$

$$V_3 \rightarrow \uparrow \quad \text{.... (5)}$$

$$V_4 \rightarrow] \quad \text{.... (6)}$$

where V_2, V_3 and V_4 are new variables.

Thus, from (1), ..., (6), the result grammar becomes:

$$S \rightarrow V_1S \mid V_2S\ V_3\ SV_4 \mid a \mid b$$

$$V_1 \rightarrow -$$

$$V_2 \rightarrow [\quad \text{.....(B)}$$

$$V_3 \rightarrow \uparrow$$

$$V_4 \rightarrow]$$

Now, in the resultant grammar (B), following is the production which is not in the form of CNF:

$$S \rightarrow V_2\, SV_3\, SV_4$$

(c) Now, considering the production:

$$S \rightarrow V_2 SV_3 SV_4$$

We can write this production as:

$$S \rightarrow V_2 V_5 V_6 \quad \text{.....(7)}$$

$$V_5 \rightarrow SV_3 \quad \text{.....(8)}$$

$$V_6 \rightarrow SV_4 \quad \text{.....(9)}$$

Thus, from (7), (8) and (9), the resultant grammar becomes :

$$S \rightarrow V_1\ S \mid V_2V_5V_6 \mid a \mid b$$
$$V_1 \rightarrow -$$
$$V_2 \rightarrow [$$
$$V_5 \rightarrow SV_3 \qquad .....(C)$$
$$V_6 \rightarrow SV_4$$
$$V_3 \rightarrow \uparrow$$
$$V_4 \rightarrow]$$

Now, in the resultant grammar (C), following is the production which is not in the form of CNF:

$$S \rightarrow V_2V_5\ V_6$$

We can write this production as :

$$S \rightarrow V_2V_7 \qquad .....(10)$$
$$V_7 \rightarrow V_5V_6 \qquad .....(11)$$

Thus, from (10) and (11), the resultant grammar becomes :

$$S \rightarrow V_1S \mid V_2V_7 \mid a \mid b$$
$$V_1 \rightarrow -$$
$$V_2 \rightarrow [$$
$$V_7 \rightarrow V_5\ V_6 \qquad .....(D)$$
$$V_5 \rightarrow SV_3$$
$$V_6 \rightarrow SV_4$$
$$V_3 \rightarrow \uparrow$$
$$V_4 \rightarrow]$$

Thus, the resultant grammar (D) is in the form of CNF, which is the required solution.

5.6.2 Greibach Normal form (GNF)

Greibach normal form can be defined as follows :

Non - terminal $\rightarrow$ one terminal. Any number of non - terminals

Example :

$S \rightarrow aA$ is in GNF

$S \rightarrow a$ is in GNF

But $S \to AA$ is not in GNF
$S \to Aa$ is not in GNF

Definition : A CFG $G = (V, T, P, S)$ is in Greibach normal form (GNF) if its all productions are of type $A \to a\alpha$, where $\alpha \in V^*$ (String of variables including null string) and $a \in T$. A grammar in GNF is the natural generalization of a regular grammar (right - linear).

Theorem 5.6.2 : Every CFL L without $\in$ is generated by grammar, where productions are of type $A \to a\alpha$, where $\alpha \in V^*$ and $a \in T$.

Proof : We use removal of left recursion (without null productions) as given below.
Let the variable A has left recursive productions given as follows :

$A \to A\alpha_1 | A\alpha_2 | A\alpha_3 | .... | A\alpha_n | \beta_1 | \beta_2 | \beta_3 | .... | B_m$, where $\beta_1, \beta_2, \beta_3, ... B_m$ do not begin with A, then we replace A - productions by the productions given below.

$$A \to \beta_1 A' | \beta_2 A' | .... | \beta_m A' | \beta_1 | \beta_2 | \beta_3 | ... | B_m, \text{ where}$$
$$A' \to \alpha_1 A' | \alpha_2 A' | \alpha_3 A' | .... | \alpha_n A' | \alpha_1 | \alpha_2 | \alpha_3 | ... | \alpha_n$$

Method for Converting a CFG into GNF :

We consider CFG $G = (V, T, P, S)$.

Step 1 : Rename all the variables of G as $A_1, A_2, A_3, ..., A_n$

Step 2 : Repeat Step 3 and Step 4 for $i = 1, 2, ..., n$

Step 3 : If $A_i \to a\alpha_1\alpha_2\alpha_3 ...... \alpha_m$, where $a \in T$, and α_j is a variable or a terminal symbol,

Repeat for $j = 1, 2, ..., m$

If α_j is a terminal then replace it by a variable A_{n+j} and add production $A_{n+j} \to \alpha_j$, and $n = n+1$. Consider the next A_i – production and go to step 3.

Step 4 : If $A_i \to \alpha_1\alpha_2\alpha_3 .... \alpha_m$, where α_1 is a variable, then perform the following :

If α_1 is same as A_i, then remove the left recursion and go to Step 3.

Else replace α_1 by all RHS of α_1-productions one by one. Consider the remaining A_i-productions, which are not in GNF and go to Step 3.

Step 5 : Exit

Advantages of GNF :

1. Avoids left recursion.
2. Always has terminal in leftmost position in RHS of each production.
3. Helps select production correctly.
4. Guarantees derivation length no longer than string length.

Example 1 : Consider the CFG $S \to S+S \mid S*S \mid a \mid b$ and find an equivalent grammar in GNF.

Solution: Let G_1 is the equivalent grammar in GNF.

Renaming the variable, we get

$P_1 : S_1 \to S_1 + S_1$ (Not in GNF)

$P_2 : S_1 \to S_1 * S_1$ (Not in GNF)

$P_3 : S_1 \to a$ (In GNF)

$P_4 : S_1 \to b$ (In GNF)

P_1 and P_2 are left recursive productions, so removing the left recursion, we get

$S_1 \to a S_2 \mid bS_2 \mid a \mid b$, where

$S_2 \to + S_1 S_2 \mid * S_1 S_2 \mid + S_1 \mid * S_1$

Now, all productions are in GNF.

Example 2 : Consider the grammar $G = (\{A_1, A_2, A_3\}, \{a, b\}, P, A_1)$, where P consists of following production rules.

$A_1 \to A_2 A_3, A_2 \to A_3 A_1 \mid b, A_3 \to A_1 A_2 \mid a$ Convert it into GNF.

Solution : (Renaming is not required)

Consider A_1 - productions :

$A_1 \to A_2 A_3$ (Not in GNF)

Replacing A_2 by its RHS, we get

$A_1 \to bA_3$ (In GNF)

$A_1 \to A_3 A_1 A_3$ (Not in GNF)

Now , consider $A_1 \to A_3 A_1 A_3$, and replacing A_3 by its RHS, we get

$A_1 \to aA_1 A_3$ (In GNF)

$A_1 \to A_1 A_2 A_1 A_3$ (Not in GNF)

So, A_1 - productions are $A_1 \to bA_3 \mid aA_1A_3 \mid A_1A_2A_1A_3$

Now, consider $A_1 \to A_1 A_2 A_1 A_3$ and removing left recursion, we get

$A_1 \to bA_3A_4 \mid bA_3$ (In GNF)

$A_1 \to aA_1A_3A_4 \mid aA_1A_3$ (In GNF), where

$A_4 \to A_2A_1A_3A_4 \mid A_2A_1A_3$

(A_4 is a new variable and its production is not in GNF)

So, now all A_1 - productions are in GNF .

Consider the A_2 - productions :

$A_2 \rightarrow b$ (In GNF)

$A_2 \rightarrow A_3 A_1$ (Not in GNF)

Now, consider $A_2 \rightarrow A_3 A_1$ and replacing A_3 by its RHS, we get

$A_2 \rightarrow aA_1$ (In GNF)

$A_2 \rightarrow A_1 A_2 A_1$ (Not in GNF)

Now, consider $A_2 \rightarrow A_1 A_2 A_1$ and replacing A_1 by its RHS, we get

$A_2 \rightarrow bA_3 A_4 A_2 A_1$, (In GNF)

$A_2 \rightarrow bA_3 A_2 A_1$ (In GNF)

$A_2 \rightarrow aA_1 A_3 A_4 A_2 A_1$ (In GNF)

$A_2 \rightarrow aA_1 A_3 A_2 A_1$ (In GNF)

So, all A_2 - productions are in GNF.

Consider A_3 - productions :

$A_3 \rightarrow a$ (In GNF)

$A_3 \rightarrow A_1 A_2$ (Not in GNF)

Now, consider $A_3 \rightarrow A_1 A_2$ and replacing A_1 by its RHS, we get

$A_3 \rightarrow bA_3 A_4 A_2 \mid bA_3 A_2 \mid aA_1 A_3 A_4 A_2 \mid aA_1 A_3 A_2$ (In GNF)

Consider A_4 - productions :

$A_4 \rightarrow A_2 A_1 A_3 A_4 \mid A_2 A_1 A_3$ (Not in GNF)

Replacing A_2 by its RHS, we get

$A_4 \rightarrow bA_1 A_3 A_4 \mid bA_1 A_3 \mid aA_1 A_1 A_3 A_4 \mid aA_1 A_1 A_3$,

$A_4 \rightarrow bA_3 A_4 A_2 A_1 A_1 A_3 A_4$,

$A_4 \rightarrow bA_3 A_4 A_2 A_1 A_1 A_3$,

$A_4 \rightarrow bA_3 A_2 A_1 A_1 A_3 A_4$,

$A_4 \rightarrow bA_3 A_2 A_1 A_1 A_3$,

$A_4 \rightarrow aA_1 A_3 A_4 A_2 A_1 A_1 A_3 A_4$,

$A_4 \rightarrow aA_1 A_3 A_4 A_2 A_1 A_1 A_3$,

$A_4 \rightarrow aA_1 A_3 A_2 A_1 A_1 A_3 A_4$, and

$A_4 \rightarrow aA_1 A_3 A_2 A_1 A_1 A_3$

Now, all A_4 - productions are in GNF.

Productions in GNF are :

$$A_1 \to aA_1A_3 \mid bA_3A_4 \mid bA_3 \mid aA_1A_3A_4 \mid aA_1A_3$$
$$A_2 \to b \mid aA_1 \mid bA_3A_4A_2A_1 \mid bA_3A_2A_1 \mid aA_1A_3A_4A_2A_1 \mid aA_1A_3A_2A_1,$$
$$A_3 \to a \mid bA_3A_4A_2 \mid bA_3A_2 \mid aA_1A_3A_4A_2 \mid aA_1A_3A_2,$$
$$A_4 \to bA_1A_3A_4 \mid bA_1A_3 \mid aA_1A_1A_3A_4 \mid aA_1A_1A_3 \mid bA_3A_4A_2A_1A_1A_3A_4,$$
$$A_4 \to bA_3A_2A_1A_1A_3A_4 \mid aA_1A_3A_4A_2A_1A_1A_3A_4 \mid aA_1A_3A_2A_1A_1A_3A_4,$$
$$A_4 \to bA_3A_4A_2A_1A_1A_3 \mid bA_3A_2A_1A_1A_3 \mid aA_1A_3A_4A_2A_1A_1A_3 \mid aA_1A_3A_2A_1A_1A_3$$

Example 3 : Find equivalent grammar in GNF.

(a) S → aB | b A, A → aS | b AA | a, B → bS | a BB | b
(b) S → abSb | a | a Ab, A → bS | a AAb
(c) S → AA | 0 , A → SS | 1

Solution :

(a) Renaming S, A and B by A_1, A_2, and A_3 respectively, we get the following productions.

$A_1 \to aA_3 \mid bA_2, A_2 \to aA_1 \mid bA_2A_2 \mid a, A_3 \to bA_1 \mid aA_3A_3 \mid b$

Since, all productions are in GNF, so there is no need of any modification.

(b) Renaming S, and A by A_1 and A_2 respectively, we get the following productions.

$A_1 \to abA_1b \mid a \mid aA_2b, A_2 \to bA_1 \mid aA_2A_2b$

Consider the A_1 - productions one by one.

$A_1 \to abA_1b$ (Not in GNF)

Replacing all the RHS terminals except the first by new variables, we get

$A_1 \to aA_3A_1A_3$ where $A_3 \to b$ (In GNF)

Considering the next A_1 - production :

$A_1 \to a$ (In GNF)

Considering the next A_1 - production :

$A_1 \to aA_2b$ (Not in GNF)

Replacing b by variable A_3 (since, we have already defined $A_3 \to b$), we get

$A_1 \to aA_2A_3$ (In GNF)

Consider the A_2 - production :

$A_2 \to bA_1$ (In GNF)

Considering the next A_2 - production :

$A_2 \rightarrow aA_2A_2b$ (Not in GNF)

Replacing b by variable A_3 (since, we have already defined $A_3 \rightarrow b$), we get

$A_2 \rightarrow aA_2A_2A_3$ (In GNF)

Now, all productions given following are in GNF.

$A_1 \rightarrow aA_3A_1A_3 | a | aA_2A_3$, $A_2 \rightarrow bA_1 | aA_2A_2A_3$, and $A_3 \rightarrow b$

(c) Renaming S, and A by A_1, and A_2 respectively, we get the following productions

$A_1 \rightarrow A_2A_2 | 0$, $A_2 \rightarrow A_1A_1 | 1$

Consider the A_1 - productions one by one.

$A_1 \rightarrow A_2A_2$ (Not in GNF)

Replacing leftmost A_2 by A_1A_1 and 1, we get

$A_1 \rightarrow A_1A_1A_2 | 1A_2$

Considering the production $A_1 \rightarrow A_1A_1A_2$, this is not in GNF and has left recursion. Considering the all A_1 - productions $A_1 \rightarrow A_1A_1A_2 | 1A_2 | 0$ and removing left recursion from the production $A_1 \rightarrow A_1A_1A_2$, we get $A_1 \rightarrow 1A_2A_3 | 0A_3$ (In GNF),

Where $A_3 \rightarrow A_1A_2A_3 | A_1A_2$

Considering A_2 - production $A_2 \rightarrow A_1A_1$ and replacing left most A_1 by $1A_2A_3$ and $0A_3$, we get

$A_2 \rightarrow 1A_2A_3A_1 | 0A_3A_1$ (In GNF)

Considering A_3 - productions $A_3 \rightarrow A_1A_2A_3 | A_1A_2$ and replacing A_1 by $1A_2A_3$ and $0A_3$, we get

$A_3 \rightarrow 1A_2A_3A_2 | 0A_3A_2 | 1A_2A_3A_2A_3 | 0A_3A_2A_3$ (In GNF)

Now, the productions in GNF are following.

$A_1 \rightarrow 1A_2A_3 | 0A_3$, $A_2 \rightarrow 1A_2A_3A_1 | 0A_3A_1 | 1$,

and $A_3 \rightarrow 1A_2A_3A_2 | 0A_3A_2 | 1A_2A_3A_2A_3 | 0A_3A_2A_3$

5.7 PUMPING LEMMA FOR CFLs

The pumping lemma for CFLs states that there are always two short substrings close together that can be pumped same number of times as we like and the result is a string in the same CFL.

Lemma :

Let L be a CFL and a long string z is in L, then there exists a constant n such that $|z| \geq n$ and z can be written as uvwxy such that

(i) $|vx| \geq 1$

(ii) $|vwx| \leq n$, and

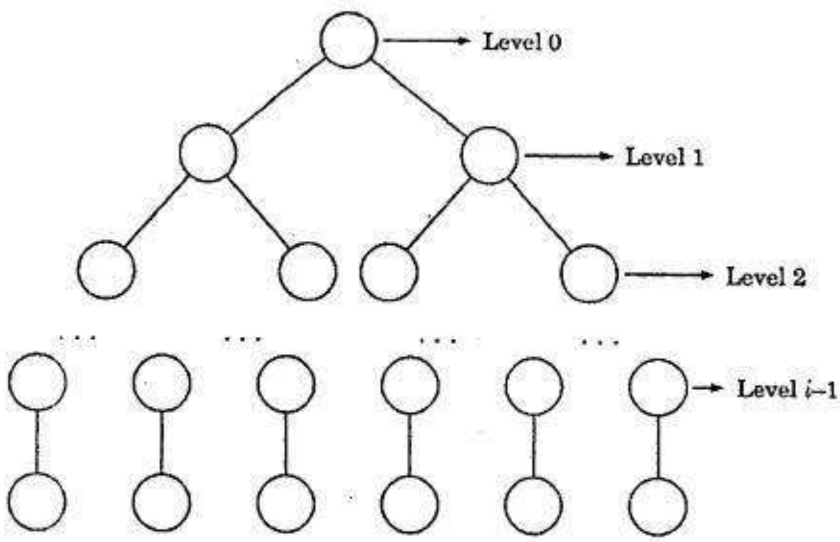
(iii) $uv^i wx^i y$ is in L for i = 0, 1, 2,.......

Proof :

Let G be a CFG in CNF and generates $L-\{\in\}$. Since, z is a long string, so parse tree for z must contain a long path. Suppose, the longest path in parse tree of z has length h. In the parse tree, no word can be greater that the length 2^{h-1} or in other words, the maximum length word would of length 2^{h-1}.

We see the proof as follows :

Since, the grammar G is in CNF (productions are of types $A \to a$ or $A \to XY$), so parse tree for z is a binary tree. The parse tree yields longest word if and only if its all levels except the last level contain two children as shown in below figure .



DEPARTMENT OF CSE

Since, the number of leaves is the length of longest string and it is equal to the number of nodes at level $i-1$ as shown in above figure .The number of nodes at level $i-1 = 2^{i-1}$

So, the longest word has the length 2^{h-1}, where h is the longest path length. In other words, we say that no word can be greater than 2^{h-1} length.

Let G has k variables and $n = 2^k$. If z is in L(G) and $|z| \geq 2^k$. So, the longest path in the parse tree of z has length $k+1$ and this path contains $k+2$ vertices ($k+1$ internal vertices and one terminal vertex). Since, all the vertices except the terminal are variables, so the longest path contains $k+1$ variables. It means, one variable appears twice in the longest path. Let variable A appears twice, So $A \overset{*}{\underset{G}{\Rightarrow}} z_3 \; A z_4 \overset{*}{\underset{G}{\Rightarrow}} (z_3)^i \; A(z_4)^i$, where z_3 and z_4 are two substrings of z. Let $A \overset{*}{\underset{G}{\Rightarrow}} z_2$ then $A \overset{*}{\underset{G}{\Rightarrow}} (z_3)^i z_2 (z_4)^i$. We say that z_3 and z_4 can be pumped same number of times as we like.

Example : Consider a CFG $S \rightarrow SS \mid a$ and $z = aaaa$. The parse tree for z is shown in figure(a) .

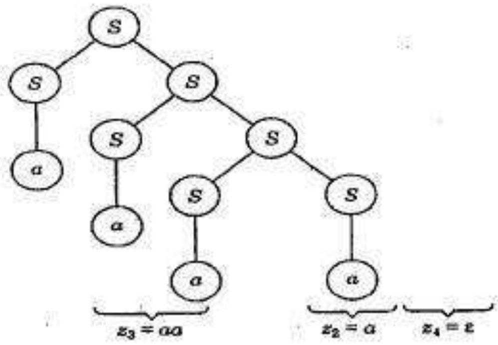


Figure (a)

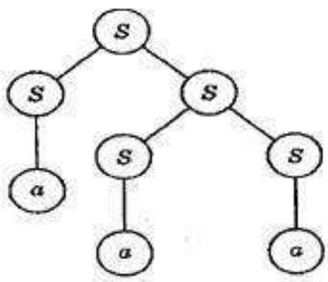


Figure (b)

Figure (c)

From the subtree shown in figure (b), we get $S \overset{*}{\Rightarrow} aaS \in$ or $S \overset{*}{\Rightarrow} z_3\ S\ z_4$ and considering the subtree shown in figure(c), we get $S \overset{*}{\Rightarrow} a$ or $S \overset{*}{\Rightarrow} z_2$.

The subtree shown in figure (b) can be added as many times as we like in the parse tree shown in figure (a). So, $S \overset{*}{\Rightarrow} z_3^i\ S\ z_4^i \overset{*}{\Rightarrow} z_3{}^i z_2 z_4^i$

Therefore, string z can be written as $uz_3z_2z_4y$ for some u and y substrings of z. The substrings z_3 and z_4 can be pumped as many times as we like. Replacing z_3, z_2 and z_4 by v, w and x respectively, we get z = uvwxy and $S \overset{*}{\Rightarrow} uv^iwx^iy$ for some i = 0, 1, 2, ...
Hence, the statement of theorem is proved.

Application of Pumping Lemma for CFLs

We use the pumping lemma to prove certain languages are not CFL. We proceed as we have seen in application of pumping lemma for regular sets and get contradiction. The result of this lemma is always negative.

Procedure for Proving Language is not Context - free

The following steps are considered to show a given language is not context - free.

Step 1 :

Suppose that L is context - free. Let 1 be the natural number obtained by using pumping lemma.

Step 2 :

Choose a string $x \in L$ such that $|x| \geq 1$ using pumping lemma principle write z = uvwxy.

Step 3 :

Find suitable i so that $uv^iwx^iy \notin L$. This is a contradiction. So L is not context - free.

Example 1 : Consider the language $L = \{a^n\ b^n\ c^n : n \geq 1\}$ and prove that L is not CFL.

Solution : All the words of L contain equal number of a's, b's and c's. Let L is a CFL and z is a long string in L such that $|z| \geq n$. Using Pumping Lemma for L, we write $z = uvwxy$ and $uv^i wx^i y$ is in L for some i = 0, 1, 2, ... and $|vx| \geq 1$ and $|vwx| \leq n$.

The substring vx may be $a^p, b^q, c^r, a^p b^q, b^q c^r$ but not $a^p c^r$.

Consider i = 0, so uwy is in L.

Case 1 : $vx = a^p$, so $z = uwy = a^{n-p} b^n c^n$ is in L.

The number of a's is fewer than the number of b's and c's for $p \geq 1$, which is a contradiction.

Case 2 : $vx = b^q$, so z = uwy = $a^n b^{n-q} c^n$ is in L.

The number of b's is fewer than the number of a's and c's for $q \geq 1$, which is a contradiction.

Case 3 : $vx = c^r$, so $z = uwy = a^n b^n c^{n-r}$ is in L.

The number of c's is fewer than the number of a's and b's for $r \geq 1$, which is a contradiction.

Case 4 : $vx = a^p b^q$, so $z = uwy = a^{n-p} b^{n-q} c^n$ is in L.

The number of a's and b's are fewer than the number of c's for $p, q \geq 1$, which is a contradiction.

Case 5 : $vx = b^q c^r$, so $z = uwy = a^n b^{n-q} c^{n-r}$ is in L.

The number of b's and c's are fewer than the number of a's for $q, r \geq 1$, which is a contradiction.
Since, we get contradiction for all values of vx, so L is not a CFL.

Example 2 : Prove that following languages are not CFL

(a) $L = \{a^p : p \text{ is a prime number}\}$

(b) $L = \{a^n b^m c^n d^m : m, n \geq 1\}$

(c) $L = \{a^n b^n c^m : m \geq n\}$

Solution :

(a) All the words of L have length prime. Let L be a CFL and z is a long string in L. Using Pumping Lemma for L, we write $z = uvwxy$ and $uv^i wx^i y$ is in L for some $i = 0, 1, 2, ...$ and $|vx| = m$ and $|uwy| = n$ where n is a prime number then $|uv^n wx^n y| = n + mn$. As $n + mn$ is not a prime number, so $uv^n wx^n y \notin L$ and this is a contradiction. Therefore, L is not a CFL.

(b) Let L be a CFL and z is a long string in L such that $z = uvwxy$ for $|vx| = 1$ and $|vwx| = k$, where k is some constant.

In L, all words have equal number of a's and c's and equal number of b's and d's. The value of vx may be combination of two consecutive symbols like $a^p b^q, b^q c^r, c^r d^s$.

According to pumping lemma $uv^i wx^i y$ is in L for some $i = 0, 1, 2, \ldots$

Consider $i = 0$, then $z = uwy$ is in L.

Case 1 : $vx = a^p b^q$, then

$$z = a^{n-p} b^{m-q} c^n d^m$$

The number of a's and b's are fewer than the number of c's and d's for $p, q \geq 1$, which is a contradiction.

Case 2 : $vx = b^q c^r$, then $z = a^n b^{m-q} c^{n-r} d^m$

The number of b's and c's are fewer than the number of d's and a's for $q, r \geq 1$, which is a contradiction.

Case 3 : $vx = c^r d^s$, then $z = a^n b^m c^{n-r} d^{m-s}$

The number of c's and d's are fewer than number of a's and b's for $r, s \geq 1$, which is a contradiction.

Since, we are getting contradiction in all cases, so L is not a CFL.

(c) All the words of L contain equal number of a's, b's and number of c's is greater than number of a's (or b's). Let L is a CFL and z is a long string in L such that $|z| \geq n$. Using pumping lemma for L, we write $z = uvwxy$ and $uv^i wx^i y$, which are in L for some $i = 0, 1, 2, \ldots$ and $|vx| \geq 1$ and $|vwx| \leq n$.

The substring vx may be $a^p, b^q, c^r, a^p b^q, b^q c^r$ but not $a^p c^r$.

Consider $i = 0$, so uwy is in L.

Case 1 : $vx = a^p$, so $z = uwy = a^{n-p} b^n c^n$ is in L.

The number of a's is fewer than the number of b's for $p \geq 1$, which is a contradiction.

Case 2 : $vx = b^q$, so $z = uwy = a^n b^{n-q} c^n$ is in L.

The number of b's is fewer than the number of a's for $q \geq 1$, which is a contradiction.

Case 3 : $vx = c^r$, so $z = uwy = a^n b^n c^{n-r}$ is in L.

The number of c's may be equal or less than the number of a's (or b's) for $r \geq 1$, which is a contradiction.

Since, we are getting contradiction in all cases, so L is not a CFL.

Example 3 : Show that the following language is not context free $\mathbf{L} = \{\mathbf{a}^{\mathbf{n}^2} / \mathbf{n} \geq 1\}$.

Solution :

Method - I : Assume L is context - free and n is the pumping lemma constant

Let $Z = a^{n^2}$

write $Z = uvwxy$, where $|vwx| \leq n$ and $|vx| \geq 1$

Let $|vx| = m$, $m \leq n$

As $|uv^2wx^2y| > n^2$, $|uv^2wx^2y| = k^2$, where k is $\geq n+1$

But $|uv^2wx^2y| = n^2 + m < n^2 + 2n + 1$

So $|uv^2wx^2y|$ strictly lies between n^2 and $(n+1)^2$ which means $uv^2wx^2y \notin L$, a contradiction. Hence $\{a^{n^2} : n \geq 1\}$ is not context-free

Method - II : We can also show that

$L = \{a, aaaa, aaaaaaaaa, ...\}$

$$Z = \underset{v}{\underline{a}}\ \underset{w}{\underline{a}}\ \underset{x}{\underline{a}}\ \underset{y}{\underline{a}}$$

$uv^2wx^2y = \in a^2aa^2a = aaaaaa$

$uv^2wx^2y \notin L$

$\therefore$ L is not context-free.

Example 4 : Show that the following language is not context-free

$$L = \{0^m 1^m 2^n / m \leq n \leq 2m\}.$$

Solution :

Method - I :

Assume L is context-free and n is the pumping lemma constant.

Let $Z = 0^m 1^m 2^n$

Then $Z = uvwxy$, where $1 \leq |vx| \leq n$

So vx cannot contain all the three symbols 0, 1 and 2. If vx contains only 0's and 1's then we can choose i such that $uv^i wx^i y$ has more than $2n$ occurrences of a 0 (or 1) and exactly $2n$ occurrences of L. This means $uv^i wx^i y \notin L$, a contradiction.

In other cases also we can get a contradiction by proper choice of i. Thus the given language is not context - free.

Method - II :

Consider the accepted set of strings from the given language

$L = \{0122, 0011222, 00112222, ...\}$

$$Z = \underset{u}{\underline{0}}\ \underset{v}{\underline{01}}\ \underset{w}{\underline{1}}\ \underset{x}{\underline{22}}\ \underset{y}{\underline{2}}$$

$$uv^2wx^2y = 0\,(01)^2 1\,(22)^2 2 = 0\ 01011\ 22222 \notin L$$

$\therefore$ L is not context-free.

5.7.2 Ogden's Lemma and Its Applications

There exist some non - context free languages which cannot be proved using the lemma of section 5.7. We need a stronger result. Ogden's lemma is more powerful than the pumping lemma. This lemma allows us to fix 'distinguished positions' in the sentence z and puts some conditions for v, x, y with respect to these positions. Proof of Ogden's lemma is beyond the scope of this book. However, we present the statement of Ogden's lemma and illustrate its application.

Statement of Ogden's Lemma

Let L be a context free language. There exists a constant n such that for any sentence $z, |z| \geq n$, we can fix at least n distinguished positions, and z can be written as uvwxy such that

i. vx contains at least one distinguished position,
ii. vwx contains at most n distinguished positions ; and
iii. any string of the form $uv^i wx^i y, i \geq 0$ is in L.

Note :

1. Pumping lemma of 5.7 is a special case of Ogden's lemma in which every position in z is distinguished.
2. In applying Ogden's lemma, choice of distinguished positions is under our control.

Example :

Prove that $L = \{ a^i b^j c^k \mid i \neq j,\ j \neq k$ and $i \neq k \}$ is not context free.

Solution :

If L is context free we can apply Ogden's lemma. Let n be the constant of the lemma. Consider the sentence $z = a^n b^{n!+n} c^{2n!+n}$. We will choose all positions in the block of a's as distinguished. z can be split as uvwxy such that (i) vx has at least one distinguished position and (ii) vwx has at most n distinguished positions : By (i), vx should contain at least one a. These are different cases.

Case 1 :

$v \in a^+$ and $x \in b^*$. Let $u = a^p$ such that $1 \leq p < n$. Then, p is divisor of n!. Let q be the integer such that $pq = n!$.

Consider $z' = uv^{2q+1}wx^{2q+1}y$.

y consists of $(2n!+n)$ $c's$ (remains unchanged).

$$v^{2q+1} = a^{2pq+p} = a^{2n!+p}$$
$$uv^{2q+1} = a^{n-p}a^{2n!+p} = a^{2n!+n}$$

Hence in z', number of a's = number of c's.

Case 2 :

$v \in a^+$ and $x \in c^*$. Let $v = a^p$ and pq=n!. Pumping v and x, $(q+1)$ times, we get : $z' = uv^{q+1}wx^{q+1}y$.

In z', no. of a's will be $n - p + n! + p = n! + n$.

No.of b's in z' will remain n! + n. Hence, no. of a's = no. of b's in z'.

Similarly, in other cases, we can arrive at strings not as per specification of L.

Hence, L is not context free.

5.8 CLOSURE PROPERTIES OF CFLs

The closure properties that hold for regular languages do not always hold for context free languages. Consider those operations which preserve CFL.

The purpose of these operations are to prove certain languages are CFL and certain languages are not CFL.

Context-free languages are closed under following properties.

1. Union
2. Concatenation and
3. Kleene Closure (Context-free languages **may** or **may not** close under following properties)
4. Intersection
5. Complementation

Theorem 5.8.1 : If L_1 and L_2 are two CFLs, then union of L_1 and L_2 denoted by $L_1 + L_2$ or $L_1 \cup L_2$ is also a CFL.

Proof :

Let CFG $G_1 = (V_1, T_1, P, S)$ generates L_1 and CFG $G_2 = (V_2, T_2, P, S)$ generates L_2 and $G = (V, T, P, S)$ generates $L = L_1 + L_2$.

We construct G as follows :

Step 1 : Rename the variables of CFG G_1

If $V_1 = \{S, A, B, ..., X\}$, then the renamed variables are $\{S_1, A_1, B_1, ... X_1\}$. This modification should be reflected in productions also.

Step 2 : Rename the variables of CFG G_2

If $V_2 = \{S, A, B,... X\}$, then the renamed variables are $\{S_2, A_2, B_2....X_2\}$. This modification should be reflected in production also.

Step 3 : We get of the productions of G_1 and G_2 to get productions of G as follows :

$S \rightarrow S_1 \mid S_2$, where S_1 and S_2 are starting symbols of grammars G_1 and G_2 respectively and S_1 - productions and S_2 - productions remain unchanged.

$T = T_1 \cup T_2$,

$V = \{S_1, A_1, B_1,... X_1\} \cup \{S_2, A_2, B_2,... X_2\}$

Since, all productions of G_1 and G_2 including $S \rightarrow S_1 \mid S_2$ are in context-free form, so G is a CFG.

Language generated by *G* :

$L(G)$ = Language generated from (S_1 *or* S_2)

= Language generated from S_1 or language generated from S_2

= $L(G_1)$ or $L(G_2)$ (Since, S_1 and S_2 are starting symbols of G_1 and G_2 respectively.)

= L_1 *or* L_2 (Since, G_1 produces L_1 and G_2 produces L_2.)

= $L_1 + L_2$

Hence, statement of the theorem is proved.

Example : Consider the CFGs $S \rightarrow aSb \mid ab$ and $S \rightarrow cSdd \mid cdd$, which generate languages L_1 and L_2 respectively. Construct grammar for $L = L_1 + L_2$.

Solution :

Let G_1 generates L_1 and G_2 generates L_2 and $G = (V, T, P, S)$ generates $L = L_1 + L_2$.

Renaming the variables of G_1 and G_2, we get

$V_1 = \{S_1\}$ and $V_2 = \{S_2\}$, where S_1 - productions are $S_1 \rightarrow aS_1b \mid ab$, and S_2 - productions are $S_2 \rightarrow cS_2dd \mid cdd$

We define G as follows :

$V = \{S, S_1, S_2\}$,

$T = \{Terminals\ of\ G_1\ or\ G_2\} = \{a, b, c, d\}$,

P includes : $S \rightarrow S_1 \mid S_2$, $S_1 \rightarrow aS_1b \mid ab$, and $S_2 \rightarrow cS_2dd \mid cdd$.

$L = L_1 + L_2$

$= \{a^m b^n : m, n \geq 1\} \cup \{c^n d^{2n} : n \geq 1\}$

Theorem 5.8.2 : If L_1 and L_2 are two CFLs, then concatenation of L_1 and L_2 denoted by L_1L_2 is also a CFL .

Proof : Let CFG $G_1 = (V_1, T_1, P, S)$ generates L_1 and CFG $G_2 = (V_2, T_2, P, S)$ generates L_2 and $G = (V, T, P, S)$ generates $L = L_1L_2$.

We construct G as follows :

Step 1 : Rename the variables of CFG G_1 .

If $V_1 = \{S, A, B, ..., X\}$, then the renamed variables are $\{S_1, A_1, B_1, ...X_1\}$. This modification is reflected in productions also.

Step 2 : Rename the variables of CFG G_2 :

If $V_2 = \{S, A, B, ... X\}$, then the renamed variables are $\{S_2, A_2, B_2, ...X_2\}$. This modification is reflected in productions also.

Step 3 : The productions of G_1 are followed by the productions of G_2 to get productions of G as follows.

$S \rightarrow S_1S_2$, where S_1 and S_2 are starting symbols of grammars G_1 and G_2 respectively and S_1 - productions and S_2 - productions remain unchanged.

$T = T_1 \cup T_2$,

$V = \{S_1, A_1, B_1 ...., X_1\} \cup \{S_2, A_2, B_2 ...., X_2\}$

Since, all productions of G_1 and G_2 including $S \rightarrow S_1S_2$ are in context-free form, so G is a CFG.

Language Generated by G :

$L(G)$ = Language generated from S_1 followed by language generated from S_2

$= L(G_1)\ L(G_2)$ (Since, S_1 and S_2 are starting symbols of G_1 and G_2 respectively).

$= L_1L_2$ (Since, G_1 produces L_1 and G_2 produces L_2 .)

Hence, statement of the theorem is proved.

Example : Consider the CFGs $S \to aSb \mid ab$ and $S \to cSdd \mid cdd$, which generate languages L_1 and L_2 respectively. Construct grammar for $L = L_1L_2$.

Solution :

Let G_1 generates L_1 and G_2 generates L_2 and $G = (V, T, P, S)$ generates $L = L_1L_2$. Renaming the variables of G_1 and G_2, we get

$V_1 = \{S_1\}$ and $V_2 = \{S_2\}$, where S_1 - productions are : $S_1 \to aS_1b \mid ab$, and S_2 - productions are : $S_2 \to cS_2dd \mid cdd$.

We define G as follows :

$V = \{S, S_1, S_2\}$, $\Sigma = \{Terminals\ of\ G_1\ or\ G_2\} = \{a, b, c, d\}$,

P includes : $S \to S_1S_2$, $S_1 \to aS_1b \mid ab$, and $S_2 \to cS_2dd \mid cdd$

$L = L_1L_2 = \{a^m b^n : m, n \geq 1\} \{c^n d^{2n} : n \geq 1\}$.

Theorem 5.8.3 : If L is a CFL generated by grammar $G = (V, T, P, S)$, then Kleene closure of L denoted by $L*$ is also a CFL.

Proof : Let grammar $G' = (V, T, P', S')$ generates $L*$. We define G' based on given grammar G

$L* = \{\epsilon, L, LL, LLL, ...\}$, since $L*$ includes null string, so G' has production : $S' \to \epsilon$ and from other productions, G' has to generate multiples of L. So, we have two recursive S' - productions : $S' \to SS' \mid S'S$, where S is the starting symbol of G

So, $P' = \{S' \to \epsilon \mid SS' \mid S'S\} \cup \{S - \text{productions of grammar } G\}$

Since, all productions of G' are in context-free form, so G' is a CFG.

Language generated by G' :

$L(G') = \{\epsilon, L, LL, LLL, ...\} = L*$

Thus, statement of theorem is proved.

Example : Consider the CFGs $S \to aSa \mid aa$, which generates $L = \{a^{2n} : n \geq 1\}$. Construct a grammar, which generates $L*$.

Solution :

Let $G' = (V, T, P', S')$ generates $L*$. We define the productions of G' as follows :

$S' \to \epsilon \mid SS' \mid S'S$, where $S \to aSa \mid aa$

Language generated by G' :

$S' \Rightarrow \in$

Hence, $\in$ is in $L(G')$.

$S' \Rightarrow S'S$ (Using $S' \rightarrow S'S$)

$\Rightarrow S'SS$ (Using $S' \rightarrow S'S$)

....

....

$\Rightarrow S'SS \ldots n$ times (Using $S' \rightarrow S'S$ n times)

$\Rightarrow \in SS \ldots n$ times (Using $S' \rightarrow \in$)

$\Rightarrow SS \ldots n$ times

$\overset{*}{\Rightarrow} LL \ldots n$ times (Since, G generates language L and S is the starting symbol of G.)

$= L^+$

So, $L(G') = \{\in\} \cup L^+ = L^*$

Theorem 5.8.4 : If L_1 and L_2 are two CFLs, then intersection of L_1 and L_2 denoted by $L_1 \cap L_2$ may or may not be a CFL.

Proof : We will discuss some examples, which prove the theorem.

Example 1 : Consider the CFLs $L_1 = \{a^n b^n c^m : m, n \geq 1\}$ and $L_2 = \{a^m b^n c^n : m, n \geq 1\}$, then intersection of L_1 and L_2 is not a CFL.

Solution:

$L_1 = \{abc, aabbcc, aaabbbccc, \ldots\}$ and $L_2 = \{abc, abbcc, aabbcc, aabbbccc, aaabbbccc, \ldots\}$

So, $L_1 \cap L_2 = \{abc, aabbcc, aaabbbccc, \ldots\}$

$= \{a^n b^n c^n : n \geq 1\}$

Clearly, $L_1 \cap L_2$ is not a CFL.

Example 2 : Consider the CFLs $L_1 = \{a^n b^n : n \geq 1\}$ and $L_2 = \{a^p b^q : p, q \geq 1\}$, then intersection of L_1 and L_2 is a CFL.

Solution :

$L_1 = \{ab, aabb, aaabbb, \ldots\}$ and $L_2 = \{ab, aab, aabb, abbb, aabbb, aaabbb, \ldots\}$

So, $L_1 \cap L_2 = \{ab, aabb, aaabbb, \ldots\} = \{a^k b^k : k \geq 1\}$

Clearly, $L_1 \cap L_2$ is a CFL.

Theorem 5.8.5 : If L is a CFL over some alphabet T, then complement of L denoted by $T^* - L$ may or may not be a CFL.

Proof :

We will discuss some mathematical identities to prove this theorem. Let us assume that complement of a CFL is also CFL. It means, $\overline{L} = T^* - L$ is CFL.

Let R and S are two CFLs over T, then we know that

$R \cap S = T^* - (\overline{R} \cup \overline{S})$ (De Morgan's law)

Since, we have assumed that complement of CFL is also a CFL, so $\overline{R}$ and $\overline{S}$ are CFLs and hence $P = \overline{R} \cup \overline{S}$ is a CFL (P is union of two CFLs).

So, $R \cap S = T^* - P$

or, $R \cap S = \overline{P}$

Since, P is a CFL, so $\overline{P}$ is a CFL.

Thus, $R \cap S$ is a CFL i.e., intersection of CFLs R and S is a CFL.

But, according to Theorem 5.8.4, $R \cap S$ may or may not be a CFL. So, our assumption about complement of a CFL is not hundred percent correct.

Since, intersection and complement are interchangeable using De Morgan's law, so whatever the truth about intersection we have proved that is also applicable to complement. Therefore, we conclude that complement of a CFL may or may not be a CFL.

We will discuss some examples, which prove the theorem.

Example 1 :

Consider a CFL L over $T = \{a, b\}$ which contains all the strings that not have the number of a's and b's equal or if number of a's and b's are equal then no two a's or b's are consecutive, then $T^* - L$ is a CFL.

Solution :

L = { All strings over $\{a, b\}$ not having number of a's and b's equal } or {All strings over $\{a, b\}$ which have number of a's and b's equal but no two a's and b's are consecutive}

So, $L = \{\in, aab, baa, aaab, ...\} \cup \{ab, abab, baba, ...\}$

$= \{\in, ab, aab, baa, aaab, baaa, abab, baba, ...\}$

Simply, $L = (a + b) * -\{a^k b^k : k \geq 2\}$

So, $T^* - L = (a + b) * -((a + b) * -\{a^n b^n : n \geq 2\})$

= {All the words over $\{a, b\}$ having equal number of a's and b's and all a's and b's are consecutive}

$= \{a^k b^k : k \geq 2\}$

Clearly, $T * -L$ is a CFL.

Example 2 :

Consider a CFL L over $\{a, b, c\}$ having all the strings in which number of a's, number of b's and number of c's are not equal or if number of a's, b's and c's are equal then no two a's, b's and c's are consecutive, then T* - L is not a CFL.

Solution :

$L = \{\in, a, b, c, ab, ba, ac, ca, ...\} \cup \{abc, abcabc, acabcb, ...\}$

$= \{\in, a, b, c, ab, ba, ac, ca, aaa, bbb, ccc, abc, ...\}$

Simply, $L = (a + b + c) * -\{a^n b^n c^n : n \geq 2\}$

Let $T = \{a, b, c\}$ then

$T^* - L = \{aabbcc \ , aaabbbccc \ , ...\}$

= { All the words over $\{a, b, c\}$ having equal number of a's, b's and c's and all a's, b's and c's are consecutive}

$= \{a^n b^n c^n : n \geq 2\}$

Clearly, $T^* - L$ is not a CFL.