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 :

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

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., $0 \le |\alpha| \le \infty$.

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 :
$S \to aSa \mid bSb \mid \in$. Check the productions and find the language generated.

Solution :

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

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

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

Since, all productions are of the form $A \to \alpha$, 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

$C \rightarrow$ 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

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

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

$$S \rightarrow aSb | \in \quad \text{.....(1)}$$

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

$$A \rightarrow aA | a$$

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

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

$$P = \{$$

$$S_1 \rightarrow AS|SB$$

$$S \rightarrow aSb \mid \in$$

$$A \rightarrow aA|a$$

$$B \rightarrow bB|b$$

$$\} \quad S_1 \text{ is the start symbol}$$

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

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

$$P = \{ \quad S \rightarrow aB \mid bA$$

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

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

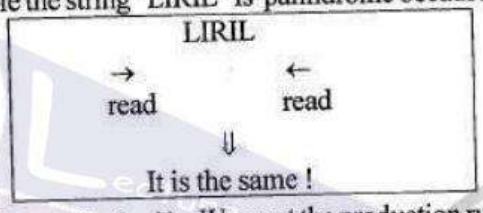
$\}$ S is the start symbol

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



Since the language L is over $T = \{a, b\}$. We want the production rules to be build a's and b's. As ϵ 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 $S \rightarrow a\ S\ a$

$S \rightarrow b\ S\ b$

$S \rightarrow a$

$S \rightarrow b$

$S \rightarrow \epsilon$

The string **abaaba** can be derived as

$$S \rightarrow aSa$$

$$\rightarrow ab\ Sba$$

$$\rightarrow ab\ a\ Sa\ ba$$

$$\rightarrow ab\ a\ \epsilon a\ ba$$

$$\rightarrow ab\ a\ a\ ba$$

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 → N | SN

N → D | ND

S → + | - | $\in$

D → 0 | 1 | 2 | ..... | 9

}

S = I which is the start symbol

Example 8 : Obtain the grammar to generate the language

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

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

$$A \rightarrow 01 | 0A1$$

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

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

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 = {

$S \rightarrow A \mid S2$

$A \rightarrow 01 \mid 0A1$

} 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

$$A \rightarrow 0A1 \mid \in$$

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

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

Finally if we apply the production

$$A \rightarrow \in$$

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

$$S \rightarrow A1$$

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 → A 1

A → 0 A 1 | ∈

} 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

$$S \rightarrow \in$$

$$S \rightarrow aSb$$

$$S \rightarrow bSa$$

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 :

$$S \Rightarrow aSb \qquad (\text{By applying } S \rightarrow aSb)$$

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

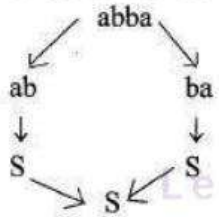
Similarly, the substring 'ba' can be generated from S and the derivation is shown below :

$$S \Rightarrow bSa \qquad (\text{By applying } S \rightarrow bSa)$$

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

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

$$S \rightarrow SS$$

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

$$V = \{ S \} \quad , \quad T = \{ a, b \}$$

$$P = \{ \; S \to \in$$
$$S \to aSb$$
$$S \to bSa$$
$$S \to SS$$
$$\}$$ 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 $S \underset{L}{\overset{*}{\Rightarrow}} w$ 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 $S \underset{R}{\overset{*}{\Rightarrow}} w$ is called rightmost derivation if and only if all steps involved in derivation have rightmost variable replacement only.

Example 1 : Consider the grammar $S \to S + S \mid S * S \mid a \mid b$. 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 $S \to S * S$)

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

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

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

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

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

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

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

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

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

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

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

Solution :

Leftmost derivation for $w = aaabbabbba$:

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

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

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

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

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

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

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

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

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

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

Rightmost derivation for $w = aaabbabbba$

$S \Rightarrow aB$ (Using $S \rightarrow aB$ 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 $B \rightarrow aBB$)

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

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

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

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

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

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

$\Rightarrow aaabbAbbba$ (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 $S \rightarrow S + S$

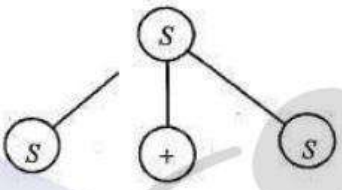


Figure (a)

For the production $S \rightarrow S * S$

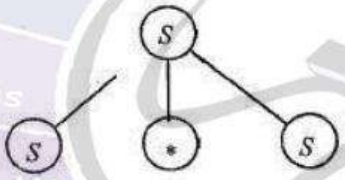


Figure (b)

For the production $S \rightarrow a$

Figure (c)

For the production $S \rightarrow b$

Figure (d)

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

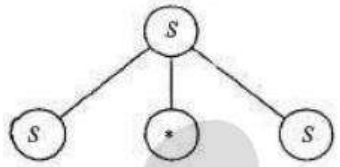
1. The root has label S (the starting symbol),
2. The all internal vertices (or nodes) are labeled with variables,
3. The leaves or terminal nodes are labeled with $\in$ or terminal symbols,
4. If $A \rightarrow \alpha_1\alpha_2\alpha_3 \ldots \alpha_n$ is a production in G, then A becomes the parent 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 $w = a * b + a$.

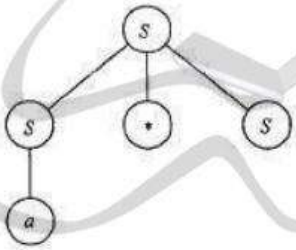
Solution : The derivation tree or parse tree is shown in below figure.

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

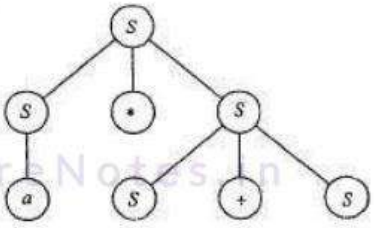
$S \Rightarrow S * S$ (Using $S \rightarrow S * S$)



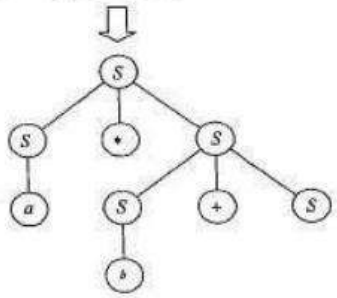
$\Rightarrow a * S$ (The first left hand symbol is a, so using $S \rightarrow a$)



$\Rightarrow a * S + S$ (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 $S \rightarrow a$)

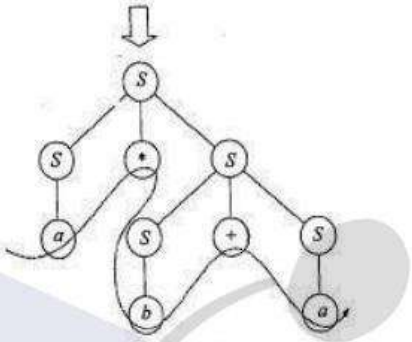


Figure : Parse tree for $a * b + a$

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.

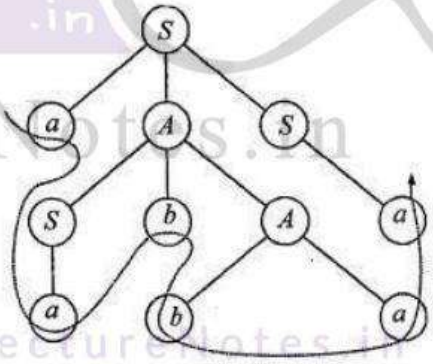


Figure : Parse tree yielding $aabbaa$

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 .

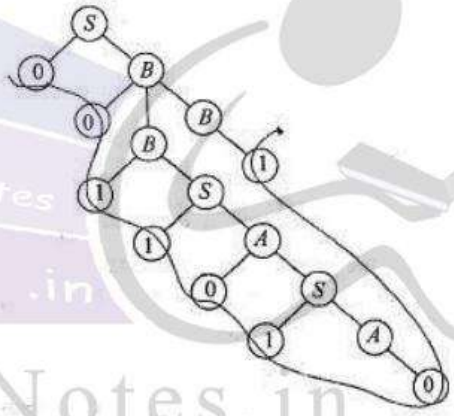


Figure : Derivation tree for 00110101

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 $S \rightarrow S + S \mid S * S \mid a \mid b$ and string $w = a * a + b$, and derivations as follows:

Solution :

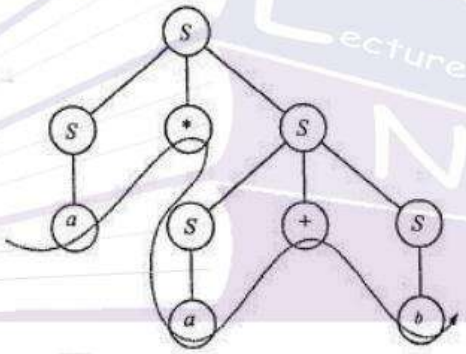
First leftmost derivation for $w = a * a + b$

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

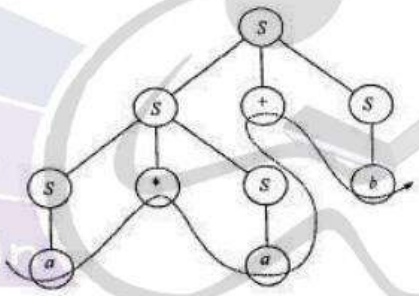
Second leftmost derivation for $w = a * a + b$

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

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) $S \rightarrow SS \mid a \mid b$

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

Solution :

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

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

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

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

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

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

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

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

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

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

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

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

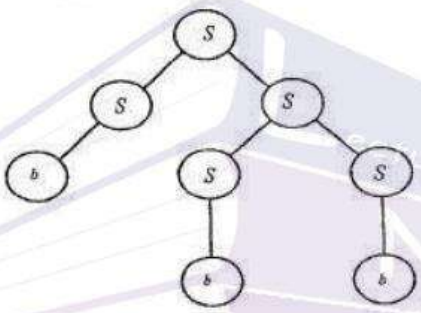


Figure (a) Parse tree for bbb

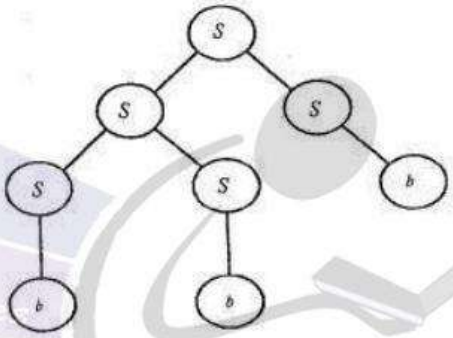


Figure (b) Parse tree for bbb

So, the given grammar is ambiguous.

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

$S \underset{L}{\Rightarrow} A$

$\underset{L}{\Rightarrow} ab$ (Using $A \rightarrow ab$)

$S \underset{L}{\Rightarrow} B$

$\underset{L}{\Rightarrow} abB$ (Using $B \rightarrow abB$)

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

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

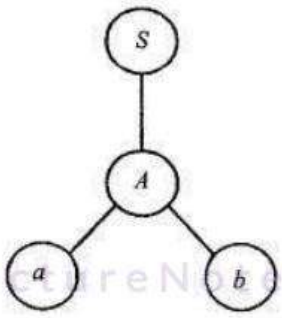


Figure (c) Parse tree for $w = ab$

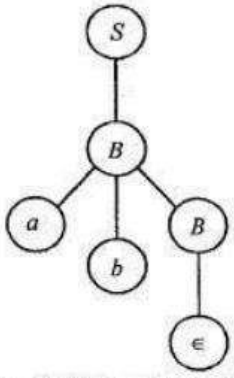


Figure (d) Parse tree for $w = ab$

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 $A \to A\alpha_1 | A\alpha_2 | A\alpha_3 ..... A\alpha_n | \beta_1 | \beta_2 | \beta_3 ..... \beta_m$

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

$$A \to \beta_1 A^1 | \beta_2 A^1 | \beta_3 A^1 | ..... \beta_m A^1$$

$$A^1 \to \alpha_1 A^1 | \alpha_2 A^1 | \alpha_3 A^1 | ..... | \alpha_n A^1 | \in$$

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

Example 1 : Eliminate left recursion from the following grammar

$$E \to E + T | T$$

$$T \to T * F | F$$

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

Solution : The left recursion 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 $A^1 \to \alpha_i A^1 \mid \in$
$E \to E+T \mid T$	$A = E$ $\alpha_1 = +T$ $\beta_1 = T$	$E \to TE^1$ $E^1 \to +TE^1 \mid \in$
$T \to T*F \mid F$	$A = T$ $\alpha_1 = *F$ $\beta_1 = F$	$T \to FT^1$ $T^1 \to *FT^1 \mid \in$
$F \to (E) \mid id$	Not applicable	$F \to (E) \mid id$

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

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

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

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

The grammar obtained after eliminating left recursion is

$$S \rightarrow Ab \mid a$$

$$A \rightarrow aaA^1$$

$$A^1 \rightarrow bA^1 \mid baA^1 \mid \in$$

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 $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n$, 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 :

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

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

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

Solution :

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

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 :

$S \Rightarrow aS'$

$\Rightarrow a*SS'$

$\Rightarrow a*aS'S'$

$\Rightarrow a*a+SS'S'$

$\Rightarrow a*a+aS'S'S'$

$\Rightarrow a*a+a \in S'S'$

$\Rightarrow a*a+a \in S'$

$\Rightarrow a*a+a \in (\equiv a*a+a)$

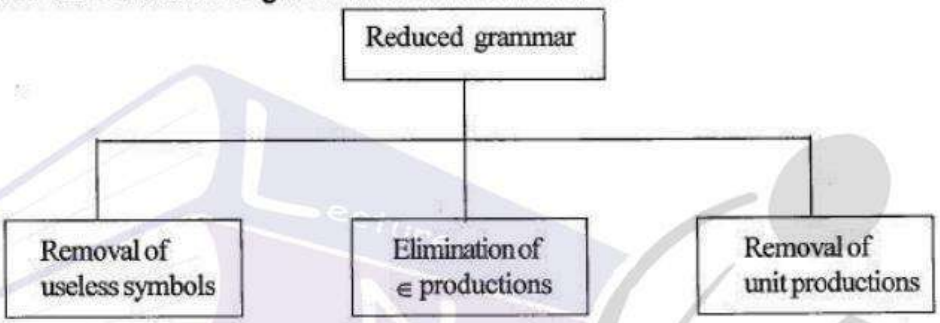
So, we conclude that removal of left recursion (and left factoring also) helps in removal of ambiguity of the ambiguous grammars.

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.

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

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 α 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 → aA| bB
A → aA | a
B → bB
D → ab | Ea
E → aC | 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
$S \rightarrow aA$	a	S, A
$A \rightarrow a \mid aA$	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 = {

$S \rightarrow aA$

$A \rightarrow a \mid aA$

} 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

S → aA | a | Bb | cC
A → aB
B → a | Aa
C → cCD
D → ddd

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 = {

S → a | Bb | aA
B → a | Aa
D → ddd
A → aB

}

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 → c | ∈
D → d

In this grammar, the productions

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

are ∈ - 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 $A \rightarrow B_1 B_2 \ldots\ldots B_n$ is a production in P, and if $B_1 B_2 \ldots\ldots B_n$ 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 ∈ - productions, the language may not derive a language containing empty string. So, in such cases, the ∈ - 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 ∈ – 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

$$A \rightarrow X_1X_2X_3 \ldots\ldots\ldots X_n,\ n \geq 1$$

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)
$S \rightarrow ABCa$	$S \rightarrow ABCa \mid BCa \mid ACa \mid ABa \mid Ca \mid Aa \mid Ba \mid a$
$S \rightarrow bD$	$S \rightarrow bD$
$A \rightarrow BC \mid b$	$A \rightarrow BC \mid B \mid C \mid b$
$B \rightarrow b \mid \in$	$B \rightarrow b$
$C \rightarrow c \mid \in$	$C \rightarrow c$
$D \rightarrow d$	$D \rightarrow d$

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 = {
S → ABCa | BCa | ACa | ABaCa | Aa | Ba | a | bD
A → BC | B | C | b
B → b
C → c
D → d
} S is the start symbol

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

S → BAAB
A → 0A2 | 2A0 | $\in$
B → AB | 1B | $\in$

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 → BAAB	S → BAAB | AAB | BAB | BAA | AB | BB | BA | AA | A | B
A → 0A 2	A → 0A2 | 02
A → 2A0	A → 2A0 | 20
B → AB	B → AB | B | A
B → 1 B	B → 1 B | 1

We can delete the productions of the form A → A. In P_1, the production $B \to B$ 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 → BAAB | AAB | BAB | BAA | AB | BB | BA | AA | A | B
A → 0A2 | 02 | 2A0 | 20
B → AB | A | 1B | 1
} S is the start symbol

5.5.3 Eliminating unit productions

Consider the production $A \to B$. 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

$$A \to B$$

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

   $A \Rightarrow^* B$

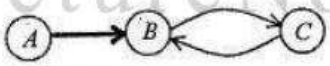
   i. e., in the derivation process from A, if we encounter only one variable in a sentential form say B (no terminals should be there), obtain all such variables.
4. Obtain a dependency graph. For example, if we have the productions

   $A \rightarrow B$

   $B \rightarrow C$

   $C \rightarrow B$

   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

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

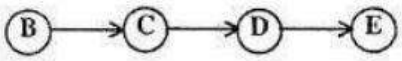
Solution : The non unit productions of the grammar G are shown below :

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

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 :



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

$$E \rightarrow d|Ab \quad \text{..... (2)}$$

Since $D \Rightarrow^* E$,

$$D \rightarrow d \mid Ab$$

The resulting D productions are

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

$D \rightarrow d \mid Ab$ .....(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,

$C \rightarrow d \mid Ab$

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,

$C \rightarrow bC$

$C \rightarrow d \mid Ab$ .....(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 :

$B \rightarrow b$

$B \rightarrow d \mid Ab$

$B \rightarrow bC$ ..... (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

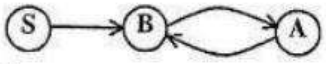
Example 2 : Eliminate unit productions from the grammar

S	→	A 0 | B
B	→	A | 11
A	→	0 | 12 | B

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

S → B
B → A
A → B

The dependency graph for the unit productions is shown below.



The non unit productions are :

S → A0
B → 11
A → 0 | 12 ..... (1)

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

S → 11 | 0 | 12
B → 0 | 12
A → 11 .....(2)

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

V_1 = { S, A, B } , T_1 = { 0, 1, 2 }
P_1 = { S → A0 | 11 | 0 | 12
A → 0 | 12 | 11
B → 11 | 0 | 12
} S is the start symbol

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

1. ϵ – 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 ϵ 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.