REGULAR GRAMMAR :-

✱ Definition Of a Grammar :-

Grammar is the set of rules that explains how the words are used in a language.

A grammar G can be formally written as 4-Tuple (V_n, Σ, P, S) where -

V_n - is a finite non empty set of variables. The elements of V_n are called the nonterminals or variables.

Σ - is a finite non-empty set called terminal symbols.

P - is a production rules for Terminals and Non-terminals. A production rule has the form $\alpha \to \beta$. $\alpha \to \beta$ read as "α derives β", where $\alpha, \beta \in (V_n \cup \Sigma)$. α has atleast one symbol, from V_n. β has no restriction. The element of P are called production or production rules.

S - is a special variable called start symbol. $S \in N$.

★ Notations :-

In grammar we can use the following notations-

★ S : the starting symbol unless otherwise specified.

★ Non-terminal is represented by capital letters such as A, B, C, ...

★ Terminal is represented by small letters such as a, b, c, ...

★ Two sides (LHS and RHS) of the production rules are separated by → symbol.

★ w, x, y, z denotes the string of terminals.

★ α, β, δ denotes the string of terminals or non-terminals or both including null string.

★ Any symbol to the power of 0 is equivalent to null i.e. $\alpha^0 = \varepsilon$, where $\alpha \in (V_n \cup \varepsilon)$.

Ex- Grammar $G1 - (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to a, B \to b\})$

Here,

→ S, A, B are Non terminal symbols.

→ a and b are Terminal symbols.

→ s is the start symbol, $s \in N$

→ Productions -

P : $S \to AB$, $A \to a$, $B \to b$

★ Production Rules :-

ⓐ **Inversion Rule :-** This rule says that the inverse operation is not possible.

For ex -

If $S \to AB$ is a production then $AB \to S$ is not allowed.

ⓑ **Reverse substitution Rule :-** This rule says that reverse substitution cannot take place.

For ex -

If $S \to AB$ i.e. S can be replaced by AB but vice-versa can't be possible.

★ Language generated by Grammar G_1 :-

The language generated by grammar G_1 is denoted by $L(G_1)$. It is the set of all terminals that is derived by G_1.

Example ① If $G = (\{S\}, \{a\}, \{S \to SS / a / \varepsilon\}, S)$ then find $L(G)$.

Solution -

Let $G_1 = (V_n, \Sigma, P, S)$ where,

$V_n = \{S\}$, $\Sigma = \{a\}$

$P = \{P_1 : S \to SS, P_2 : S \to a, P_3 : S \to \varepsilon\}$

Using rule $P_3 : S \to \varepsilon$

Null string (ε) is also present in $L(G)$.

Using rule $P_2 : S \to a$

Means string 'a' is in $L(G)$.

Using rule $P_2 : S \to SS$

$S \to SS$
$\to SSS$ (replacing rightmost S by SS)
$\to SSSSSS.....$ n times (using (n-1) step derivation)
$\to aaaaaa....$ n times (replacing leftmost S by 'a' n times)

OR

$\to \varepsilon\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon.....$ n times (replacing leftmost S by ε n-times)

Also ε, a and a^n for $n >= 1$ is in $L(G_1)$
Therefore,

$$L(G_1) = \{\varepsilon, a, aa, aaa, aaaa, ...\}$$

$$L(G_1) = \{a^n \mid n >= 0\} \text{ or } a^*$$

Ex (2) A grammar G_1 has the following production

$S \to AB$
$A \to aA / a$
$B \to bB / b$
Find $L(G_1)$.

Solution – $G_1 = (V_n, \Sigma, P, S)$
where,
$V_n = \{S, A, B\}$, $\Sigma = \{a, b\}$
$P = \{ P_1 : S \to AB,$
$P_2 : A \to aA,$
$P_3 : A \to a,$
$P_4 : B \to bB,$
$P_5 : B \to b \}$

Using rule $P_3 : A \to a$
Using rule $P_5 : B \to b$
Therefore,
$S \to ab$
we know that
$P_1 : S \to AB$
$S \to aAbB$ {using rule P_2 & P_4}
$S \to aaAbbB$
$S \to aaaA bbbB$
$S \to aaa\ldots$ n times $bbbb\ldots$ m times
i.e.
$S \to a^n b^m$

Therefore string $a^n b^m$ for $a >= 1$ are in $L(G)$

$$L(G) = \{ ab, aabb, abb, aaab, \ldots \}$$

$$L(G) = \{ a^n b^m \mid n, m >= 1 \}$$

★ Grammar generated by Language $L(G)$:-

For generating the grammar from a given language we will follow the following steps-

step① Expand the given language.
step② Introduce necessary variables, production rules & recursive production rules.
step③ Verify the language generated by grammar.

Ex① Construct a grammar for language

$$L = \{ a^n b^n c^i \mid n >= 1, i >= 0 \}$$

Solution- Case 1 when $i = 0$

$$L_1 = \{ a^n b^n \mid n >= 1 \}$$

$$L_1 = \{ ab, aabb, aaabbb, \ldots\ldots \}$$

Case 2: When $i >= 1$

$L_2 = \{ a^n b^n c^i / n >= 1, i >= 1 \}$

$L_2 = \{ abc, aabbcc, aabbc, ... \}$

So the language -

$L = L_1 \cup L_2$

$L = \{ ab, aabb, aaabbbb, ... \}$
$\cup \{ abc, aabbcc, aabbc, ... \}$

$L = \{ ab, abc, aabb, aabbcc, aabbc, ... \}$

Production rules for language L_1 -

P_1: $A \rightarrow ab$

P_2: $A \rightarrow aAb$ [P_2 is recursive production over A]

Production rule for language L_2 -

P_3: $S \rightarrow Sc$

P_4: $S \rightarrow A$

Hence the grammar is -

$G = (\{S, A\}, \{a, b, c\}, \{S \rightarrow A/Sc, A \rightarrow aAb/ab\}, S)$

Ex ② Construct a grammar for the language
$L = \{wcw^R / w \in (a,b)^+\}$

Sol:- Let the grammar $G = (V_n, \Sigma, P, S)$

In the given language L, every string w is symmetric over the symbol c. And $(a,b)^+$ indicates that w contains at least one symbol. Hence the language.

$L = \{aca, bcb, abcba, aacaa, bbcbb, \ldots\}$

Production rules-

P_1: $S \to asa$ {for string 'a'}
P_2: $S \to bSb$ {for string 'b'}
P_3: $X \to c$ {for string 'c'}
P_4: $S \to aXa$
P_5: $S \to bXb$

After concatenation the production rules-

$S \to asa / bSb / aXa / bXb$
$X \to c$

Thus the grammar G -

$$G = (\{S, X\}, \{a, b, c\}, S \to asa/bSb/aXa/bXb, X \to c\}, S)$$

Regular Language:-

A language is said to be regular if there exists a finite acceptor for it. So, every regular language can be accepted by some DFA or NFA/NDFA.

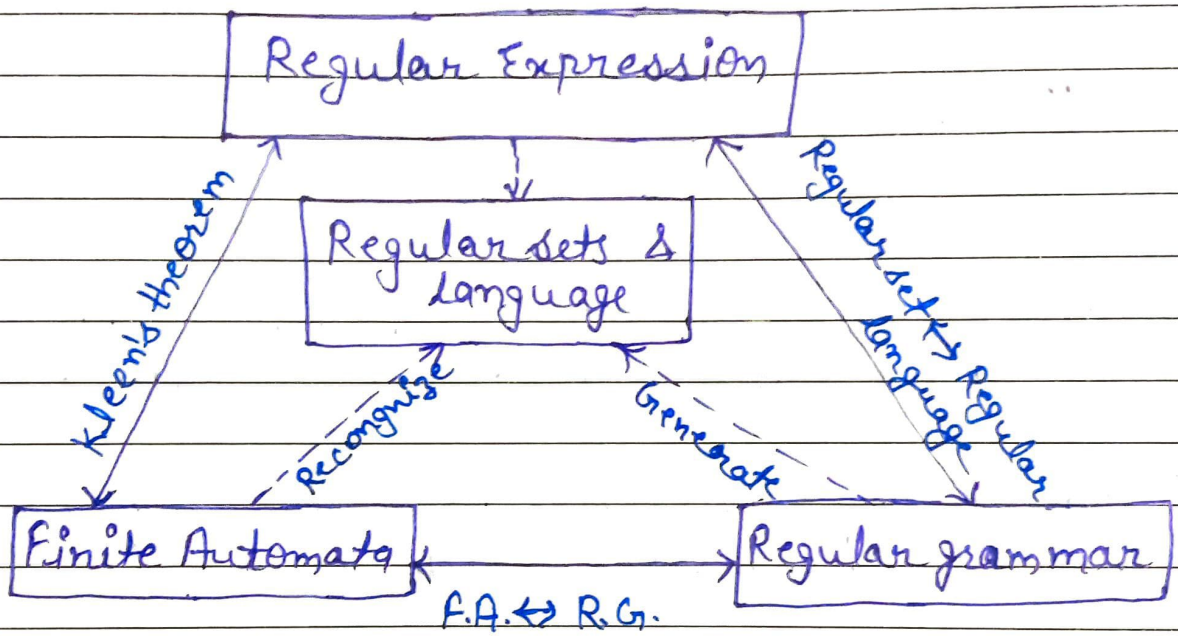


Fig- Relationship between FA and regular language.

From this figure we conclude that the regular grammar and regular expressions are generated by regular languages and regular sets. For converting regular expression to finite automata or vice-versa we use Kleene's theorem.

Regular Grammar-

Generally, a grammar Gr is a quadruples (V_n, Σ, P, S) where,

- V_n - is a finite nonempty set of non-terminals
- Σ - is a finite nonempty set of terminals
- P - is a finite nonempty set of productions
- S - is a start symbol.

"If the productions of the grammar Gr are in the form that are -

→ Atmost one non terminal symbol appears on the right hand side of the production.

→ Atmost one non terminal symbol appears on the left hand side of the production.

→ Non-terminal symbol which is on the right-hand side, must be either the leftmost or the rightmost symbol.

then Gr is known as regular grammar!"

CONTEXT FREE GRAMMARS

★ **Definition** -

A grammar $G = (V_n, \Sigma, P, S)$ is said to be context free, if all productions in P have the form - $\alpha \rightarrow \beta$

where,

$\alpha \in V_n$ and $\beta \in (V_n \cup \Sigma)^*$

&

$|\alpha| = 1$

$|\beta| =$ infinite.

Here -

V_n - set of variables or nonterminals.

Σ - set of terminal alphabet

P - set of production rule.

S - start symbol & $S \in V_n$.

A language is said to be context free if and only if there is a context free grammar G_1 such that $L = L(G_1)$

The productions in a CFG are restricted in two ways -

① The left hand side must be a single non-terminal

② RHS can be any combination of terminal & non-terminal.

Every RG is CFG, so a regular language is also a context free one.

★ **Derivation :-** Derivation is a sequence of production rules. It is used to get the input string through these production rules. During parsing, we have to take two decisions. These are :-

→ We have to decide the non-terminal which is to be replaced.

→ We have to decide the production rule by which the non-terminal will be replaced.

① **Leftmost Derivation :-** In the leftmost derivation, the input is scanned and replaced with the production rule from left to right. So in leftmost derivation we read the input string from left to right.

Ex- Production rules :-

$E = E + E$

$E = E - E$

$E = a / b$

Input -

$a - b + a$

The leftmost derivation is -

$E = E+E$
$E = E-E+E$
$E = a-E+E$
$E = a-b+E$
$E = a-b+a$

② Rightmost Derivation:- In rightmost derivation, the input is scanned and replaced with the production rule from right to left. so in rightmost derivation, we read the input string from right to left.

Ex- Production rules -

$E = E+E$
$E = E-E$
$E = a|b$

Input

$a-b+a$

The rightmost derivation is -

$E = E-E$
$E = E-E+E$
$E = E-E+a$
$E = E-b+a$
$E = a-b+a$

Ex ① Derive the string "abb" for leftmost and rightmost derivation using a CFG given by-

$S \rightarrow AB / \varepsilon$

$A \rightarrow aB$

$B \rightarrow Sb$

Solution- Leftmost derivation-

$S \rightarrow AB$

$\rightarrow aBB$

$\rightarrow aSbB$

$\rightarrow a\varepsilon bB$

$\rightarrow abSb$

$\rightarrow ab\varepsilon b$

$\rightarrow abb$

Rightmost derivation-

$S \rightarrow AB$

$\rightarrow ASb$

$\rightarrow A\varepsilon b$

$\rightarrow aBb$

$\rightarrow aSbb$

$\rightarrow a\varepsilon bb$

$\rightarrow abb$

★ Derivation Tree :-

Derivation tree is a graphical representation for the derivation of the given production rules for a given CFG. It is the simple way to show how the derivation can be done to obtain some string from a given set of production rules. The derivation tree is also called a "parse tree".

A parse tree contains the following properties

① The root node is always a node indicating start symbols.
② The derivation is read from left to right
③ The leaf node is always terminal nodes.
④ The interior nodes are always the non-terminal nodes.

Ex① Production rules -

$E = E + E$

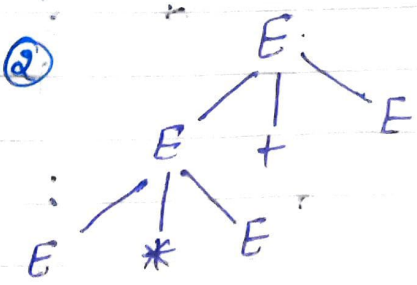
$E = E * E$

$E = a/b/c$

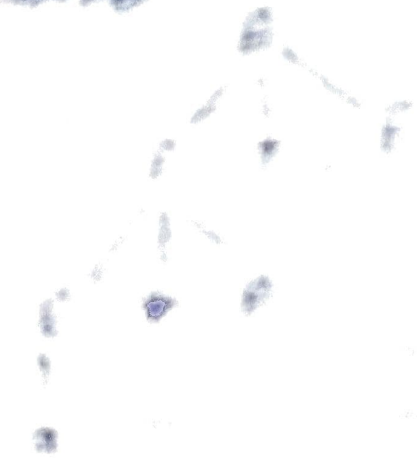
Input - $a * b + c$

Solution- step ①

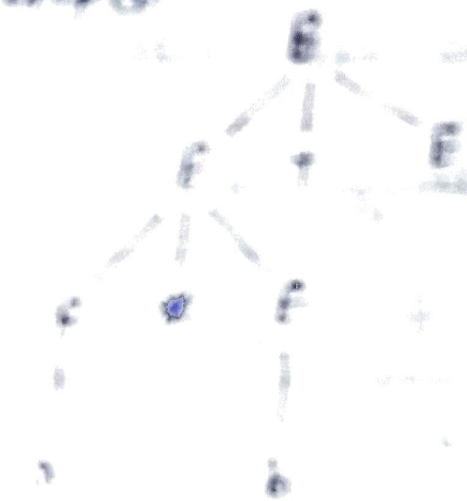
step②



Arbre a)



Arbre b)

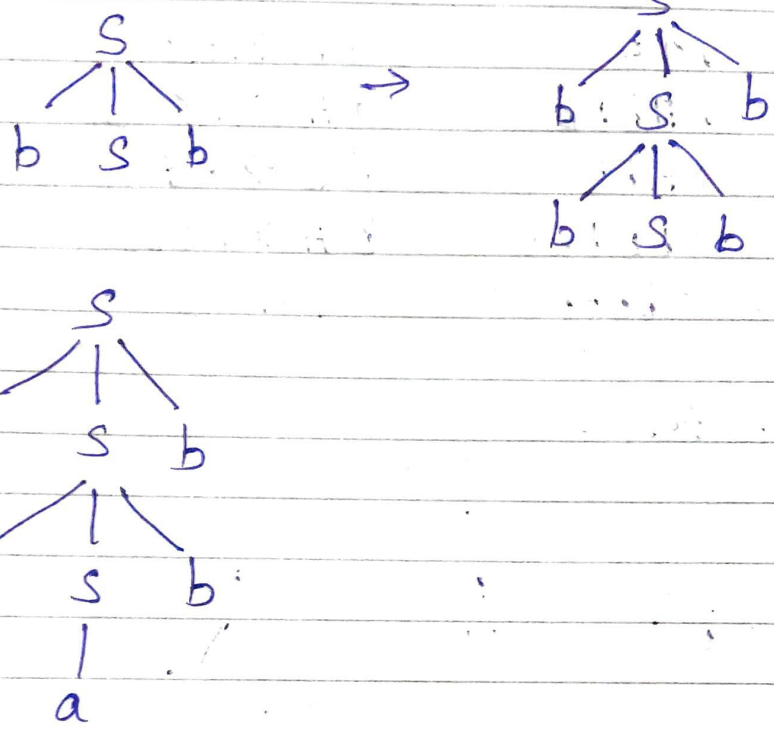


Arbre c)

Ex② Draw a derivation tree for the string "bbabb" from the CFG given by-

$S \rightarrow bSb \mid a \mid b$

Solution-



Ex③ Consider the grammar G, with productions

$S \rightarrow aAB$

$A \rightarrow bBb$

$B \rightarrow A \mid \varepsilon$

Find - ⓐ leftmost derivation ⓑ Rightmost derivation

ⓒ Parse tree

For string - "abbbb"

Solution Leftmost derivation -

$S \rightarrow aAB$

$S \rightarrow aaBbB$ $\{A \rightarrow bBb\}$

$S \rightarrow abAbB$ $\{B \rightarrow A\}$

$S \rightarrow abbBbbB$ $\{A \rightarrow bBb\}$

$S \rightarrow abbbbB$ $\{B \rightarrow \varepsilon\}$

$S \rightarrow abbbb$ $\{B \rightarrow \varepsilon\}$

Rightmost derivation -

$S \rightarrow aAB$ $\{B \rightarrow A\}$
$S \rightarrow aAA$ $\{A \rightarrow bBb\}$
$S \rightarrow aAbBb$ $\{B \rightarrow \varepsilon\}$
$S \rightarrow aAbb$ $\{A \rightarrow bBb\}$
$S \rightarrow abBbbb$ $\{B \rightarrow \varepsilon\}$
$S \rightarrow abbbb$

Parse tree -

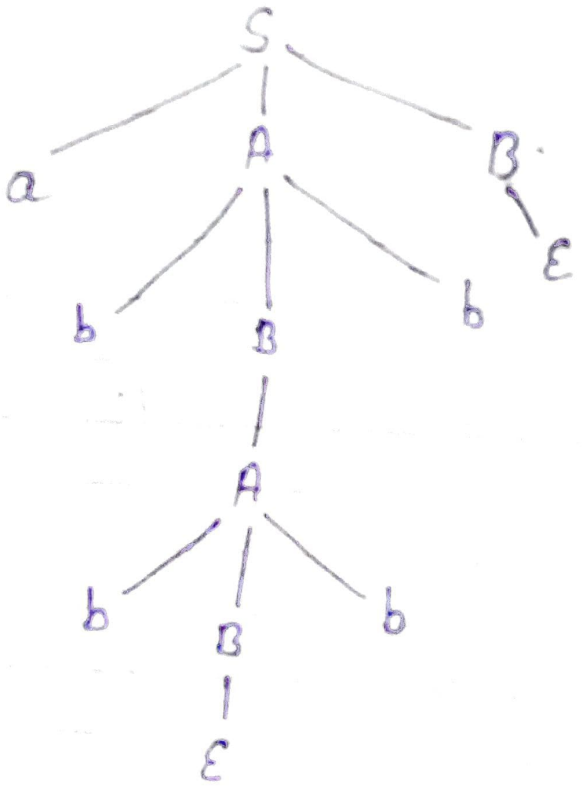


Fig - Parse tree for string $w = abbbb$

★ Ambiguity in Context Free Grammars-

A grammar is said to be ambiguous if there exists more than one leftmost or rightmost derivation or more than one parse tree for the given input string. If the grammar is not ambiguous, then it is called unambiguous.

If the grammar has ambiguity, then it is not good for compiler construction. No method can automatically detect and remove the ambiguity, but we can remove ambiguity by re-writing the whole grammar without ambiguity.

Ex① Show that the grammar
$S \rightarrow aS / Sa / a$ is ambiguous

Solution- Here we can generate string aaa from following parse tree.

① $S \rightarrow aS$
$S \rightarrow aaS \quad \{S \rightarrow aS\}$
$S \rightarrow aaa \quad \{S \rightarrow a\}$

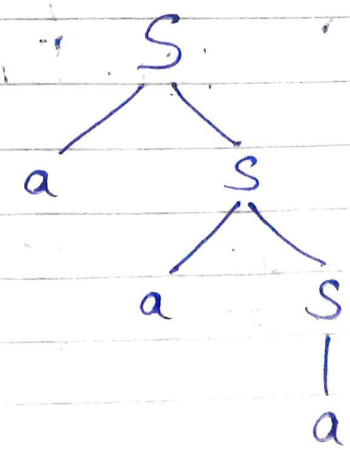


Fig. First parse tree for string $w = aaa$

(II) $S \to aS$ $\{S \to Sa\}$
$S \to aSa$ $\{S \to a\}$
$S \to aaa$

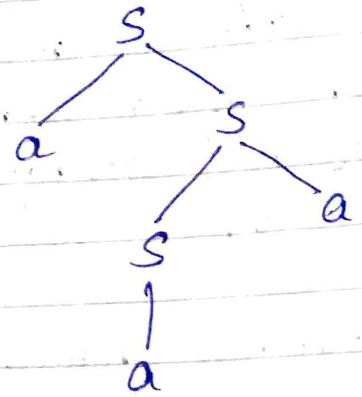


Fig- Second parse tree for string $w = aaa$

(III) $S \to Sa$
$S \to Saa$ $\{S \to Sa\}$
$S \to aaa$ $\{S \to a\}$

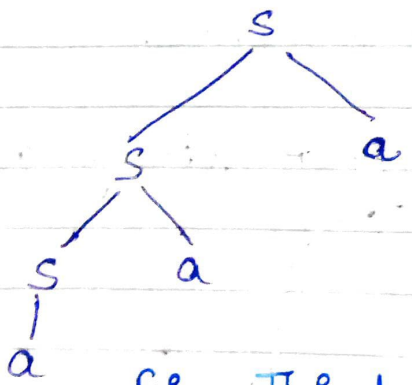


Fig- Third parse tree for string $w = aaa$

(IV) $S \to Sa$
$S \to aSa$ $\{S \to aS\}$
$S \to aaa$ $\{S \to a\}$

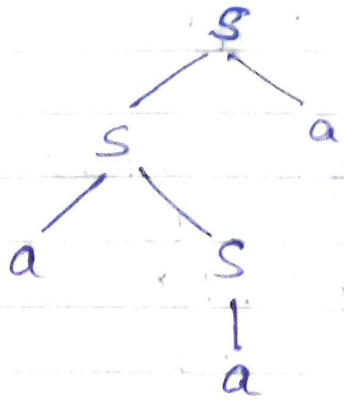


Fig- Fourth parse tree for string $w = aaa$

Since there are more than one parse tree for string aaa. The grammar is ambiguous.

Ex② Show that the grammar

$S \rightarrow S+S$

$S \rightarrow S*S$

$S \rightarrow a$

$S \rightarrow b$

$S \rightarrow c$.

is ambiguous.

Solution- let we consider string $w = a+b*c$

Leftmost derivation-

① $S \rightarrow S+S$

$S \rightarrow a+S$ $\{S \rightarrow a\}$

$S \rightarrow a+S*S$ $\{S \rightarrow S*S\}$

$S \rightarrow a+b*S$ $\{S \rightarrow b\}$

$S \rightarrow a+b*c$ $\{S \rightarrow c\}$

another leftmost derivation -

(II)

$S \rightarrow S*S$ $\quad \{S \rightarrow S+S\}$

$S \rightarrow S+S*S$ $\quad \{S \rightarrow a\}$

$S \rightarrow a+S*S$ $\quad \{S \rightarrow b\}$

$S \rightarrow a+b*S$ $\quad \{S \rightarrow \}$

$S \rightarrow a+b*c$

Derivation tree

(I)

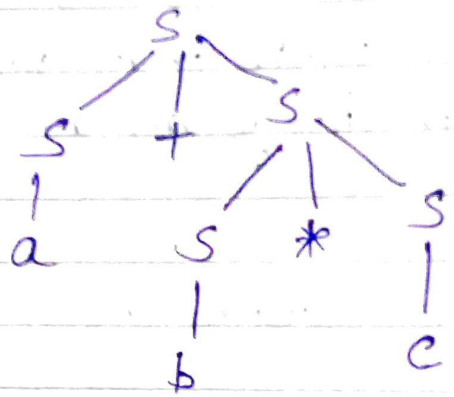


Fig- First parse tree for string w = a+b*c

(II)

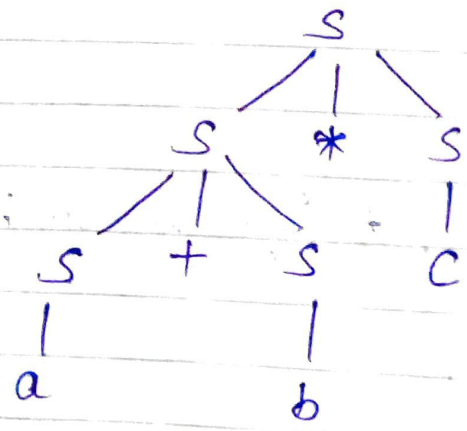


Fig- Second parse tree for string w = a+b*c

Ex ③ Consider the grammar $G = (V_n, \Sigma, P, S)$ with

$V_n = \{S, A\}$

$\Sigma = \{a, b\}$

$P = \{ S \to AA$

$A \to AAA$

$A \to a$

$A \to bA$

$A \to Ab \}$

show that the grammar is ambiguous

Solution- Let the string $w =$ babbab

① $S \to AA$ $\{A \to Ab\}$

$S \to AbA$ $\{A \to bA\}$

$S \to bAbA$ $\{A \to a\}$

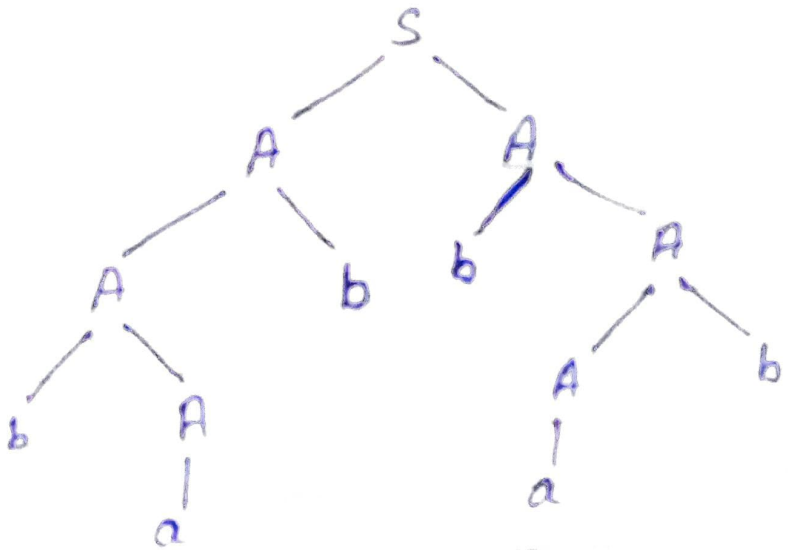
$S \to babA$ $\{A \to bA\}$

$S \to babbA$ $\{A \to Ab\}$

$S \to babbAb$ $\{A \to a\}$

$S \to babbab$

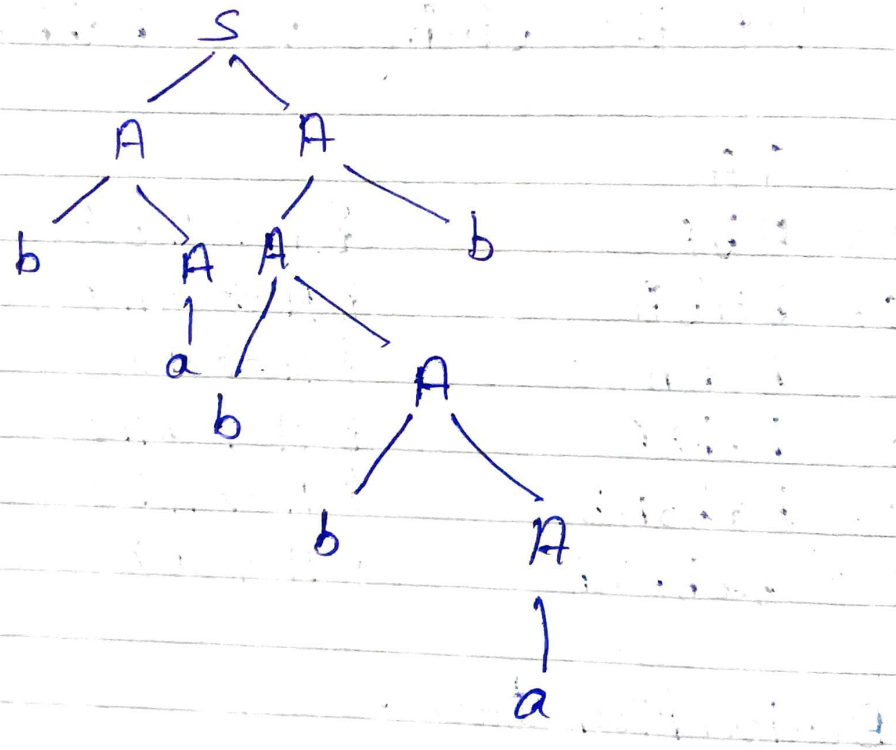
Derivation tree -



(11)

$S \rightarrow AA$	
$S \rightarrow bAA$	$\{A \rightarrow bA\}$
$S \rightarrow baA$	$\{A \rightarrow a\}$
$S \rightarrow baAb$	$\{A \rightarrow Ab\}$
$S \rightarrow babAb$	$\{A \rightarrow bA\}$
$S \rightarrow babbAb$	$\{A \rightarrow bA\}$
$S \rightarrow babbab$	$\{A \rightarrow a\}$

Derivation tree –



Unambiguous Grammar :-

A grammar can be unambiguous if the grammar does not contain ambiguity that means if it does not contain more than one leftmost derivation or more than one rightmost derivation or more than one parse tree for the given input string.

To convert ambiguous grammar to unambiguous grammar, we will follow some steps which are -

(I) If the left associative operators (+, -, *, /) are used in the production rule, then apply left recursion in the production rule. Left recursion means that the leftmost symbol on the right side is the same as the non-terminal on the left side.

For ex. $X \rightarrow Xa$

(II) If the right associative operator ($) is used in the production rule then apply right recursion in the production rule. Right recursion means that the rightmost symbol on the left side is the same as the non-terminal on the right side.

For ex.

$X \rightarrow aX$.

Ex① Consider a grammar G -

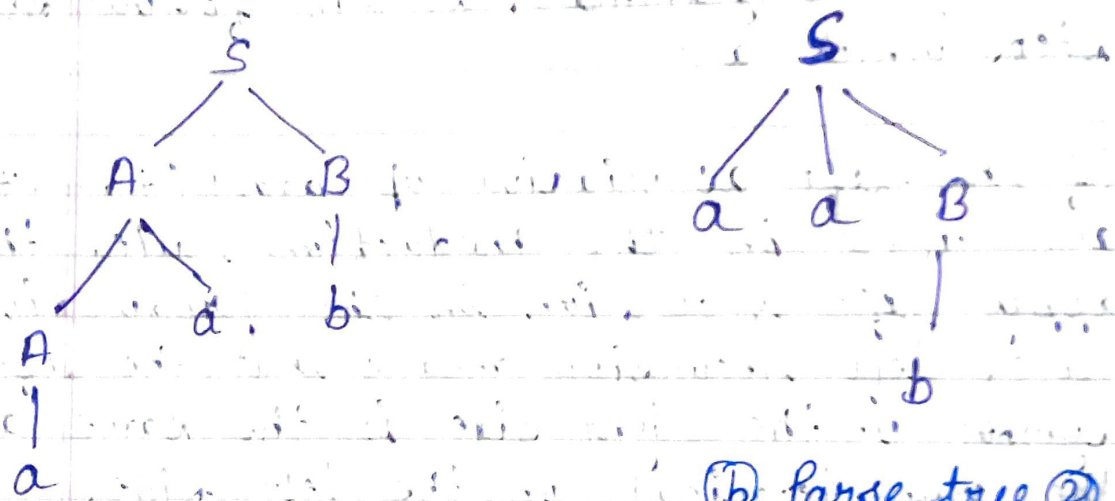
$S \rightarrow AB / aaB$

$A \rightarrow a / Aa$

$B \rightarrow b$

Determine whether the grammar G is ambiguous or not. If G is ambiguous, construct an unambiguous grammar equivalent to G.

Solution - For string "aaab"



ⓐ Parse tree ①

ⓑ Parse tree ②

There are two parse tree for deriving same string, so the given grammar is ambiguous

The unambiguous will be :-

$S \rightarrow AB$
$A \rightarrow Aa / a.$
$B \rightarrow b$

Ex ② $S \rightarrow ABA$
$A \rightarrow aA / \varepsilon$
$B \rightarrow bB / \varepsilon$

Show that the grammar is ambiguous or not. also find an equivalent unambiguous grammar.

Sol:-

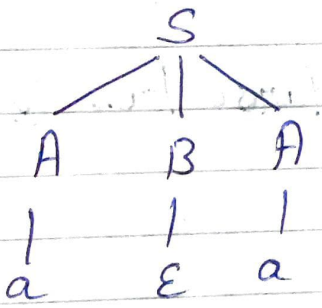


Fig- parse tree ①

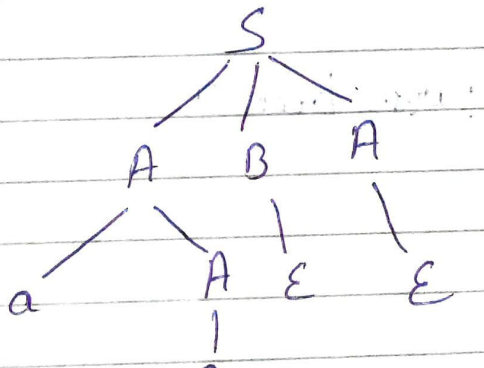


Fig- parse tree ②

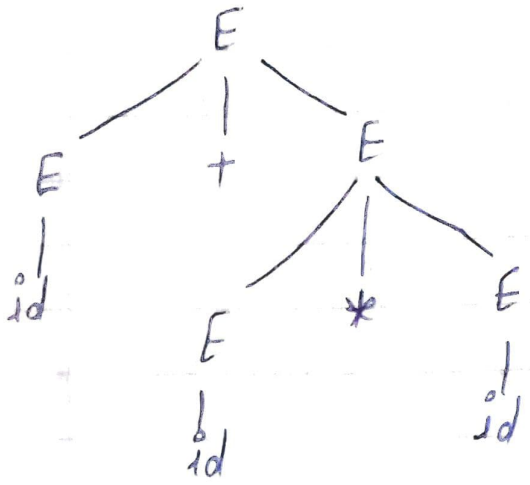
So the given grammar is ambiguous.

The unambiguous grammar is -

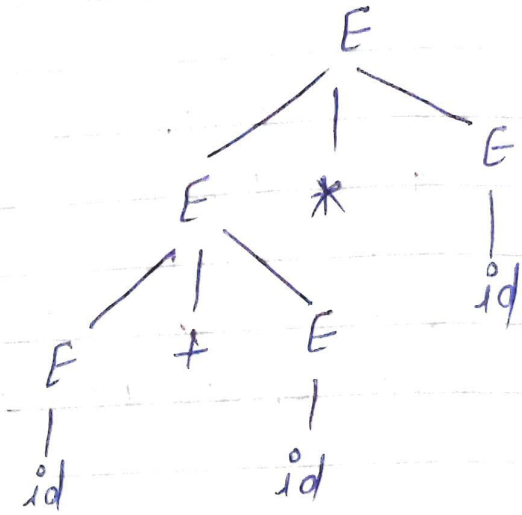
$S \rightarrow aXY \mid bYZ \mid \varepsilon$
$Z \rightarrow aZ \mid a$
$X \rightarrow aXY \mid a \mid \varepsilon$
$Y \rightarrow bYZ \mid b \mid \varepsilon$

Ex ③ $E \to E + E$
$E \to E + E$
$E \to id$

Sol – For string "id + id * id"



Parse tree ①



Parse tree ②

The unambiguous grammar will be -

$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to id$

Simplification of CFG:-

All the grammar are not always optimized that means the grammar may consist of some extra symbols (non-terminals). Having extra symbols, unnecessary increase the length of grammar. Simplification of grammar means reduction of grammar by removing useless symbols.

CFGs are simplified to some standard form so it will be easy to perform some of the compiler operations.

CFG simplification include -
1. Removal of Useless symbol
2. Elimination of ε Production
3. Removal of Unit Production

1. Removal of Useless symbols :-

Production from a grammar that can never take part in any derivation can be removed from grammar.

For ex- Let $G = (V_n, \Sigma, P, S)$
and P: $S \rightarrow aSb / \varepsilon / A$
$A \rightarrow aA$

Here the production $S \rightarrow A$ clearly plays no role, as A can not be transformed into a terminal string. Removing this production leave the language unaffected

Definition:- Let $G = (V_n, \Sigma, P, S)$ be a CFG. A variable $A \in V_n$ is said to be useful if and only if there is at least one $w \in L(G)$ such that

$$S \xrightarrow{*} xAy \xrightarrow{*} w$$

with x, y in $(V_n \cup \Sigma)^*$.

A variable is useful if and only if it occurs in at least one derivation. A variable that is not useful is called useless. A production is useless if it involves any useless variable.

Useless symbols are categorized in two type symbols.

① Non-generating symbol :- The symbol which does not produce any terminal string is known as non-generating symbol.

② Non reachable symbol:- The symbol which can not be reached by starting symbol S are known as non reachable symbol.

Ex① Consider a grammar $G_1 = (V_n = \{A, S, B, C, D, E\}, \Sigma = \{a, b, c\}, P, S)$ with its P:-

$S \rightarrow AB$
$A \rightarrow aA / a$
$B \rightarrow bB / cC$
$C \rightarrow cc$
$D \rightarrow d$
$E \rightarrow a$

Solution- step① After observing the grammar we can see that there is no any non-generating symbol, each non-terminal is producing some terminal.

Step② After observing the CFG we can see that D and E can never be reached. so they are useless and remove from grammar.

so the final grammar will be-

$S \rightarrow AB$
$A \rightarrow aA / a$
$B \rightarrow bB / cC$
$C \rightarrow cc$

Ex② Let a CFG $= (V_n = \{S, A, B, C\}, \Sigma = \{a, b\}, P, S)$

P:-

$S \rightarrow aS / A / C$
$A \rightarrow a$
$B \rightarrow aa$
$C \rightarrow aCb$

Solution- ① Non-generating symbol :- "C"

C is not generating any terminal symbol
so the production -

$C \rightarrow aCb$

$S \rightarrow C$ is removed from grammar

Now the grammar -

$S \rightarrow aS/A$

$A \rightarrow a$

$B \rightarrow aa$

② Non-reachable symbol - "B"

so the production $B \rightarrow aa$ is removed from

Now the grammar -

$S \rightarrow aS/A$

$A \rightarrow a$

② Elimination of ε production-

The production of type $S \to \varepsilon$ are called ε productions. These type of productions can only be removed from those grammars that don't generate ε.

step① First find out all nullable non-terminal variable which derives ε.

step② For each production $A \to a$, construct all production $A \to X$, where X is obtained by removing one or more non-terminal from step ①.

step③ Now combine the result of step ② with the original production & remove ε productions.

Ex① Remove the production from the following CFG by preserving the meaning of it.

$$S \to XYX$$
$$X \to 0X \mid \varepsilon$$
$$Y \to 1Y \mid \varepsilon$$

Sol- To remove ε production from CFG, we need to delete the rule $X \to \varepsilon$ and $Y \to \varepsilon$. So first we will place ε at the RHS whenever X and Y have appeared.

(1) Let us take → $S \rightarrow XYX$

If the first X at RHS is ε then,

$S \rightarrow YX$

If last X at RHS is ε then,

$S \rightarrow XY$

If $Y = \varepsilon$ then

$S \rightarrow XX$

If Y and 1 X are ε then,

$S \rightarrow X$

If both X are replaced by ε

$S \rightarrow Y$

Now $S \rightarrow X|Y|YX|XY|XX|X|Y$

Let us take, $X \rightarrow 0X$

If we place ε in RHS for X then

$X \rightarrow 0$

Similarly - $Y \to 1Y \mid 1$

So the CFG with removed ε production as -

$S \to XY \mid YX \mid XX \mid X \mid Y$
$X \to 0X \mid X$
$Y \to 1Y \mid 1$

Ex② Find a CFG without ε production equivalent to grammar defined by -

$S \to ABaC$
$A \to BC$
$B \to b \mid \varepsilon$
$C \to D \mid \varepsilon$
$D \to d$

Sol- First, find the set of nullable variable -

$B \to \varepsilon$
& $C \to \varepsilon$

So, the production whose RHS contain nullable variable -

$S \to ABaC$
$A \to BC$

Here A is also nullable variable.

So -

$X_n = \{A, B, C\}$ { X_n = set of nullable variable.

Now again the production whose RHS contains nullable variable -

$S \to ABaC$

By putting A, B, C null one by one -

$S \to ABaC$ — (1)
$S \to BaC$ — $\{A \to \varepsilon\}$ — (2)
$S \to AaC$ — $\{B \to \varepsilon\}$ — (3)
$S \to ABa$ — $\{C \to \varepsilon\}$ — (4)
$S \to aC$ — $\{A \to \varepsilon, B \to \varepsilon\}$ — (5)
$S \to Aa$ — $\{B \to \varepsilon, C \to \varepsilon\}$ — (6)
$S \to Ba$ — $\{A \to \varepsilon, C \to \varepsilon\}$ — (7)
$S \to a$ — $\{A \to \varepsilon, B \to \varepsilon, C \to \varepsilon\}$ — (8)

From production $A \to BC$ — (9)
we get -

$A \to B$ — $\{C \to \varepsilon\}$ — (10)
$A \to C$ — $\{B \to \varepsilon\}$ — (11)

The production that not contain any nullable variable are -

$B \to b$ — (12)
$C \to D$ — (13)
$D \to d$ — (14)

Finally, the resultant grammar is from production - (1) to (14)

$S \to ABaC / BaC / AaC / ABa / aC / Aa / Ba / a$
$A \to BC / B / C$
$B \to b$
$C \to D$
$D \to d$

③ Removal of Unit productions :-

The unit productions are the productions in which one non-terminal gives another non-terminal.

OR-

Any production of a CFG of the form-

$A \rightarrow B$

where

$A, B \in V_n$ is called unit Production.

To remove unit productions follow these steps-

step① Construction of unit pairs- If there is a production $A \rightarrow B$ and $B \rightarrow C$. Then make a unit pair $A \rightarrow C$ & apply this chain rule until we get non unit production $A \rightarrow \alpha$.

step② Replace these unit productions by a single production $A \rightarrow \alpha$. Repeat above steps for each unit production pair in given CFG.

step③ Add all non unit productions in the resultant grammar.

Ex① Consider the following CFG-

$G = (V_n = \{S, B, A\}, \Sigma = \{a, b, c\}, P, S)$

where P-

$S \rightarrow Aa \mid B$

$B \rightarrow A \mid bb$

$A \rightarrow a \mid bc \mid B$

Remove all the unit productions and find resultant CFG.

Sol- In the given language, the pair of unit production -

Pair 1:

$S \rightarrow B$

$B \rightarrow A$

$A \rightarrow a \mid bc$

Pair 2:

$S \rightarrow B$

$B \rightarrow A$

$A \rightarrow B$

$B \rightarrow bb$

I For pair 1 -

After removing first unit production

$\boxed{S \rightarrow a \mid bc}$ -①

Remaining pair 1 unit production are -

$B \rightarrow A$

$A \rightarrow a \mid bc$

B is replaced by - $B \rightarrow a \mid bc$ -②

and unit production $B \rightarrow A$ is removed.

II For pair 2-

When we remove unit production then -

$S \to bb$ — (3)

Remaining unit pair 2 is -

$B \to A$

$A \to B$

$B \to bb$

$B \to A$ is replaced by: $B \to bb$ — (4)

and remaining unit production is replaced by -

$A \to bb$ — (5)

Now the production that not contain any unit production are -

$S \to Aa$ — (6)

$A \to a/bc$ — (7)

$B \to bb$ — (8)

Resultant grammar that doesnot contain any unit production are from exp (1) to (8)

$S \to a/bc$

$B \to a/bc$

$S \to bb$

$B \to bb$

$A \to bb$

$S \to Aa$

$A \to a/bc$

$B \to bb$

OR

$S \to a/bc/bb/Aa$

$B \to a/bc/bb$

$A \to bb/a/bc$

Designing Context Free Language from CFG:

For find out CFL from CFG we have to generate a number of strings & then find out a pattern among that string.

Ex① Find the CFL generated by CFG with following production-

$S \rightarrow aSa \mid bSb \mid \varepsilon$

Solution- The given production rules are -

$S \rightarrow \varepsilon$
$S \rightarrow aSa$
$S \rightarrow bSb$

I The first rule $S \rightarrow \varepsilon$, generates single string ε.

II
$S \rightarrow aSa$
$S \rightarrow aa$ $\{S \rightarrow \varepsilon\}$ — ①
$S \rightarrow aSa$
$S \rightarrow aaSaa$ $\{S \rightarrow aSa\}$
$S \rightarrow aaaa$ $\{S \rightarrow \varepsilon\}$ — ②
$S \rightarrow aaSaa$
$S \rightarrow aaaSaaa$ $\{S \rightarrow aSa\}$
$S \rightarrow aaaaaa$ $\{S \rightarrow \varepsilon\}$ — ③

III $S \to bSb$

$S \to bb$ $\{S \to \varepsilon\}$ -④

$S \to bSb$

$S \to bbSbb$ $\{S \to bSb\}$

$S \to bbbb$ $\{S \to \varepsilon\}$ -⑤

$S \to bbSbb$

$S \to bbbSbbb$ $\{S \to bSb\}$

$S \to bbbbbb$ $\{S \to \varepsilon\}$ -⑥

by using same process as above again -

→ $S \to aSa$

$S \to abSba$ $\{S \to bSb\}$

$S \to abba$ $\{S \to \varepsilon\}$ -⑦

$S \to abSba$

$S \to abasaba$ $\{S \to aSa\}$

$S \to ababaa$ $\{S \to \varepsilon\}$ -⑧

→ $S \to bSb$

$S \to basab$ $\{S \to aSa\}$

$S \to baab$ $\{S \to \varepsilon\}$ -⑨

$S \to baSab$

$S \to babSbab$ $\{S \to bSb\}$

$S \to babbab$ $\{S \to \varepsilon\}$ -⑩

Here the production from ① to ⑩ show the some sample string, with the null (ε) string.

$L = \{\varepsilon, aa, bb, aaaa, bbbb, abba, ababa, baab, \ldots\}$

This grammar generates - even length palindrome strings.

Ex ② Find the CFL generated by the CFG with the production -

$S \to aS/bS/a$

Solution - The given production rules are -

$S \to a$
$S \to aS$
$S \to bS$

I $S \to a$ —①

This is shortest and base string

II
$S \to aS$
$S \to aa$ $\{S \to a\}$ —②
$S \to aS$
$S \to aaS$ $\{S \to aS\}$
$S \to aaa$ $\{S \to a\}$ —③
$S \to aaS$
$S \to aaaS$ $\{S \to aS\}$
$S \to aaaa$ $\{S \to a\}$ —④

$S \to aS$
$S \to abS$ $\{S \to bS\}$
$S \to aba$ $\{S \to a\}$ —⑤
$S \to abS$
$S \to abbS$ $\{S \to bS\}$
$S \to abba$ $\{S \to a\}$ —⑥
$S \to abbS$
$S \to abbaS$ $\{S \to aS\}$
$S \to abbaa$ $\{S \to a\}$ —⑦

III

$S \to bS$

$S \to ba$ $\quad \{S \to a\}$ — (8)

$S \to bS$

$S \to bbS$ $\quad \{S \to bS\}$

$S \to bba$ $\quad \{S \to a\}$ — (9)

$S \to bbS$

$S \to bbbS$ $\quad \{S \to bS\}$

$S \to bbba$ $\quad \{S \to a\}$ — (10)

$S \to bS$

$S \to baS$ $\quad \{S \to aS\}$

$S \to babS$ $\quad \{S \to bS\}$

$S \to baba$ $\quad \{S \to a\}$ — (11)

The production from (1) to (11) shows some sample string. so

$L = \{ a, aa, aaa, ba, bba, bba, aba, baa, \ldots \}$

Regular Expression -

$\{a+b\}^* a$

"so, the grammar generates the language end with a."

✳ Designing CFG from CFL :-

Ex① Give the CFG for the set of string that end with 'ab'.

Solution- The minimum string in this language is ab, to accept ab the following production is introduced.

$$S \rightarrow ab$$

All the combinations of a's and b's are derived by the production -

$$A \rightarrow aA \mid bA \mid \varepsilon$$

The derivation that begin with the non-terminal A derives any no. of a's & b's. So the grammar is -

$$S \rightarrow Aab$$
$$A \rightarrow aA \mid bA \mid \varepsilon$$

Ex② Give the CFG for the set of string over $\Sigma = \{0, 1\}$ with equal no. of 1's and 0's.

Sol.- The strings accepted by given language

$\varepsilon \in L$

$01 \in L$

$10 \in L$

The production to generate ε is -

$S \rightarrow \varepsilon$

The production to generate string that begin with 0 and followed by equal no. of 1's is -

$S \rightarrow 0S1$

The production to generate string that begin with 1 and followed by equal no. of 0's is -

$S \rightarrow 1S0$

After combining the above production, we get -

$S \rightarrow \varepsilon / 0S1 / 1S0$

Normal Forms:-

The production of CFG is of the form:-
$A \rightarrow (V_n \cup \Sigma)^*$

There is no restriction on the RHS of the production. The string on RHS can contain any no. of terminals or non terminal in any combination.

If some standard can be developed for the RHS of a production in CFG. Then it will serve the following purposes.-

(1) The complexity of the automaton will be low as it will have to be designed for the standard form only.

(2) The automaton will be easir to implement.

The mandatory requirements for such a standard form is that it should be broad enough so that all the possible productions in CFG should be convertible to this standard form. Such a standard form is known as normal form.

"A normal form puts certain restrictions on the design of production in a CFG and it is possible to convert any available production system of a CFG to normal form."

Note - The start symbol can produce ϵ in CNF & GNF
for ex → $S \to \epsilon$ (it may possible) both in CNF & GNF

Commonly used normal forms in CFG are.
① Chomsky normal form (CNF)
② Greibach normal form (GNF).

① Chomsky Normal Form (CNF):-

In CNF we have a restriction on the length of RHS; which is elements in RHS should be either be two variable or a Terminal

A context free grammar $G=(V_n, \Sigma, P, S)$ is in CNF if all production $\alpha - \beta$ are of the form.

$$A \to a, \text{ or}$$
$$A \to BC$$

where

$A, B, C \in V_n$ &
$a \in \Sigma$

★ Steps to convert a given CFG to CNF -

Step ① If the start symbol S occurs on some right side, create a new start symbol S' and a new production. $S' \to S$

Step ② Remove null productions

Step ③ Remove Unit production

Step④ Replace each production $A \to B_1 B_2 \ldots B_n$ where $n > 2$, with $A \to B_1 C$ where $C \to B_2 \ldots B_n$ Repeat this step for all production having more than two symbols on the right side.

Step⑤ If the right side of any production is in the form $A \to aB$ where 'a' is a terminal and A & B are non-terminals, then the production is replaced by $A \to XB$ and $X \to a$. Repeat this step for every production which is of the form $A \to aB$.

Ex① Convert the CFG with productions -

$S \to ABa$

$A \to aab$

$B \to Ac$ to CNF.

Sol- ① No start symbol occurs at right side

② No null production

③ No unit production

④ In this step we will introduce new variables B_a, B_b, B_c -

$S \to AB B_a$

$A \to B_a B_a B_b$

$B \to A B_c$

$B_a \to a$

$B_b \to b$

$B_c \to c$

⑤ Now find out the productions that has more the two variable in RHS & replace them. and the final result is -

$S \rightarrow A\ D_1$

$D_1 \rightarrow B\ B_a$

$A \rightarrow B_a D_2$

$D_2 \rightarrow B_a B_b$

$B \rightarrow A\ B_c$

$B_a \rightarrow a$

$B_b \rightarrow b$

$B_c \rightarrow c$

Ex ⑥ Convert the following CFG to CNF -

P: $S \rightarrow aAbB$

$A \rightarrow Ab / b$

$B \rightarrow Ba / a$

Sol. - The grammar doesn't have any null production or any unit production.

First we introduce new variables B_a, B_b and we get -

$S \rightarrow B_a A B_b B$

$A \rightarrow A B_b$

$A \rightarrow b$

$B \rightarrow B B_a$

$B \rightarrow a$

$B_a \rightarrow a$

$B_b \rightarrow b$

Now we will replace more than 2 non-terminal from RHS. and we will get final result —

$S \rightarrow D_1 D_2$
$D_1 \rightarrow B_a A$
$D_2 \rightarrow B_b B$
$A \rightarrow A B_b$
$A \rightarrow b$
$B \rightarrow B B_a$
$B \rightarrow a$
$B_a \rightarrow a$
$B_b \rightarrow b$

Ex③ Convert the following CFG to CNF
P: $S \rightarrow ASA \mid aB$
$A \rightarrow B \mid S$
$B \rightarrow b \mid \varepsilon$

Sol.-

Step① — Since S appears in RHS, we add a new state S' and $S' \rightarrow S$ is added -

$S' \rightarrow S$
$S \rightarrow ASA \mid aB$
$A \rightarrow B \mid S$
$B \rightarrow b \mid \varepsilon$

Step② Remove the Null production: $B \rightarrow \varepsilon$ & $A \rightarrow \varepsilon$:
@ After Removing $B \rightarrow \varepsilon$
P: $S' \rightarrow S$, $S \rightarrow ASA \mid aB \mid a$.
$A \rightarrow B \mid S \mid \varepsilon$,
$B \rightarrow b$

(b) After Removing $A \to \varepsilon$

P:

$S' \to S$

$S \to ASA \mid aB \mid a \mid AS \mid SA$

$A \to B \mid S$

$B \to b$

Step (3) Remove the unit productions $S \to S$, $S \to S$, $A \to B$, and $A \to S$:

(a) After removing $S \to S$:

P: $S' \to S$

$S \to ASA \mid aB \mid a \mid AS \mid SA$

$A \to B \mid S$

$B \to b$

(b) After removing $S' \to S$:

P: $S' \to ASA \mid aB \mid a \mid AS \mid SA$

$S \to ASA \mid aB \mid a \mid AS \mid SA$

$A \to B \mid S$

$B \to b$

(c) After removing $A \to B$:

$S' \to ASA \mid aB \mid a \mid AS \mid SA$

$S \to ASA \mid aB \mid a \mid AS \mid SA$

$A \to b \mid S$

$B \to b$

④ After removing $A \to s$:

$S' \to ASA \mid aB \mid a \mid AS \mid SA$
$S \to ASA \mid aB \mid a \mid AS \mid SA$
$A \to b \mid ASA \mid aB \mid a \mid AS \mid SA$
$B \to b$

Step ④ Now find out the productions that has more than two variables in RHS
$S' \to ASA$, $S \to ASA$, $A \to ASA$

After removing these we get -

$S' \to AX \mid aB \mid a \mid AS \mid SA$
$S \to AX \mid aB \mid a \mid AS \mid SA$
$A \to b \mid AX \mid aB \mid a \mid AS \mid SA$
$B \to b$
$X \to SA$

Step ⑤ Now change the productions
$S' \to aB$, $S \to aB$, $A \to aB$
Finally we get -

$S' \to AX \mid YB \mid a \mid AS \mid SA$
$S \to AX \mid YB \mid a \mid AS \mid SA$
$A \to b \mid AX \mid YB \mid a \mid AS \mid SA$
$B \to b$
$X \to SA$
$Y \to a$

✳ Greibach Normal Form (GNF):-

The GNF was proposed by "Sheila Greibach". In GNF we put restrictions not on the length of the RHS of a production, but on the positions in which terminals and variable can appear.

A CFG is in GNF if all the production rules satisfy one of the following conditions-

1. A start symbol generating ε. For ex, $S \to \varepsilon$
2. A non-terminal generating a terminal. For ex, $A \to a$
3. A non-terminal generating a terminal which is followed by any number of non-terminals. For ex, $S \to aABC$

Steps for converting CFG into GNF:-

step① Convert the grammar into CNF.

step② Apply left factoring to the production set

step③ If the grammar exists left recursion, eliminate it.

step④ In the grammar, convert the given production rule into GNF form.

Ex① Convert the following CFG to GNF

$S \to CA \mid BB$

$B \to b \mid SB$

$C \to b$

$A \to a$

Sol. - step① - The given grammar is already in CNF

step② Change the names of the Non-terminal symbols into some A_i in ascending order of i.

For the given grammar :- S with A_1,
C with A_2,
A with A_3,
B with A_4

we get -

$A_1 \to A_2A_3 \mid A_4A_4$

$A_4 \to b \mid A_1A_4$

$A_2 \to b$

$A_3 \to a$

step③ Alter the rules so that the Non-terminals are in ascending order, such that, if the production is of the form $A_i \to A_jx$, then $i < j$ and should never be $i >= j$

In the grammar (we get in step②) the production rule $A_4 \to b \mid A_1A_4$ doesn't follow this condition.

Here, A_4 is A_i and A_1 is A_j

so. $(4 > 1)$ then

$i > j$

so we need to resolve this issue -

$A_4 \rightarrow b \mid A_1 A_4$

$A_4 \rightarrow b \mid A_2 A_3 A_4 \mid A_4 A_4 A_4$ { Replace A_1 with other values.

Again A_4 is A_i and A_2 is A_j

$4 > 2$ then $i > j$

so -

$A_4 \rightarrow b \mid b A_3 A_4 \mid A_4 A_4 A_4$ { Replace A_2 with other values.

step (4) Remove left Recursion -

Introduce a new variable to remove the left recursion -

$A_4 \rightarrow b \mid b A_3 A_4 \mid A_4 A_4 A_4$

$Z \rightarrow A_4 A_4 Z \mid A_4 A_4$

$A_4 \rightarrow b \mid b A_3 A_4 \mid b Z \mid b A_3 A_4 Z$

Now the grammar is -

$A_1 \rightarrow A_2 A_3 \mid A_4 A_4$

$A_4 \rightarrow b \mid b A_3 A_4 \mid b Z \mid b A_3 A_4 Z$

$Z \rightarrow A_4 A_4 \mid A_4 A_4 Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Step ⑤ Check whether the new grammar is in GNF or not - and if not then convert it using GNF rules -

$A_1 \to b A_3 \mid b A_4 \mid b A_3 A_4 A_4 \mid b Z A_4 \mid b A_3 A_4 Z A_4$
$A_4 \to b \mid b A_3 A_4 \mid b Z \mid b A_3 A_4 Z$
$Z \to b A_4 \mid b A_3 A_4 A_4 \mid b Z A_4 \mid b A_3 A_4 Z A_4 \mid$
$b A_4 Z \mid b A_3 A_4 A_4 Z \mid b Z A_4 Z \mid b A_3 A_4 Z A_4 Z$
$A_2 \to b$
$A_3 \to a$

Ex ② Convert the following CFG to GNF -

$S \to AB$
$A \to BS \mid a$
$B \to SA \mid b$

Sol:- The given CFG is in CNF.

$S \to AB$
$A \to BS \mid a$
$B \to SA \mid b$

Rename all the variables (non-terminal symbols) into some A_i in ascending order

So, replace

S	S with A_1
	A with A_2
	B with A_3

we get -

$A_1 \rightarrow A_2 A_3$

$A_2 \rightarrow A_3 A_1 / a$

$A_3 \rightarrow A_1 A_2 / b$

Let all the productions are in $A_i \rightarrow A_j X$ form. Then check whether $i < j$ and should never be $i >= j$.

The production rule $A_3 \rightarrow A_1 A_2$ doesn't follow the above rule.

Here A_3 is A_i and A_1 is A_j

So $(3 > 1)$ then $i > j$

So we need to resolve this issue -

$A_3 \rightarrow A_1 A_2 / b$

$A_3 \rightarrow A_2 A_3 A_2 / b$ $\{A_1 \rightarrow A_2 A_3\}$

again $i > j$ -

$A_3 \rightarrow A_3 A_1 A_3 A_2 / a A_3 A_2 / b$ $\{A_2 \rightarrow A_3 A_1 / a\}$

here $i = j$, Remove left recursion

Left Recursion: If a grammar is in form of

$A \rightarrow A\alpha / \beta$

To remove left recursion from the grammar we can rewrite the grammar in form -

$$A \rightarrow \beta A'$$
$$A' \rightarrow \alpha A' / \epsilon$$

To remove ϵ from here -
After removing ϵ we get -

$$A \rightarrow \beta A' / \beta$$
$$A' \rightarrow \alpha A' / \alpha$$

After removing left recursion from

$A_3 \rightarrow A_3 A_1 \; A_3 A_2 / a \; A_3 A_2 / b$

we get -

$$A_3 \rightarrow a A_3 A_2 A_2 A' / bA' / a A_3 A_2 / b$$
$$A' \rightarrow A_1 A_3 A_2 A' / A_1 A_3 A_2$$

So the Grammar is -

$$A_1 \rightarrow A_2 A_3$$
$$A_2 \rightarrow A_3 A_1 / a$$
$$A_3 \rightarrow a A_3 A_2 A' / bA' / a A_3 A_2 / b$$
$$A' \rightarrow A_1 A_3 A_2 A' / A_1 A_3 A_2$$

Convert the new grammar using GNF rules -

$A_2 \rightarrow A_3 A_1$
$A_2 \rightarrow aA_3A_2A'A_1 / bA'A_1 / aA_3A_2A_1 / bA_1$

$A_1 \rightarrow A_2 A_3$
$A_1 \rightarrow aA_3A_2A'A_1A_3 / bA'A_1A_3 / aA_3A_2A_1A_3 / bA_1A_3$

$A' \rightarrow A_1A_3A_2A' / A_1A_3A_2$
$A' \rightarrow aA_3A_2A'A_1A_3A_3A_2A' / bA'A_1A_3A_3A_2A' / aA_3A_2A_1A_3A_3A_2A' / aA_3A_2A'A_1A_3A_3A_2 / bA'A_1A_3A_3A_2 / aA_3A_2A_1A_3A_3A_2 / bA_1A_3A_3A_2$

So now the grammar is -

$A_1 \rightarrow aA_3A_2A'A_1A_3 / bA'A_1A_3 / aA_3A_2A_1A_3 / bA_1A_3$
$A_2 \rightarrow aA_3A_2A'A_1 / bA'A_1 / aA_3A_2A_1 / bA_1$
$A_3 \rightarrow aA_3A_2A' / bA' / aA_3A_2 / b$
$A' \rightarrow aA_3A_2A'A_1A_3A_3A_2A' / bA'A_1A_3A_3A_2A' / aA_3A_2A_1A_3A_3A_2A' / bA_1A_3A_2A' / aA_3A_2A'A_1A_3A_3A_2 / bA'A_1A_3A_3A_2 / aA_3A_2A_1A_3A_3A_2 / bA_1A_3A_3A_2$