Unit - III

GoodLuck | Page No.
Date

Grammers

- **Grammer :-** A grammer is a set of production rules which are used to generate Strings of a language

or

Set of rule to define any language by which meanigful communication link will be established.

- **Language :-** Set of Strings generated by any grammer is known a language.

or

It is a alpha numeric algebric & Syntatical representation by which we can initialize a communication link b/w two machine, two human, & machine to human.

Ex:- General language Hindi, English etc.
programming language c, c++, Java etc.

- Grammer generate language & language accepted by finite automata.

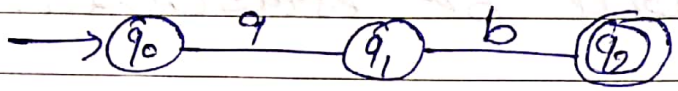
> Grammer is a language generator finite automata is language acceptor.

Example:-

$S \to AB$	$S \to AB$
$A \to a$	$S \to aB$ $[A \to a]$
$B \to b$	$S \to ab$ $[B \to b]$
grammer	Derivation

So this grammer generate language

$$L = \{ab\}$$



NFA

Formal definition of grammer:-

A grammer is a 4-tuple such that

$$G = \{V, T, P, S\}$$

where

V = non-empty set Variables or non-terminal. It is represented by capital letters.

Ex:- $V = \{A, B, C, D \cdots\}$

T = non-empty set of terminal represented by small letter, no. or symbolic form.

Ex:- $T = \{a, b, c \cdots 0, 1, 2, \cdots *, \# \cdots\}$

GoodLuck | Page No.
Date

P = non-empty set of production

S = Starting production Variable.

Ex:- S → aB | bA
A → ABB | as | a
B → bAA | bS | b

V = {S, A, B}

T = {a, b,}

P = S → aB | bA
A → ABB | as | a
B → bAA | bS | b

S = S

★ Types of grammer :-

4 types of grammes

i] Type(0) grammes :-

It is also known as unrestricted or phares Structured grammes (UGs / PSGs).

- It is accepted by turing machine.
- Any grammes is type(0) if it contain production.

$$\alpha \rightarrow \beta$$

GoodLuck | Page No.
Date

In following 3 condition.

1) $\alpha \in (V+T)^+$
2) $\beta \in (V+T)^*$
3) $|\alpha| \leq |\beta|$
4) α Contain atleast one NT.

Example: 1] $A \to aB / \varepsilon$ ✓ $aAb \to bB$
$A \to aB$ $aA \to \varepsilon$
$A \to \varepsilon$

no ε on α (L.H.S) ε may on β (R.H.S)

~~2) $aAb \to abd / bC \to$ not type(0).~~

ii] Type (1) :- It is also known as Content sensitive grammer (CSG).

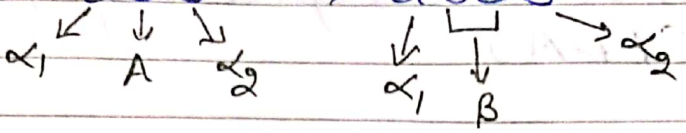
- It is accepted by LBA (Linear bounded automata).
- Any grammer is type(1) if it Contain a production for

$$\alpha_1 A \alpha_2 \longrightarrow \alpha_1 B \alpha_2$$
$$\alpha \longrightarrow \beta$$

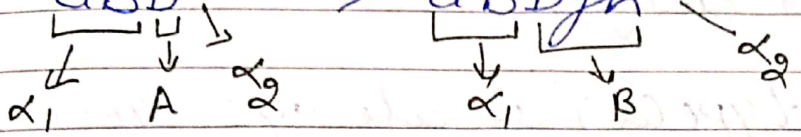
in Following 4 Condition.

1) $\alpha_1, \alpha_2 \in (V+T)^*$
2) $B \in (V+T)^+$
3) $A \in V$
4) $|LHS| \leq |RHS|$
$|\alpha| \leq |\beta|$ (length of α) $\leq$ (length of β)

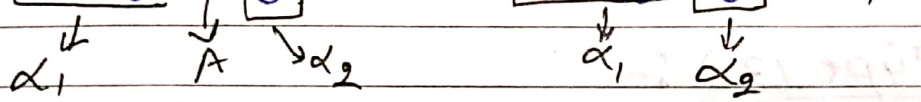
Example 1) aSb → aBCb ✓



2) aBD → aBDgh ✓



3) bdAgHbg → bdAgbg ✗



- Every type(1) grammer is always type(0) but reverse is not always true.

iii] **Type (2) :-** It is also known as Content free grammer.

- Any grammer is CFG if it contain production.

$$A \rightarrow B$$

1) $A \in V$
2) $B \in (V+T)^*$

V = Variable or non-terminal.

- If one non-terminal produce any value (terminal or non-terminal) is known as CFG.

Ex: $E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid id.$

- CFG is accepted by push down automata (PDA).
- CFG is used in parser construction in Compiler design.
- Every type(2) is always type(0) but reverse is not always true. & type(1)

iv) Type (3) :-

- It is known as regular grammer.
- It is accepted by finite automata with the help of regular expression.
- This grammer is used in lexical analysis phase of Compiles design in form of regular expression.
- Every type(3) grammer is always type(0) and type(2) but reverse is not always true.

$A \to Ba$ (left linear)
$A \to aB$ (Right linear)

$A \to \epsilon$ also Regular grammer.
$A \to ba$

- There is no middle Linear $S \to aSb$ X

GoodLuck | Page No.
Date

Context Sensitive grammer
or
Type1 grammer
or
Length increasing grammer

Content Sensitive grammer has production of the form.

$$\alpha A \beta \longrightarrow \alpha \gamma \beta$$

(α: Left Context, β: Right Context)

Where $A \in V$ (Variable / Non-terminal)
(atleast single non-terminal)

$\alpha, \beta \in (V+T)^*$

Where γ is non-empty (ϵ is not allowed)

$$\gamma \in (V+T)^+$$

Length of L.H.S $\leq$ Length of R.H.S

OR

Content Sensitive grammer has production of the form.

$$\alpha \rightarrow \beta$$

$\alpha \in (V+T)^* V (V+T)^*$ (Atleast single Variable on L.H.S)

$\beta \in (V+T)^+$

GoodLuck | Page No.
Date

$|\alpha| \leq |\beta|$ $A \to \varepsilon$ not allowed

$aAb \to bb$ not allowed.

$aAb \to bbb$ ✓

$aAb \to aaaa$ ✓

~~Example~~:

Exception:- $S \to \varepsilon$

S should be Start Symbol but S should not apear in R.H.S of production.

$S \to aSb$] not allowed | $S \to AB|\varepsilon$ ✓

$S \to \varepsilon$]

- Context Sensitive grammer generates Context Sensitive Language.
- Context Sensitive language is accepted by Linear bounded automata (LBA).

Example:- $\alpha \to \beta$ $|\alpha| \leq |\beta|$

Production rules ←

$S \to aSBC$

$S \to abc$

$CB \to BC$

$bB \to bb$

$G = (V, T, PS)$

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

GoodLuck | Page No.
Date

Derivation:

$S \rightarrow aSBc$
$S \rightarrow aabcBc$ [$S \rightarrow abc$]
$S \rightarrow aabBcc$ [$cB \rightarrow Bc$]
$S \rightarrow aabbcc$ [$bB \rightarrow bb$]

So language generated by grammer is

$$L(G) = \{a^n b^n c^n / n \geq 1\}$$

Content-Free grammer :-

or

Type 2 grammer

Content aAb → Right Context of A.
↓ Left Context of A.

Context-free $\in A \in$ or A nonterminal A is free of Context Symbol.

Definition :-

If in a grammer, the production rule are of the form

$$\alpha \rightarrow \beta$$

Where

$\alpha \in V$ (Single Variable or non-terminal without Context)

$\beta \in (V+T)^*$ that means β can be ϵ.

Example. $S \to sAS$ ✓

$S \to \epsilon$ ✓

$aS \to bBX$ ✗

then the grammer is called a context free grammer

- The language generated by context free grammer is called a context free language (CFL) which is accepted by pushdown automata (PDA).

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

Pare. $S \to aSb$

$S \to ab$

$S = \{S\}$

$V = \{S\}$

$T = \{a, b\}$

Derivation:- $S \to aSb$

$\to aabb$ [$S \to ab$]

$L = \{aabb, aaabbb \text{----}\}$

$S \to aSb$

$\to aaSbb$ [$S \to aSb$]

$\to aaabbb$ [$S \to ab$]

$L(a) = \{a^n b^n \geq 1\}$

Example:-

$E \rightarrow E+T / T$
$T \rightarrow T*F / F$
$F \rightarrow (E) / a$

$\Big] \rightarrow P$

$G = (V, T, P, S)$

$V = \{E, T, F\}$

$T = \{+, *, (,), a\}$

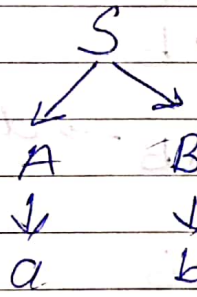
$E \rightarrow$ Start Symbol $\quad S = \{E\}$

☆ Derivation tree :-

The process of deriving a string is called as derivation.

or

Step by step derivation process can be expressed by a tree structure which is known as derivation tree.



Parse tree / derivation tree :-

It is the geometrical representation of a derivation.

- A derivation tree is the tree representation of a CFG.

GoodLuck | Page No.
Date

- In derivation tree, nodes, are labled with the left side of production (of CFG) & the children of a node represent its corresponding right side of productions

- Types of derivation:

Derivation

Left most derivation (LMD)

Right most derivation (RMD).

1) Left most derivation (LMD):- whenever we expand left most non-terminal at derived side of grammer than it is LMD.

Ex:- $S \to AB$
$A \to a$
$B \to b$

Solve:

$S \to AB$ → derived side.

derivative ←

$S \to aB$
$S \to ab$.

2) Right most derivation (RMD):- whenever we expand right most non-terminal at derived side of grammer to generate any string then it is RMD.

GoodLuck | Page No.
Date

Ex:- $S \to AB$
$S \to Ab$
$S \to ab$

Example:- CFG

$S \to AB$
$A \to 0A \mid 0$
$B \to 1B \mid 1$

Let as Consider a String $w = 00111$

LMD:

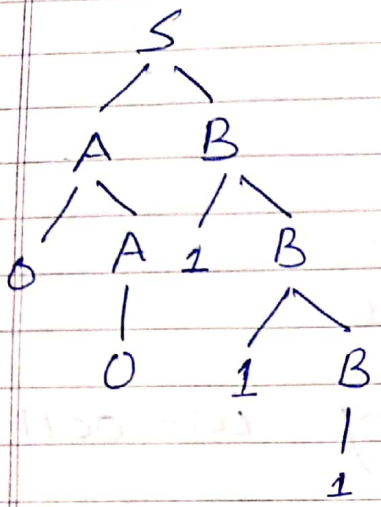
$S \to AB$
$S \to 0AB$ (using $A \to 0A$)
$S \to 00B$ (Using $A \to 0$)
$S \to 001B$ (Using $B \to 1B$)
$S \to 0011B$ (Using $B \to 1B$)
$S \to 00111$ (Using $B \to 1$)

RMD

$S \to AB$
$S \to A1B$ (Using $B \to 1B$)
$S \to A11B$ (Using $B \to 1B$)
$S \to A111$ (Using $B \to 1$)
$S \to 0A111$ (Using $A \to 0A$)
$S \to 00111$ (Using $A \to 0$)

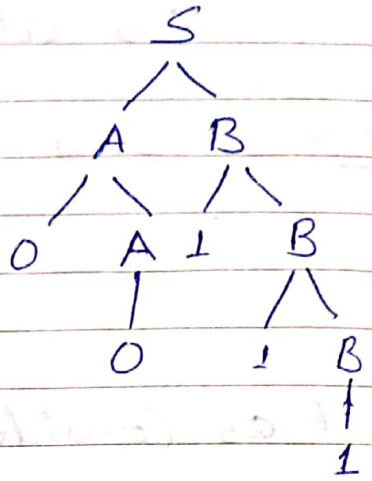
- Acceptibility:- Any String will be accepted by any grammer if starting non-terminal of grammer will produce the String exactly.

LMD tree:



(used by top-down parser)

RMD tree:



(Bottom-up parser uses of reverse RMD)

Note For unambiguous grammer, Leftmost & rightmost derivation represent the same parse tree.

hence above given grammer is unambiguous, hence both parse trees are same

Q $E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

1) (id+id) * id

2) id + id * id.

GoodLuck | Page No.
Date

* **Ambiguity :-** If any string will be accepted by any grammer in more than one ways then sometimes it is possible to get more than one different derivation tree for the single string by any grammer than such grammer are ambiguous grammer & the property in ambiguity.

- For ambiguous grammer, LMD & RMD derivation represent different parse trees.

Example: Consider the following grammer.

$E \rightarrow E+E \mid E*E \mid id.$

Let input string w is id + id * id

Solve: LMD

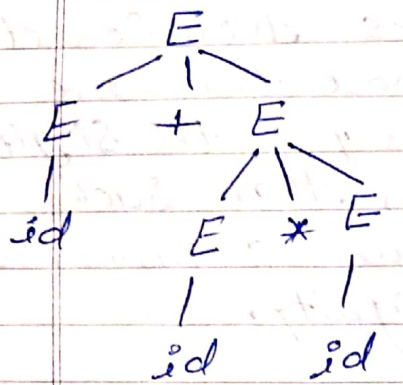
$E \rightarrow E+E$
$E \rightarrow id+E$ (using $E \rightarrow id$)
$E \rightarrow id+E*E$ (using $E \rightarrow E*E$)
$E \rightarrow id+id*E$ (using $E \rightarrow id$)
$E \rightarrow id+id*id$ (using $E \rightarrow id$)

RMD

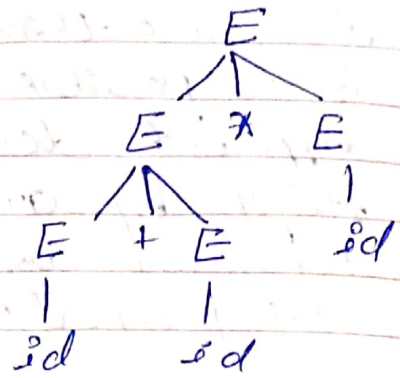
$E \rightarrow E*E$
$E \rightarrow E*id$ (using $E \rightarrow id$)
$E \rightarrow E+E*id$ (using $E \rightarrow E+E$)
$E \rightarrow E+id*id$ (using $E \rightarrow id$)
$E \rightarrow id+id*id$ (using $E \rightarrow id$)

Derivation tree:-

LMD



RMD



Parse tree1 for String id+id*id

Parse tree 2 for String id+id*id

- Since given grammer is ambiguous, hence there are two different parse trees exist for given input string.

✱ Simplification of Context -free grammer:-

- CFG may consist of some extra symbols or non-terminal.

presence of extra symbols unnecessary increase the length of grammer.

- Simplification of CFG means reduction of CFG by removing useless symbols.
- To simplify the grammer we need to eliminate

1) Left Recursion
2) Left factor
3) Removal of useless production.

GoodLuck | Page No.
Date

4] Removal of unreachable production
5] Removal of null production or ε-production
6] Removal of unit production.

1] Left Recursion :- Whenever any non-terminal repeat itself at left most position of grammer production than it is left recursion.

Technically format:

$$A \rightarrow A\alpha_1 + A\alpha_2 + A\alpha_3 \cdots A\alpha_4 \mid \beta_1 \mid \beta_2 \cdots$$

where A is left recursion non-terminal
$\alpha_1, \alpha_2, \alpha_3 \cdots \alpha_n$ Can Contain any Value.
$\beta_1, \beta_2 \cdots \beta_n$ Can Contain any value but it doesn't Contain left recursion non-terminal at left most side.

Ex:- $E \rightarrow E+T \mid E*F \mid E/F \mid a \mid b \mid a \mid g$

(α_1 = +T, α_2 = *F, α_3 = /F; β_1 = a, β_2 = b, β_3 = a, β_4 = g)

Removal of left recursion :-

$$A \rightarrow A\alpha \mid \beta$$

(A → V, α → (VUT)*)

Removal

$$A \rightarrow \beta B$$
$$B \rightarrow \alpha B \mid \varepsilon$$

Q:- $E \rightarrow E+T \mid T$

S:- $E \rightarrow TB$
$B \rightarrow +TB \mid \varepsilon$

GoodLuck | Page No.
Date

2) **Left factor** :- When any value repeat itself more than one time at left most position of any grammes production & it construct a left factor of it.

Ex: $S \to aBd / aBgh / aBFD / bg / gh / ij / \varepsilon$

(B_1, B_2, B_3, γ_1, γ_2, γ_3, γ_4)

Where A is any non-terminal α is combine or single value responsible for left factor $\beta_1, \beta_2, \beta_3 \dots \beta_n$ can contain any value. $\gamma_1, \gamma_2, \gamma_3 \dots \gamma_n$ can contain any value but it does not contain left factor combination at left most side.

$$A \to \alpha\beta_1 / \alpha\beta_2 / \alpha\beta_3 \dots \alpha\beta_n / \gamma_1 / \gamma_2 \dots \gamma_n$$

Ex:- $A \to aBD / aD / ag / b$

$A \to aE / b$

$E \to BD / D / g$

Q. $E \to abE / abBh / Bhg / d$.

$E \to abA / Bhg / d$

$A \to E / Bh$.

3) Removal of useless production :-

A production which contain useless non-terminal is known as useless production.

Useless non-terminal :- A non-terminal which never terminate completely anywhere in the grammer is useless non-terminal.

Ex. $S \to AB \mid AC$
$A \to a$
$B \to BC \mid BA$
$C \to b$

Sol^n :

Useful N.T. = {A, C}

$A \to a$] → direct termination
$C \to B$]

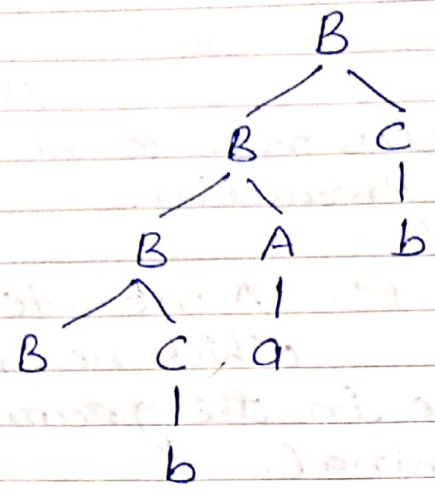
Indirect

$S \to AC$
$S \to ab$

Useful N.T. = {A, C} ∪ {S}

= {A, C, S}

Useless N.T = {B}



$S \rightarrow AC$
$A \rightarrow a$
$B \rightarrow b$

This simplified grammer.

Q. $S \rightarrow AB/a$
$A \rightarrow Bc/b$
$B \rightarrow aB/c$
$C \rightarrow ac/B$

[B, C useless]

Q. $S \rightarrow AB/AC$
$A \rightarrow aAb/bAa/a$
$B \rightarrow bbA/aaB/AB$
$C \rightarrow abCA/aDb$
$D \rightarrow bD/aC$

Q $S \rightarrow AB/a$
$A \rightarrow aA$
$B \rightarrow b$

GoodLuck | Page No.
Date

4] Removal of null production
or
Removal of ϵ production

The production of the form

Non terminal $\rightarrow \epsilon$
or non terminal $\rightarrow \wedge$

is said to be null production.

Elimination of null production:- Delete all the null productions & add new productions by substituting ϵ in the old productions

Ex:
$S \rightarrow AB$
$A \rightarrow aA / \epsilon$
$B \rightarrow b$

Sol:
$S \rightarrow AB / B$
$A \rightarrow aA / a$
$B \rightarrow b$

Q.
$S \rightarrow ABC / aABb$
$A \rightarrow aAB / bA / \epsilon$
$B \rightarrow bBAC / cB / \epsilon$
$C \rightarrow cC / \epsilon$

Sol:
Step-1
Remove ϵ for C

$S \rightarrow ABC/aABb/AB$
$A \rightarrow aAB/bA/\varepsilon$
$B \rightarrow bBAC/cB/\varepsilon/bBA$
$C \rightarrow cC/c$

Step II:- Remove ε of B

$S \rightarrow ABC/aABb/AB/AC/aAb/A$
$A \rightarrow aAB/bA/\varepsilon/aA$
$B \rightarrow bBAC/cB/bBA/bAC/c/bA.$

Step III:-
Remove ε of A.

$S \rightarrow ABC/aABb/AB/AC/aAb/A/BC/aBb/Bb/$
ab/ε

$A \rightarrow aAB/bA/aA/aB/b/a$
$B \rightarrow bBAC/cB/bBA/bAC/c/bA/bBc/Bb/$
$bc/b.$

$C \rightarrow cC/c.$

5) Removal of Unit production:-

If A & B are non-terminals than any production of type

$A \rightarrow B$

is called as unit production.

8 $E \to E+T/T$
$T \to T*F/F$
$F \to (E)/id$

Sol:- Step1 = Check for unit derivability.

$\{E, T\}$ and $\{T, F\}$

Removal of $\{E, T\}$

$E \to E+T/T*F/F$
$T \to T*F/F$
$F \to (E)/id$

Step II:- Check for unit derivability.

$\{E, F\}$ and $\{T, F\}$

Removal of E, F

$E \to E+T/T*F/(E)/id$
$T \to T*F/F$
$F \to (E)/id$

Step III:- Check for unit derivability

$\{T, F\}$

Removal of T, F

$E \to E+T/T*F/(E)/id$
$T \to T*F/(E)/id$
$F \to (E)/id$

Step IV: Check for unit derivability

$\{\emptyset\}$

The grammer is unit production free.

★ ε - production Example:-

given CFG

$S \rightarrow aSaSb \mid \varepsilon$

Sol:- Eliminate of $S \rightarrow \varepsilon$ null production

$S \rightarrow aSaSb$
$S \rightarrow a\varepsilon aSb$ ⟹ Step I
$S \rightarrow aaSb$

$S \rightarrow aSaSb$
$S \rightarrow aSa\varepsilon b$ ⟹ Step II
$S \rightarrow aSab$

$S \rightarrow aSaSb$ ⟹ Step III
$S \rightarrow a\varepsilon a\varepsilon b$
$S \rightarrow aab$

Eliminate $S \rightarrow \varepsilon$ production from given CFG & Add $S \rightarrow aaSb$, $S \rightarrow aSab$, $S \rightarrow aab$

So the new CFG without ε is

$S \rightarrow aSaSb \mid aaSb \mid aSab \mid abb$.

Ans

GoodLuck | Page No.
Date

✱ Regular Grammes or Type 3 grammes

Regular grammes
||
⇓ Right Linear grammes
⇓ Left Linear grammes

1) **Right Linear grammes:-** A grammes $G = \{V, T, P, S\}$ is said to be right linear if all productions are of the form

$$A \to aB, \quad A \to a$$

where $A, B \in V$ & $a \in T^*$

V = finite set of non-terminal
T = finite set of terminal.
S = Starting production of (Symbol)
P = finite set of productions

2) **Left Linear grammes:-** A grammes $G = \{V, T, P, S\}$ is said to left linear grammes if all production are of the form.

$$A \to Ba, \quad A \to a$$

Where $A, B \in V$, $a \in T^*$

Example:- Right Linear grammer.

production P: $S \rightarrow abS/a$

S → Starting Symbol
a, b → terminal.

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

Left Linear grammer:-

P: $S \rightarrow Aab$
$A \rightarrow Aab/B$
$B \rightarrow b$

S → Starting Symbol
a, b → terminal
S, A, B → non terminal.

Note:- Only Single non-terminal allowed in L.H.S of any production & ϵ(1) or more terminals are allowed in R.H.S of any production.

$A \rightarrow abC$ ✓
$A \rightarrow aB/a$ ✓
$A \rightarrow \epsilon$ ✓
$A \rightarrow Ba$ ✓

$A \rightarrow aBb$ ✗
$A \rightarrow BC$ ✗

A grammer is said to be regular if it is either left linear or right linear.

Right Linear grammer	Left Linear grammer
$A \to aB/a$	$A \to Ba/a$
$B \to aB/bB/a/b$	$B \to Ba/Bb/a/b$

Q. Grammer $G = (\{S, AB\}, \{a, b\}, S, P)$ with production

$S \to A$
$A \to aB/\varepsilon$
$B \to Ab$

It is a regular language?

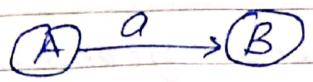
Sol:- • It is not a regular ~~grammer~~ language because it is neither right linear nor left linear.

• No mixing of left linear & right linear allowed.

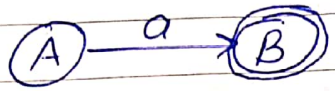
• Regular grammer generates regular language which is accepted by finite automata.

GoodLuck | Page No.
Date

★ Conversion of finite automata to regular grammer:-

Rule 1:- FA 

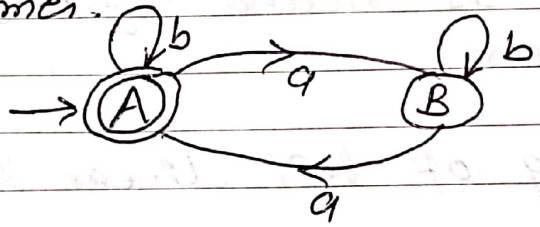
Production $A \rightarrow aB$

Rule 2:- 

Production $A \rightarrow a$
$A \rightarrow aB$

Rule 3:- If initial state is final state than add $\wedge (\varepsilon)$ in a production

Q Convert the following DFA to regular grammer.



Sol:- Regular grammer

$$G = (\{A, B\}, \{a, b, \varepsilon\}, P, A)$$
$$V \qquad T \qquad P \quad S$$

when P:

$A \rightarrow aB \mid bA \mid b \mid \varepsilon$
$B \rightarrow aA \mid bB \mid a.$

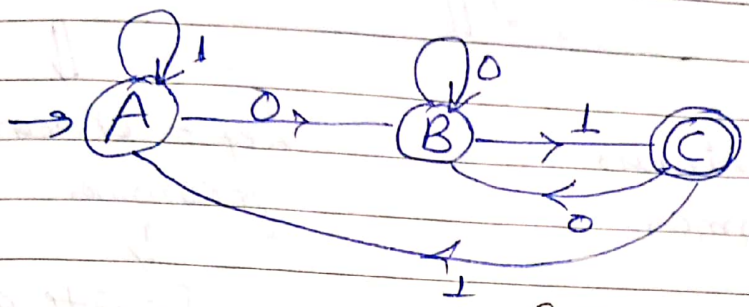
OR

Q. Write a grammer which generate strings containing even number of a's.

GoodLuck | Page No.
Date

Q:- Write a grammer which generate strings end with 01.

Sol:- First design DFA which accepts string end with 01



(DFA)

Grammer for this DFA is

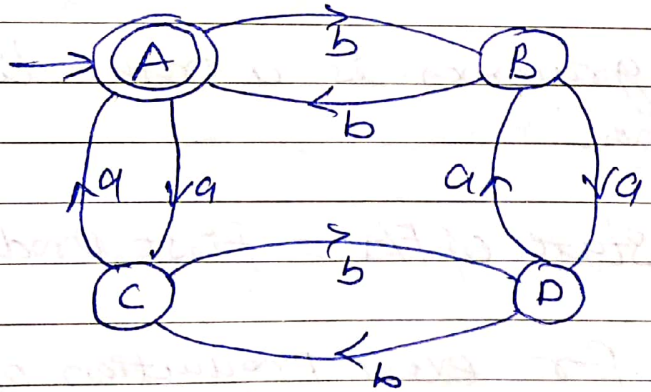
$G = (\{A, B, C\}, \{0, 1\}, P, A)$

Where P: $A \to 0B / 1A$
$B \to 0B / 1C / 1$
$C \to 0B / 1A$.

Right Linear

Q. Write a grammer which generate string Containing even no. of a, & even no of b.

Sol:-



Grammer $G = (\{A, B, C, D\}, \{a, b\}, P, A)$

Where P: $A \to aC / bB / \varepsilon$
$B \to bA / aD / b$
$C \to aA / bD / a$
$D \to aB / bC$

Right Linear

Date

☆ Conversion of regular grammar (type 3) to automata machine (finite automata) :-

Regular grammar

method I → Right Linear grammar → finite automata (simple).

Left Linear grammar → finite automata (Method II)

method I ⇒ Conversion of Right Linear regular grammar to finite automata.

Q Convert the following grammar to finite automata.

$A \to aB \mid bA \mid b$
$B \to aC \mid bB$
$C \to aA \mid bC \mid a.$

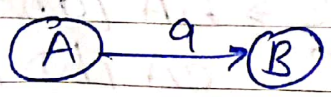
Sol:- Given grammar is a right linear regular grammar.

Start State of FA = first production's state

Step 1:- for every production of type

$A \to aB$

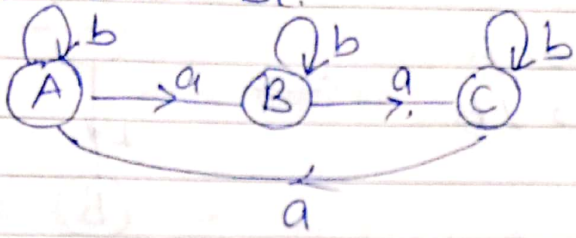
FA.



In given grammer, following production are of above type.

$A \to aB$ $\quad A \to bA$
$B \to aC$ $\quad B \to bB$
$C \to aA$ $\quad C \to bC$

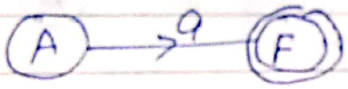
~~for this AA~~
So FA will be.



Step II:- for every production of type.

$A \to a$

FA

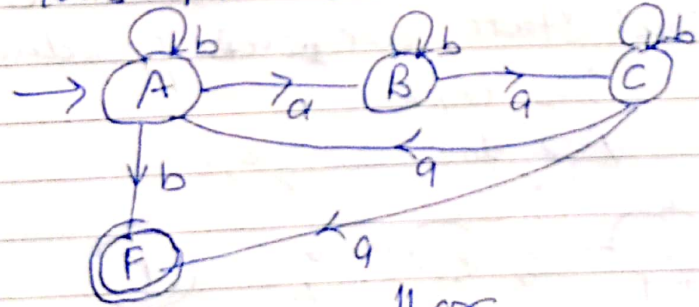


finall state

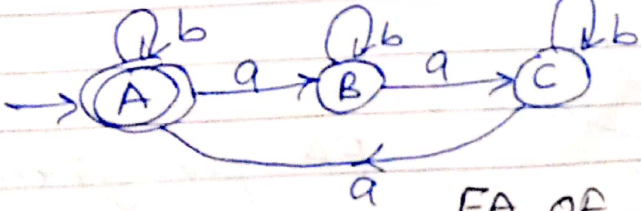
In given grammer, we have

$A \to b$ & $C \to a$

for this FA will be come



⇓ or



FA of given grammer.

Q2:- Given grammer is

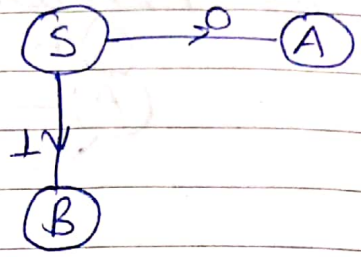
$S \to 0A \mid 1B \mid 0 \mid 1$
$A \to 0S \mid 1B \mid 1$
$B \to 0A \mid 1S$

Right Linear grammer

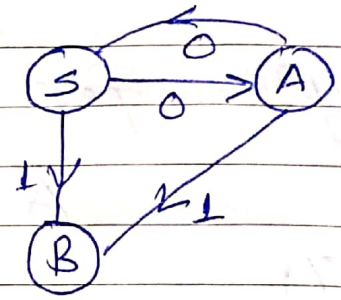
Convert it into FA.

Sol:- In given grammer. FA.

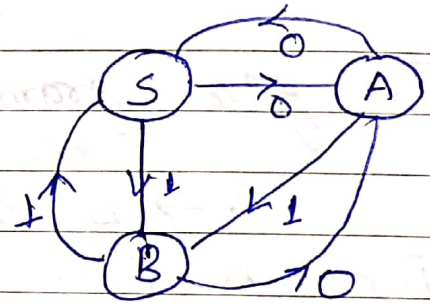
$S \to 0A$
$S \to 1B$ $\Longrightarrow$



$A \to 0S$
$A \to 1B$ $\Longrightarrow$

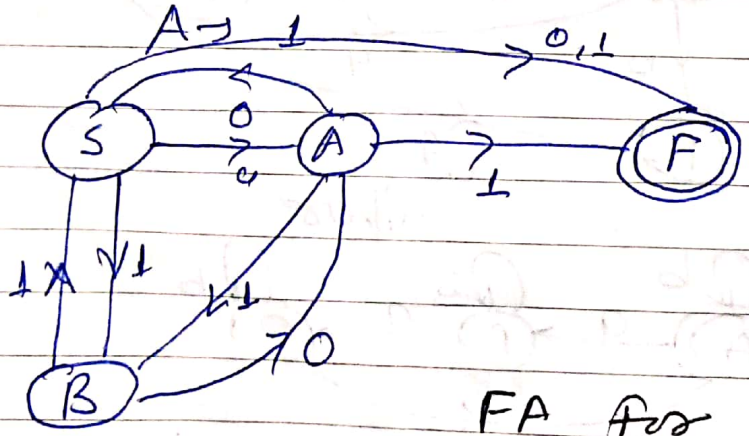


$B \to 0A$
$B \to 1S$ $\Longrightarrow$



for final state, we ~~state~~ productions.

$S \to 0 \mid 1$
$A \to 1$



FA for given grammer

GoodLuck | Page No.
Date

☆ Conversion of left linear regular grammer to finite automata:-

LLG ⇒ FA.

LLG (L) —Reverse (1)→ RLG (LR) —(2)→ FA (LR) —Reverse (3)→ FA (LR)R, L

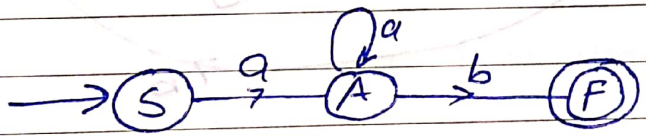
Example:- Left Linear grammer

$S \rightarrow Aa$
$A \rightarrow Aa/b$

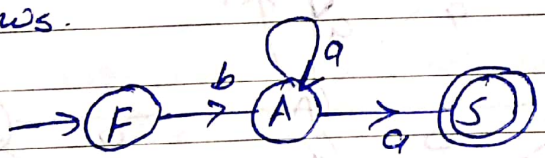
Step 1:- Reverse given grammer to right linear grammer

$S \rightarrow aA$
$A \rightarrow aA/b$ Right Linear grammer.

Step 2:- Construct FA corresponding to Right Linear grammer.



Step 3:- Reverse finite Automata change final state to initial state & initial state to final. also change direction of arrows.



finite automata of given Left linear grammer.

GoodLuck | Page No. | Date

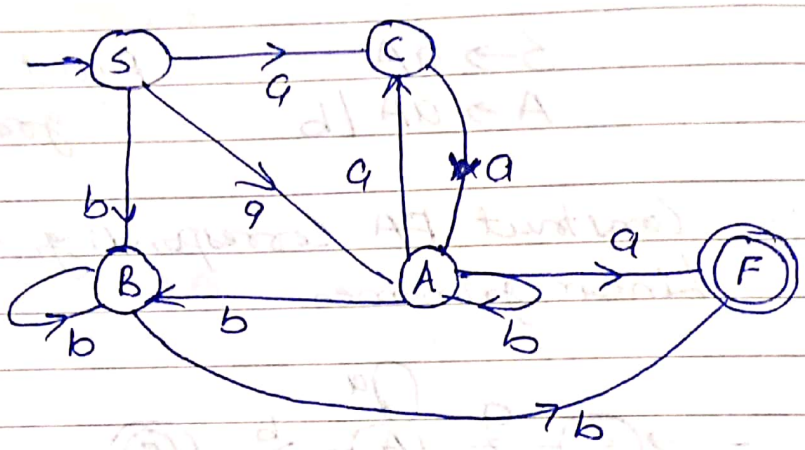
Q

S → Ca / Aa / Bb
A → Ab / Ca / Bb / a
B → Bb / b
C → Aa

Sol:- Step I:- LLG $\xrightarrow{\text{Reverse}}$ RLG

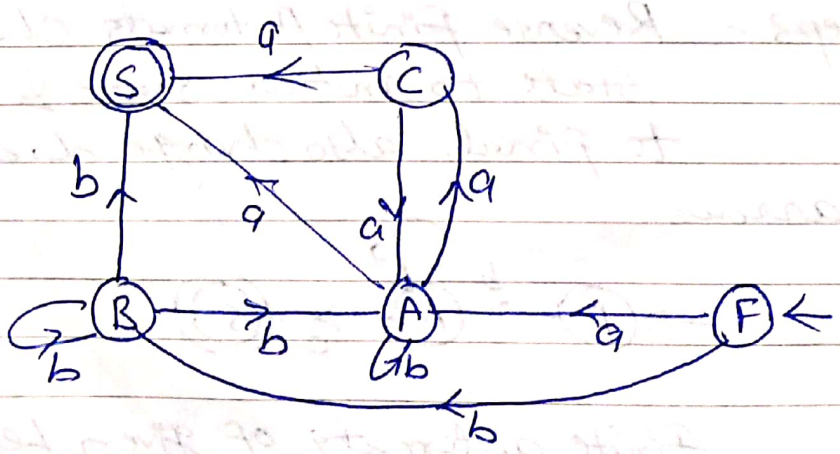
S → aC / aA / bB
A → bA / aC / bB / a
B → bB / b
C → aA.

Step II:- RLG → FA.



Step III:- Reverse FA.

F → aA / bB
A → aC / a / bA
B → bA / bB / b
C → aA / a
S →



GoodLuck | Page No. | Date

Standard Normal form of CFG

||

↓ Chomsky Normal form (CNF)

↓ Greibach Normal form (GNF)

1) Chomsky Normal form (CNF):-

A CFG is said to be in CNF if all of its production rules are of the form:

i) Single non-terminal → Single terminal

Example: $A \rightarrow a$

ii) Single non-terminal → Two non-terminal

Example $A \rightarrow BC$

iii) If ϵ is in $L(G)$? Language generated by grammer G ?

then only $S \rightarrow \epsilon$ [S is ~~not~~ Start Symbol of grammer & S is not in R.H.S of grammer]

- Every grammer in CNF is Context free that means there is only Single non-terminal on the L.H.S of each production.
- equivalent Every CFG can be transformed into CNF.

Date

- CNF is used as input in CYK algorithm which is a parsing algorithm, it employs bottom up parsing & dynamic programming - most efficient parsing algorithm.

★ Conversion of CFG to CNG (Context free grammer to chomsky normal form) :-

CFG	CNF
$A \to a$	$A \to a$ (Single T)
$A \to BC$	$A \to BC$ (2 NT)
$A \to BCDE$ (any no. of NT)	
$A \to aBcD$ (comb. of T & NT)	
$A \to abcd$ (any no. of T)	$S \to \epsilon$ [S-start Symbol not present in R.H.S]
$A \to \epsilon$	

Steps used to convert CFG to CNF.

Step1:- Convert the given grammer into Simplified grammer.

a) Eliminate null production
b) Eliminate unit production
c) Eliminate useless provariable.

Step2:- Separate all production which are already in CNF.

$A \to a$, $A \to BC$ $S \to \epsilon$.

GoodLuck | Page No. | Date

[Apply only for production which are not in CNF.]

Step 3:- If more than one terminal symbol present in production.
Replace terminals by introducing non-terminal.

$A \to x_1 x_2 \cdots x_m$, $m \geq 2$

⇓ Replace

x_1 are terminal

$A \to C_1 x_2, x_3 \cdots x_m$, $C_1 \to x_1$

$A \to C_1 C_2 x_3 \cdots x_m$, $C_2 \to x_2$

$A \to C_1 C_2 C_3 \cdots C_m$ $C_m \to x_m$

Example:- $A \to abcd$

$C_1 \to a$ $A \to C_1 bcd$
$C_2 \to b$ $A \to C_1 C_2 cd$
$C_3 \to c$ $A \to C_1 C_2 C_3 d$
$C_4 \to d$ $A \to C_1 C_2 C_3 C_4$

Step 4:- Reduce the number of non-terminal from productions if it contains more than two non-terminals

$A \to C_1 C_2 C_3 C_4 \cdots C_m$ $m \geq 3$

$\boxed{A \to C_1 D_1}$

$D_1 \to C_2 C_3 \cdots C_m$

$\boxed{D_1 \to C_2 D_2}$

$D_2 \to C_3 C_4 \cdots C_m$

$\boxed{D_{m-1} \to C_m D_m}$

Example: $A \to BCDE$
$A \to BD_1$ $(D_1 \to CDE)$
$D_1 \to CD_2$
$D_2 \to DE$

★ Conversion of CFG to CNF :-

Q. $S \to ASB \mid \varepsilon$
$A \to aAS \mid a$
$B \to SbS \mid A \mid bb$

Sol :- Step1 :- Convert the given grammer in to simplified CFG

After Simplification we get (ε, unit production)

$S \to ASB \mid AB$
$A \to aAS \mid aA \mid a$
$B \to SbS \mid bS \mid Sb \mid b \mid aBS \mid aA \mid a \mid bb$

Step2 :- Separate all productions which are already in CNF from grammer

$S \to AB$, $A \to a$, $B \to a \mid b$

Now apply following steps on remaining grammer production

Step3 :- Replace terminals by introducing new non-terminals.

$A \to aAS \longrightarrow CAS$
$A \to CAS$, $\boxed{C \to a}$

$A \to aA \Rightarrow A \to CA$

$B \to bS \Rightarrow B \to C_b S$ $\boxed{C_b \to b}$

GoodLuck | Page No.
Date

$B \to SbS \Rightarrow B \to SC_2S$

$B \to Sb \Rightarrow B \to SC_2$

$B \to aAS \Rightarrow B \to C_1AS$

$B \to aA \Rightarrow B \to C_1A$

$B \to bb \Rightarrow B \to C_2C_2$

So production set becomes

$S \to ASB / A\bar{B}$
$A \to C_1AS / C_1\bar{A} / a$
$B \to SC_2S / C_2\bar{S} / SC_2 / b / C_1AS / C_1\bar{A} / a / C_2C_2$
$C_1 \to a$
$C_2 \to b$

Step 4:- Reduce the number of non-terminals from production containing ~~same~~ more than two NTs.

i) $S \to ASB$ [3 non Terminals]
$S \to AD_1$ $\quad D_1 \to SB$

ii) $A \to C_1AS$
$A \to C_1D_2$ $\quad D_2 \to AS$

iii) $B \to SC_2S \Rightarrow B \to SD_3 = D_3 \to C_2S$

iv) $B \to C_1AS \Rightarrow B \to C_1D_2$

Now the following production are in CNF

$S \rightarrow AD_1 / AB$
$A \rightarrow C_1 D_2 / C_1 A / a$
$B \rightarrow SD_3 / C_2 S / SC_2 / b / C_1 D_2 / C_1 A / a / C_2 C_2$
$C_1 \rightarrow a$
$C_2 \rightarrow b$
$D_1 \rightarrow SB$
$D_2 \rightarrow AS$
$D_3 \rightarrow C_2 S$

Any.

Q. Convert the following CFG to CNF.

$S \rightarrow AbB / C$
$B \rightarrow AA / AC$
$C \rightarrow b / c$
$A \rightarrow a / \varepsilon$

1) Removal of ε-Production
2) Removal of unit production

Simplified CFG is

$S \rightarrow AbB / bB / Ab / b / c$
$B \rightarrow AA / a / Ac / b / c$
$C \rightarrow b / c$
$A \rightarrow a$

GoodLuck | Page No.
Date

✱ Greibach Normal form (GNF):-

A context free grammer (CFG) is in GNF if every production of the grammer is of the form.

$A \rightarrow a$ [single NT → single terminal]

$A \rightarrow aB_1B_2B_3 \dots B_n$ [Single NT → Sigle terminal followed by the number of NTs not including start Symbol]

and $S \rightarrow \epsilon$ is in G if $\epsilon \in L(G)$.

Example:-

$S \rightarrow aB \mid cA$
$A \rightarrow a \mid aS$
$B \rightarrow b \mid bAA$

In above grammer all production are in GNF.

✱ Conversion of CFG to GNF :- (CNF to GNF)

Step 1:- Convert CFG to CNF (if required)

$A \rightarrow a$ → (one terminal)
$A \rightarrow BC$ → (Two non-terminal)

1) Simplification → CNF.
2) Conversion.

Step2:- Change the names of the non-terminal symbols to A_i in ascending order of i.

$A_i \xrightarrow{\text{subscript}} A_j x$

$i = j$	$i > j$	$i < j$
$A_3 \to A_3 A_2 A_1$	$A_3 \to A_2 A_4$ ↑	$A_1 \to A_2 A_3$ ↑
Then we have to remove left recursion	Substitute R.H.S of production A_2 (repeat till $i < j$)	Substitute R.H.S of A_2 production which are in GNF.

Q $S \to AA/0$, $A \to SS/1$ Convert given CFG to GNF.

Sol:- Step1:- Conversion of CFG to CNF.
Given grammer is already in CNF form.

Step2:- Change the names of the non-terminal symbols to A_i in ascending order of i.

$S \to A_1$ $A \to A_2$

Now the given grammer bleoms.

$A_i \xrightarrow{\text{subscript}} A_j A_2 / 0$ $[i < j]$ ✓

$A_2 \to A_1 A_1 / 1$ $[i > j]$

$A_2 \rightarrow A_1A_1 / 1 \qquad (i > j)$

Substitute R.H.S of production $A_1 \rightarrow A_2A_2/0$ in above production

$A_2 \rightarrow A_2A_2A_1 / 0A_1 / 1 \qquad [i = j]$

Left recursion Remove.

α_1 β_1 β_2

$A_2 \rightarrow A_2A_2A_1 / 0A_1 / 1$

$A_2 \rightarrow 0A_1 / 1 / 0A_1Z / 1Z$ (ak bar add ke sath ak bar aur hi)

$Z \rightarrow A_2A_1 / A_2A_1Z$

$A_1 \rightarrow A_2A_2 / 0$

$A_2 \rightarrow 0A_1 / 1 / 0A_1Z / 1Z$ [GNF]

$Z \rightarrow A_2A_1 / A_2A_1Z$ [~~GNF~~]

Now select all those production which have R.H.S first non-terminal. Replace this non-terminal by new GNF alternative so that it also converted to GNF.

$A_1 \rightarrow A_2A_2 / 0$

Substitute A_2 by its R.H.S production.

$A_1 \rightarrow 0A_1A_2 / 1A_2 / 0A_1ZA_2 / 1ZA_2 / 0$

Now A_1 production is in GNF.

GoodLuck | Page No
Date

$Z \rightarrow A_2A_1 / A_2A_1Z$

Substitute A_2 by its R.H.S production.

$Z \rightarrow 0A_1A_1 / 1A_1 / 0A_1ZA_1 / 1ZA_1 / 0A_1A_12 / 1A_12 / 0A_1ZA_1Z / 1ZA_12.$

Now Z production also in GNF.

Hence the required GNF is.

$A_1 \rightarrow 0A_1A_2 / 1A_2 / 0A_1ZA_2 / 1ZA_2 / 0$

$A_2 \rightarrow 0A_1 / 1 / 0A_1Z / 1Z$

$Z \rightarrow 0A_1A_1 / 1A_1 / 0A_1ZA_1 / 1ZA_1 / 0A_1A_12 / 1A_12 / 0A_1ZA_1Z / 1ZA_12.$

Ans