UNIT-II

Pumping Lemma for Regular Languages

MREC Exam Cell

→ Pumping Lemma is used to prove that a Language is "NOT Regular"

→ It cannot be used to prove that a Language is Regular

→ If 'A' is a Regular Language, then A has a pumping Length 'p' such that any string 's' where $|s| \geq p$ may be divided into 3 parts $S = xyz$ such that the following conditions must be "true."

(1) $xy^iz \in A$ for every $i \geq 0$

(2) $|y| > 0$

(3) $|xy| \leq p$

→ ~~To prove that a~~

→ To prove that a Language is not Regular using PUMPING LEMMA, follow the below steps: (We prove using Contradiction)

MREC Exam Cell

→ Assume that A is Regular

→ It has to have a pumping length (say P)

→ All strings longer than P can be pumped $|s| \geq P$

→ Now find a string 's' in A such that $|s| \geq P$

→ Divide s into xyz

→ show that $xy^iz \notin A$ for some i

→ Then consider all ways that s can be divided into xyz

→ show that none of these can satisfy all the 3 pumping conditions at the same time.

→ S cannot be Pumped == Contradiction

(Q) Using Pumping Lemma prove that the Language $A = \{a^n b^n \mid n \geq 0\}$ is Not Regular

MREC Exam Cell

Sol:- Assume that A is Regular

pumping length of $A = P$

string $'S' = a^P b^P$

$x \quad y \quad z$

let $P = '7'$ $\quad S =$ aaaaaaabbbbbbb

Case 1: The Y is in the 'a' part

$\underbrace{aa}_{x}\underbrace{aaaa}_{y}\underbrace{abbbbbbb}_{z}$

$xy^iz \Rightarrow xy^2z$ $\quad$ aa aaaaaaaa a bbbbbbb

11 a's $\quad$ 7 b's

Case 2: The Y is in the 'b' part

$\underbrace{aaaaaaabb}_{x}\underbrace{bbbb}_{y}\underbrace{b}_{z}$

$xy^iz \Rightarrow xy^2z$ $\quad$ aaaaaaa bb bbbbbbbb b

7 a's $\quad$ 11 b's

Case 3: The y is in the 'a' and 'b' part

MREC Exam Cell

$$\underbrace{aaaa}_{x}\underbrace{aaabbb}_{y}\underbrace{bbbb}_{z}$$

$xy^iz \Rightarrow xy^2z$ aaaaa aabbaabb bbbbb

does not follow pattern a^nb^n

$\boxed{\text{'A' is not regular}}$

(Q) using pumping lemma prove that the Language $A = \{yy \mid y \in \{0,1\}^*\}$ is 'Not Regular'

Sol: Assume that A is Regular

Then A must have pumping length = P

$S = 0^P 1 0^P 1$ let $P = 7$

x y z

$S = \underbrace{00}_{x}\underbrace{00000}_{y}\underbrace{100000001}_{z}$

$xy^iz \Rightarrow xy^2z$ 000000000000100000001

$\notin A$

Equivalence of Finite Automata

MREC Exam Cell

Steps to identify equivalence

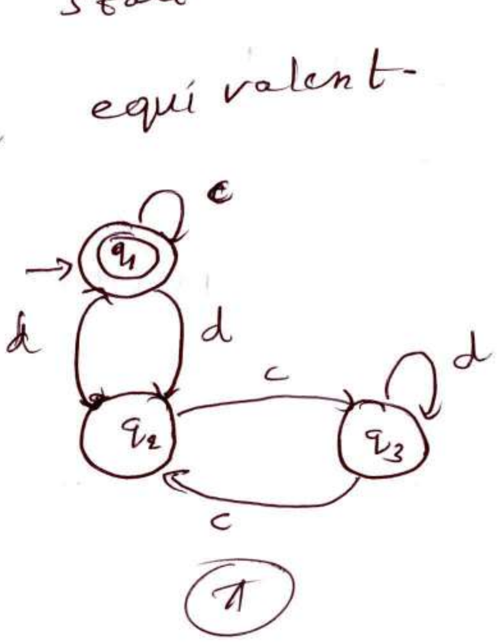
1) For any pair of states $\{q_i, q_j\}$ the transition for input $a \in \Sigma$ is defined by $\{q_a, q_b\}$ where $\delta(q_i, a) = q_a$ and

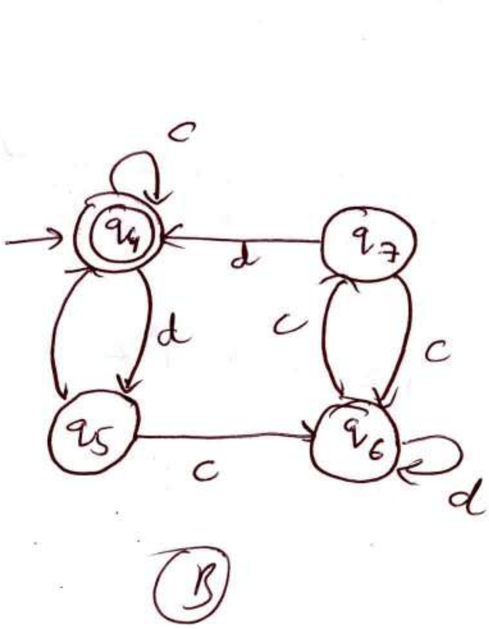
$\delta(q_j, a) = q_b$.

The two automata are not equivalent if for a pair $\{q_a, q_b\}$ one is INTERMEDIATE state and other is FINAL state.

☆ 2) If Initial state is Final state of one automaton, then in second automaton also Initial state must be Final state for them to be equivalent.

Eg



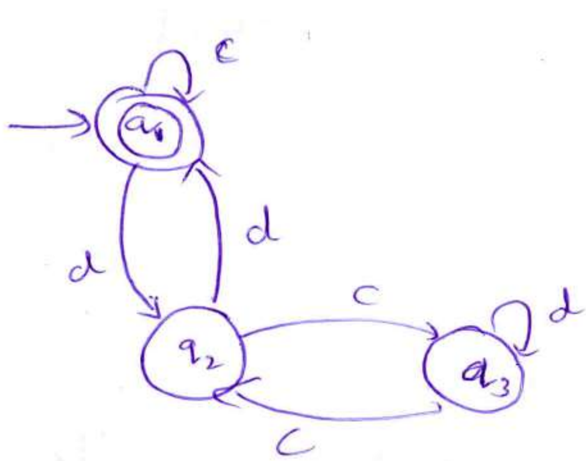


MRECExam Cell

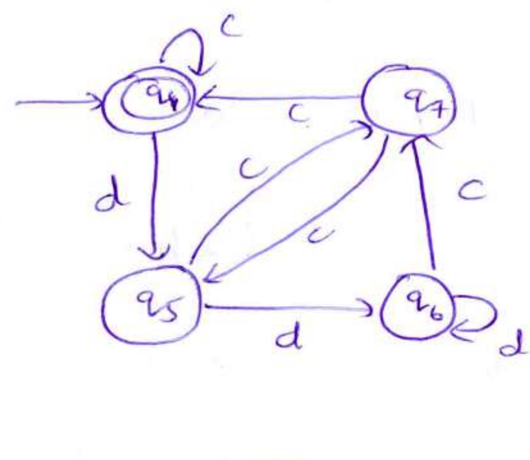
states	c	d
(q_1, q_4)	(q_1, q_4) F.S F.S	(q_2, q_5) IS IS
(q_2, q_5)	(q_3, q_6) IS IS	(q_1, q_4) F.S F.S
(q_3, q_6)	(q_2, q_7) IS IS	(q_3, q_6) IS IS
(q_2, q_7)	(q_3, q_6) IS IS	(q_1, q_4) F.S F.S

→ Both A and B automata are equivalent

(Q)



A



B

	c	d
(q_1, q_4)	(q_1, q_4) F.S F.S	(q_2, q_5) IS IS
(q_2, q_5)	(q_3, q_7) IS IS	(q_1, q_6) F.S I.S ×

(Not equivalent)

~~Clsos~~ Decision properties of Regular Languages

MREC Exam Cell

→ Irrespective of representation of regular Language, there are some fundamental questions that need to answered?

Is the given Language empty?

Is the given Language finite?

Does the string belong to given Language?

Are the two Languages equivalent?

$\boxed{\text{Decidable problem}}$

→ Emptiness (Is the given Language empty?)

Step 1: select the state that cannot be reachable from initial states & delete them (remove unreachable states)

step 2: If the resulting Machine contains atleast one final states, so then the Finite Automata accepts the none empty language.

Step 3:- If the resulting machine is free from final state, then Finite Automata accepts empty Language.

MREC Exam Cell

$\boxed{\text{Decidable Problem}}$

→ Finiteness (Is the given Language Finite?)

Step 1: select the state that cannot be reached from the initial state & delete them (remove unreachable states)

step-2: select the state from which we cannot reach the final state & delete them (remove dead states)

Step 3:- If the resulting machine contains loops or cycles then the Finite Automata accepts infinite Language

step 4: If the resulting Machine donot contain loops or cycles then the Finite Automata accepts Finite Language

→ Membership (Does the string belong to given Language)

Membership property: Let M is a finite Automata that accepts some strings over an alphabet, and let 'ω' be any string defined over MREC Exam Cell the alphabet, if there exists a transition path in M, which starts at initial state & ends in anyone of the final state then string 'ω' is a member of 'M' otherwise 'ω' is not a member of 'M'.

→ Are two Languages Equivalent?

Two Finite state Automata M_1 & M_2 is said to be equal if and only if, they accept the same Language.

→ Minimize the finite state automata and the minimal DFA will be unique.

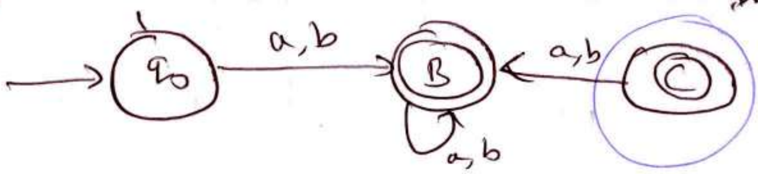
Example Emptiness

Eg

eliminate
Inaccesible
unreachable state

[No Final state] Accepts Empty Language

Eg

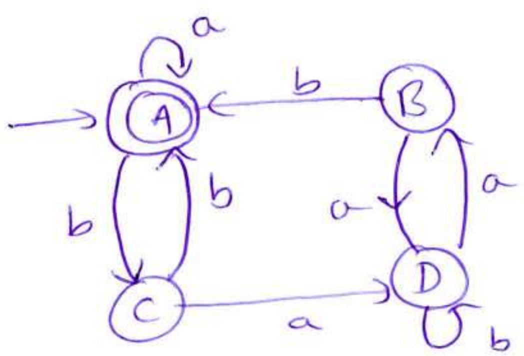


Unreachable state

MREC Exam Cell

Final state [Accepts Non-Empty Language]

Example Finiteness



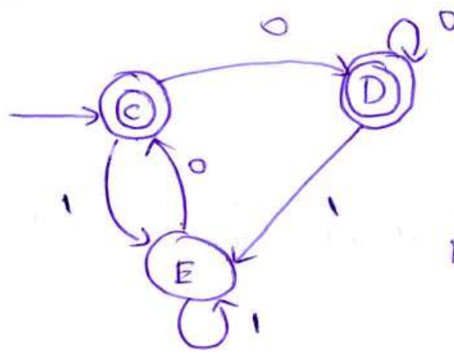
No unreachable state

No dead state

Consists of loop and cycle

So Language is Infinite Language

Example Equivalence



F {A} NF {B}

F {C, D} NF {E}

Comparision Table

	0	1
{A C}	{A D}	{B E}
{A D}	{A D}	{B E}
{B E}	{A C}	{B E}

Final and Non-final are together

Two FA are equal.

Closure properties of Regular Language

MREC Exam Cell

Regular Language is closed under

1. Union
2. Intersection
3. Concatenation & kleen Closure
4. Difference
5. Complement
6. Reverse operator

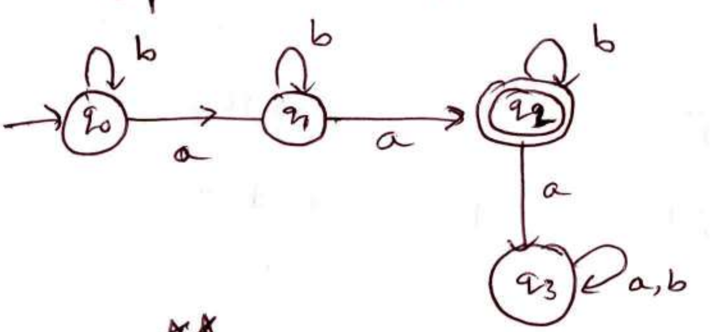
(i) union

Eg: $L = \{ w \in \Sigma^* \mid w$ has exactly 2a's (or) atleast 2b's $\}$ $\Sigma = \{a, b\}$

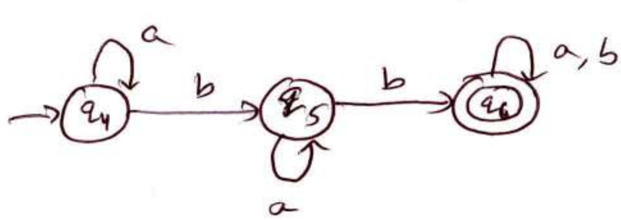
L is union of L_1 = exactly 2a's

L_2 = atleast 2b's

L_1 = exactly 2a's

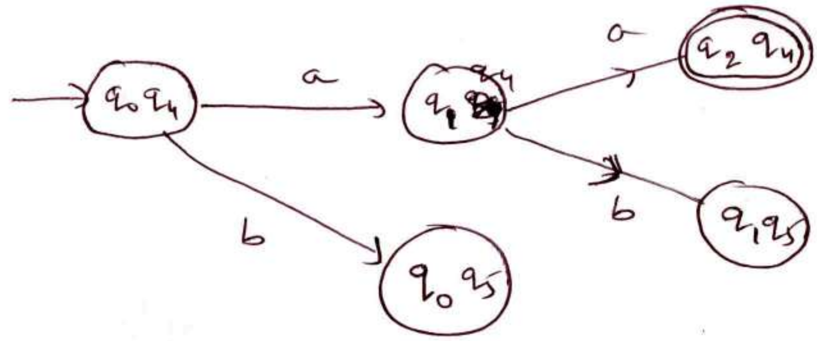


L_2 = atleast 2b's



→ $\boxed{\text{Merge two start states}}$

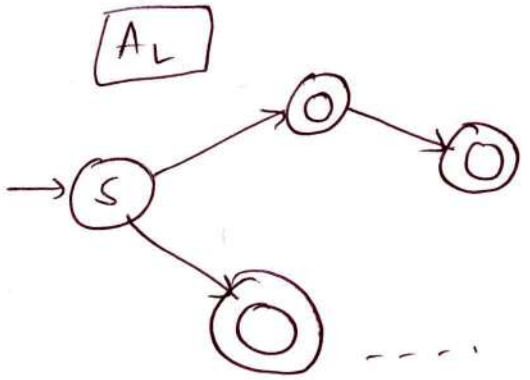
MREC Exam Cell

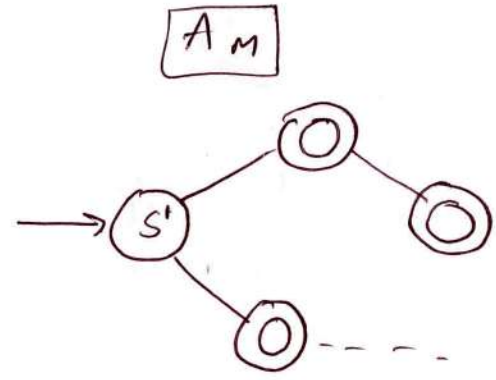


→ If L & M are Regular Languages their union is defined by

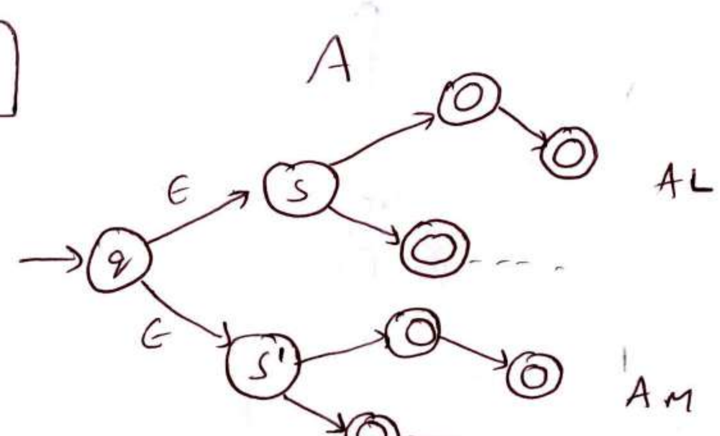
$$L \cup M = \{w : w \in L \text{ (or) } w \in M\}$$

→ If R is regular expression for L and S is regular expression for M, then R+S is a Regular Expression whose Language is L ∪ M.





$\boxed{L \cup M}$



② Intersection

MREC Exam Cell

→ If L & M are Regular Languages. So their Intersection is defined by:

$$L \cap M = \{w : w \in L \text{ and } w \in M\}$$

$$L \cap M = \overline{(\overline{L} \cup \overline{M})}$$

→ If P & Q are D.F.A for L & M respectively

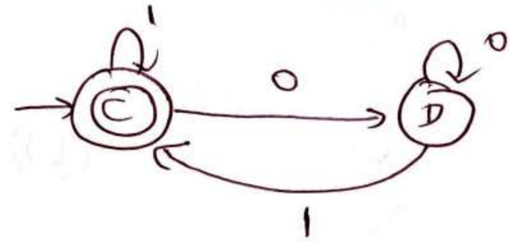
→ R is Finite Automata which is $P \cap Q$

→ Final state in R we make pairs of final state of both P & Q

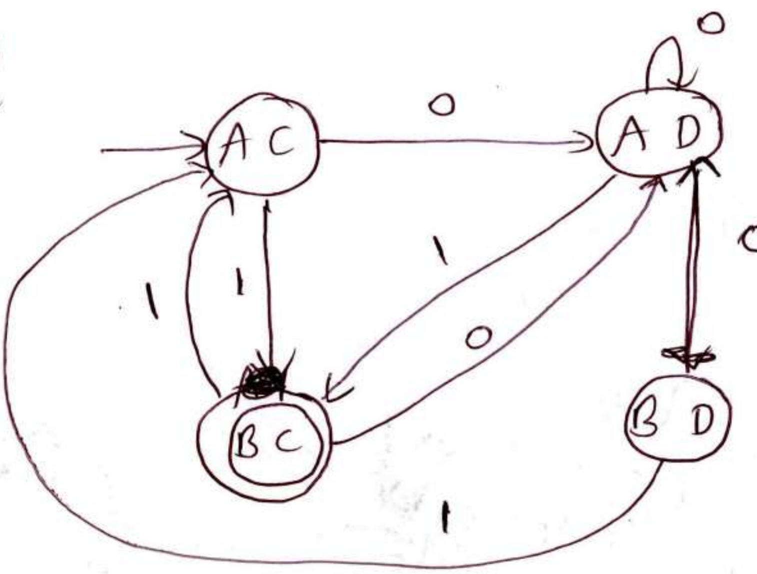
Eg:

P

Q



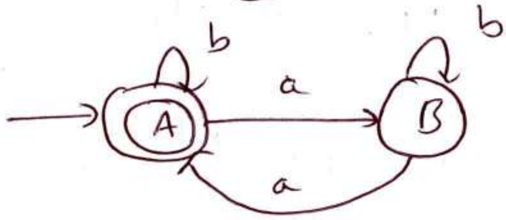
R



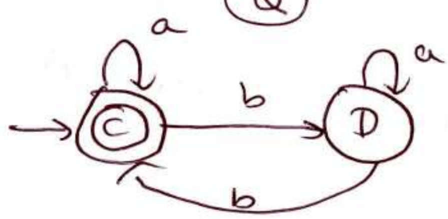
Eg: Even no of a's && Even No of b's

MREC Exam Cell

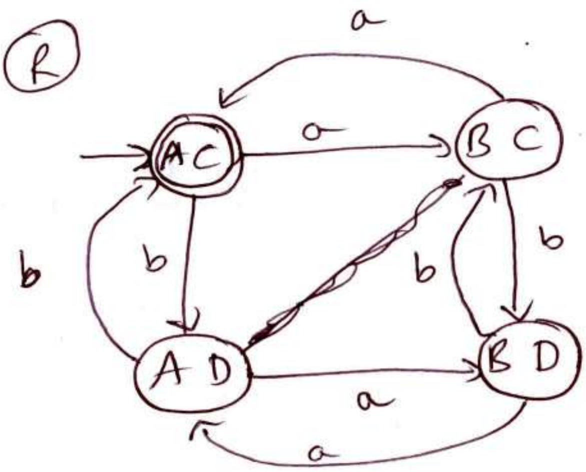
(P)



(Q)



(R)



③ Concatenation and Kleen Closure

→ If 'R' and 'S' are the Regular Expression for 'L' and 'M'.

→ 'L' and 'M' are the Regular Languages.

→ L concatenation with M is concatenation of two Regular Expression R and S

$\boxed{R \cdot S}$

→ If $\boxed{R^*}$ is a Regular Expression whose Language is L^*

Eg: starts with 'a' & ends with 'b'

MREC Exam Cell

L_1 = {starts with 'a'}

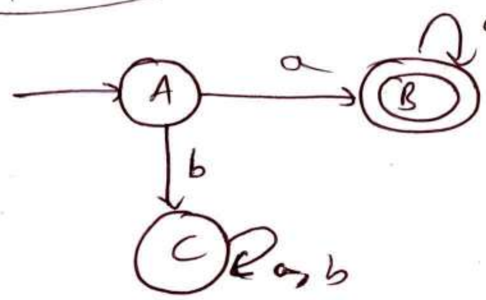
L_1 = {a, ab, aa, ...}

L_2 = {ends with 'b'}

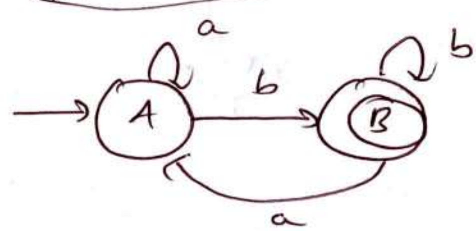
L_2 = {b, ab, bb, ...}

$L_1 \cdot L_2$ = {ab, aab, aab, ...}

DFA for L_1

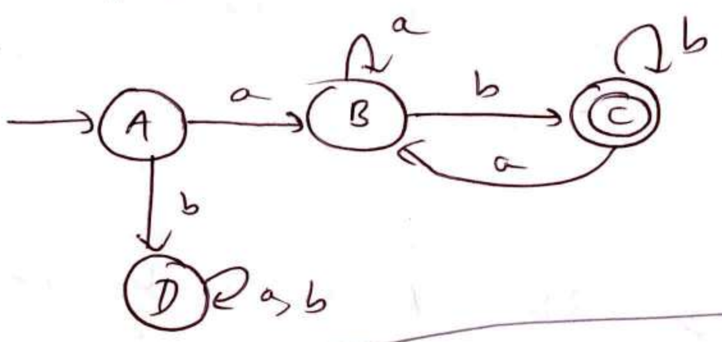


DFA for L_2



DFA for

$L_1 L_2$



Final state of D_1 and Initial state of D_2 have to be merged and transitions have to be changed accordingly

④ Difference

MREC Exam Cell

→ let L & M be two Languages then their difference is defined by

$$L - M = \{ \omega : \omega \in L \ \& \ \omega \notin M \}$$

using DeMorgan's Law

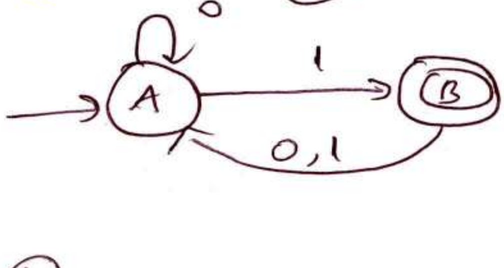
$$\boxed{L - M = L \cap M^c}$$

→ P and Q are D.F.A for L & M respectively

→ R is product Automation of P & Q

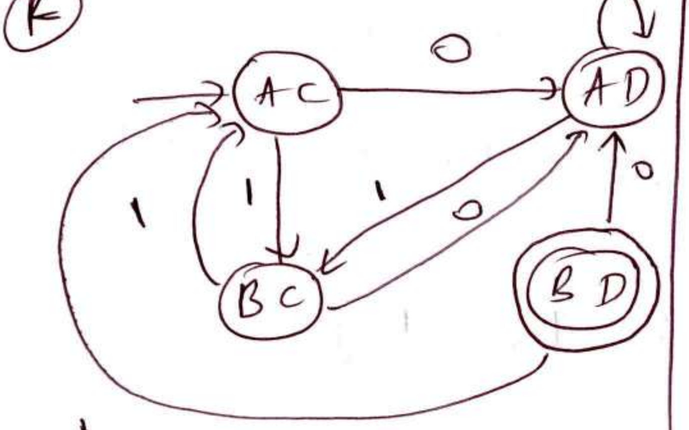
→ Final state in R is pair in 'P' but not in 'Q'.

(P)



(Q)

(R)



(5) Complement

MREC Exam Cell

→ The Complement of a Language L (with respect to an alphabet Σ such that Σ^* contains L) is $\Sigma^* - L$.

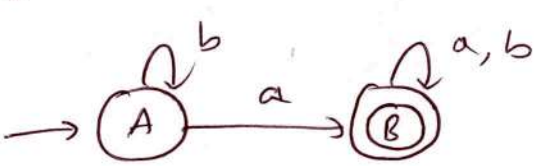
→ Since Σ^* is surely regular, the complement of a Regular Language is always Regular.

(Eg:) Language that 'does not contain a'

L_1 = {containing 'a'}

$L_1 = \{a, aa, ba, \ldots\}$

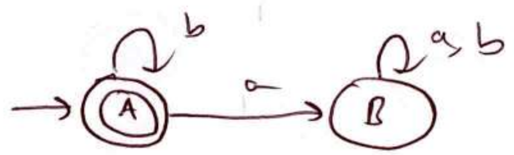
DFA for L_1



$\overline{L_1}$ = {does not contain 'a'}

$\overline{L_1} = \{\epsilon, b, bb, \ldots\}$

D.F.A for $\overline{L_1}$



$\boxed{\text{→ Make all final states to Non-Final states and Non Final states to Final}}$

$L_1 = \{Q, \Sigma, \delta, q_0, F\}$

$\overline{L_1} = \{Q, \Sigma, \delta, Q-F, q_0\}$

(6) Reversal

MREC Exam Cell

→ L is a Regular Language, L^R is the Reversal of L.

Eg:- $L = \{0, 01, 100\}$

$L^R = \{0, 10, 001\}$

→ If E is a symbol, ϵ or ϕ, then

$E^R = E$.

→ If E is

$F+G$, then $E^R = F^R + G^R$

FG, then $E^R = G^R F^R$

F^*, then $E^R = (F^R)^*$

$$\boxed{E = (01^* + 10^*)}$$

$$E^R = (01^* + 10^*)^R$$

$$= (01^*)^R + (10^*)^R$$

$$= (1^*)^R 0^R + (0^*)^R 1^R$$

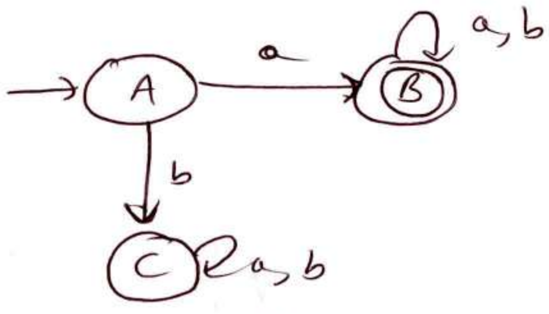
$$= (1^R)^* 0 + (0^R)^* 1$$

$$\boxed{E^R = 1^* 0 + 0^* 1}$$

Eg: L_1 = {starts with a}

L_1 = {a, aa, ab, aaa ---}

DFA for L_1



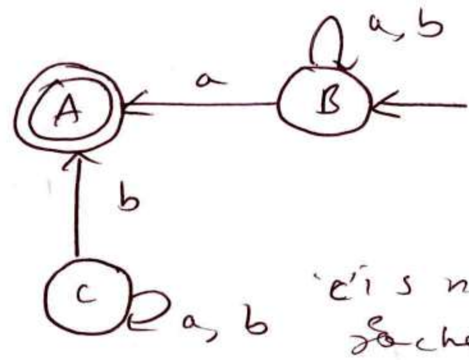
Dead state

Take each string and reverse it

MREC Exam Cell

L_1^R = {a, aa, ba, aaa, ---}

NFA not D.FA



'c' is not reached useless

→ Draw states as it is

→ Make Final state as Initial state

→ Make Initial state as Final state

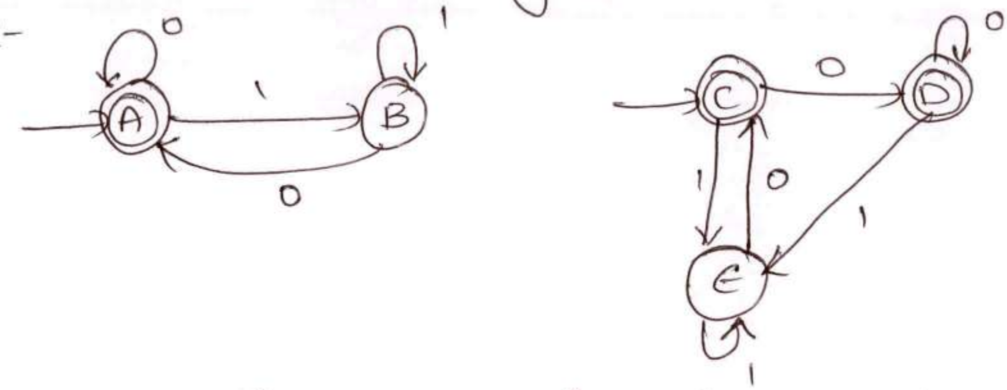
→ Just Reverse the edges

L_1 — $(DFA)^R$ — L_1^R [NFA (or) / DFA

Equivalence of Regular languages :-

MREC Exam Cell

Eg :-



$(A,C) \Rightarrow \delta(A,0) - A$ (Final state) $\quad \delta(A,1) - B$ (Intermediate state)

$\delta(C,0) - D$ (Final state) $\quad \delta(C,1) - E$ (Intermediate state)

$(A,D) \Rightarrow \delta(A,0) - A$ (FS) $\quad \delta(A,1) - B$ (IS)

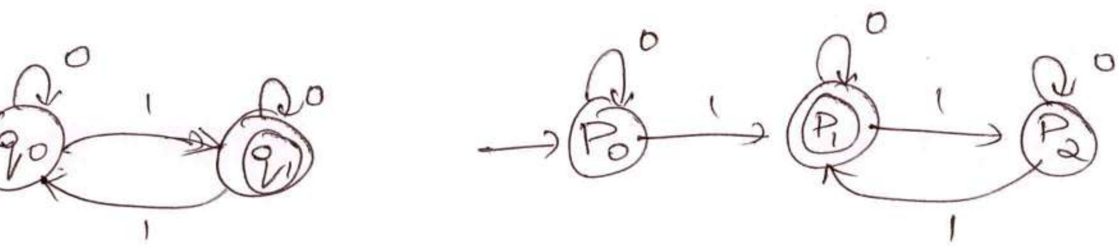
$\delta(D,0) - D$ (FS) $\quad \delta(D,1) - E$ (IS)

$(B,E) \Rightarrow \delta(B,0) - A$ (FS) $\quad \delta(B,1) - B$ (IS)

$\delta(E,0) - C$ (FS) $\quad \delta(E,1) - E$ (IS).

$\therefore$ Both the FSM are equivalent.

Eg :-



$(q_1, p_1) \Rightarrow \delta(q_1,0) - q_1$ (FS) $\quad \delta(q_1,1) - q_0$ (IS)

$\delta(P_1,0) - P_1$ (FS) $\quad \delta(P_1,1) - P_2$ (IS)

$(q_0, P_2) \Rightarrow \delta(q_0,0) - q_0$ (IS) $\quad \delta(q_0,1) - q_1$ (FS)

$\delta(P_2,0) - P_2$ (IS) $\quad \delta(P_2,1) - P_1$ (IS)

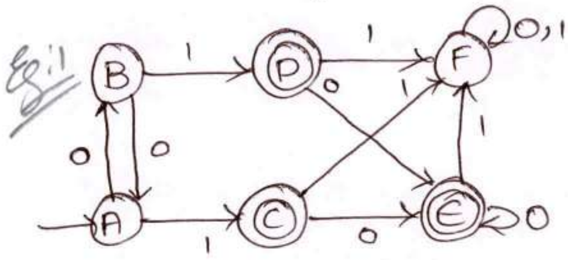
$(q_0, P_0) \Rightarrow \delta(q_0,0) - q_0$ (IS) $\quad \delta(q_0,1) - q_1$ (FS)

$\delta(P_0,0) - P_0$ (IS) $\quad \delta(P_0,1) - P_1$ (FS).

Both the FSM are equivalent.

Minimization of DFA :-

Eg:



0- Equivalent :-

$\{A, B, F\}$ $\{D, C, E\}$

1- Equivalent :-

$\left.\begin{array}{l}\delta(A,0) - B \\ \delta(B,0) - A\end{array}\right\}$ same set $\quad$ $\left.\begin{array}{l}\delta(A,1) - C \\ \delta(B,1) - D\end{array}\right\}$ same set

$\left.\begin{array}{l}\delta(A,0) - B \\ \delta(F,0) - F\end{array}\right\}$ same set $\quad$ $\left.\begin{array}{l}\delta(A,1) - C \\ \delta(F,1) - F\end{array}\right\}$ same set

$\left.\begin{array}{l}\delta(D,0) - E \\ \delta(C,0) - E\end{array}\right\}$ same set $\quad$ $\left.\begin{array}{l}\delta(D,1) - F \\ \delta(C,1) - F\end{array}\right\}$ same set

$\left.\begin{array}{l}\delta(B,0) - A \\ \delta(F,0) \; F\end{array}\right\}$ same set $\quad$ $\left.\begin{array}{l}\delta(B,1) - D \\ \delta(F,1) - F\end{array}\right\}$ different set

$\left.\begin{array}{l}\delta(C,0) - E \\ \delta(E,0) - E\end{array}\right\}$ same set $\quad$ $\left.\begin{array}{l}\delta(C,1) - F \\ \delta(E,1) - F\end{array}\right\}$ same set

$\left.\begin{array}{l}\delta(D,0) - E \\ \delta(E,0) - E\end{array}\right\}$ same set $\quad$ $\left.\begin{array}{l}\delta(D,1) - F \\ \delta(E,1) - F\end{array}\right\}$ same set

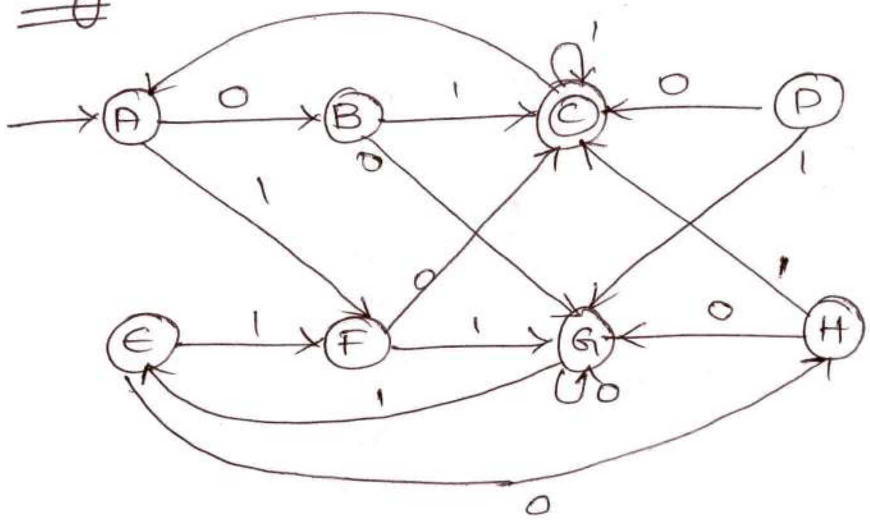
split AB from F as 1- Equivalent of F doesnt belong to same set.

$\{A, B\}$ $\{F\}$ $\{D, C, E\}$

check for 2- Equivalent & it is observed that no changes occur.

Resultent DFA :-

Eg:-2



	A	B	C	D	E	F	G
B	X						
C	X	X					
D	X	X	X				
E		X	X	X			
F	X	X	X		X		
G	X	X	X	X	X	X	
H	X		X	X	X	X	X

A,0 – B A,1 – F
B,0 – G B,1 – C } X

As FC b

(A,0) – B (A,1) – F
(D,0) – C (D,1) – G } X

(B,0) – G (B,1) – C
(D,0) – C (D,1) – G } X

(A,0) – B (A,1) – F
(E,0) – H (E,1) – F

(B,0) – G (B,1) – C
(E,0) – H (E,1) – F } X

(E,0) – H (E,1) – F
(D,0) – C (D,1) – G } X

(A,0) – B (A,1) – F
(F,0) – C (F,1) – G } X

(B,0) – G (B,1) – C
(F,0) – C (F,1) – G } X

(D,0) – C (D,1) – G
(F,0) – C (F,1) – G

(E,0) – H (E,1) – F
(F,0) – C (F,1) – G } X

(A,0) – $\boxed{B}$ (A,1) – F
(G,0) – $\boxed{G}$ X (G,1) – E

as it is marked AG also should be marked

(B,0) – G (B,1) – C
(G,0) – G (G,1) – E } X

(D,0) – C (D,1) – G
(G,0) – G (G,1) – E } X

(E,0) – $\boxed{H}$ X (E,1) – F
(G,0) – $\boxed{G}$ (G,1) – E } X

(F,0) – C (F,1) – G
(G,0) – G (G,1) – E } X

(A,0) – B (A,1) – F
(H,0) – G (H,1) – C X

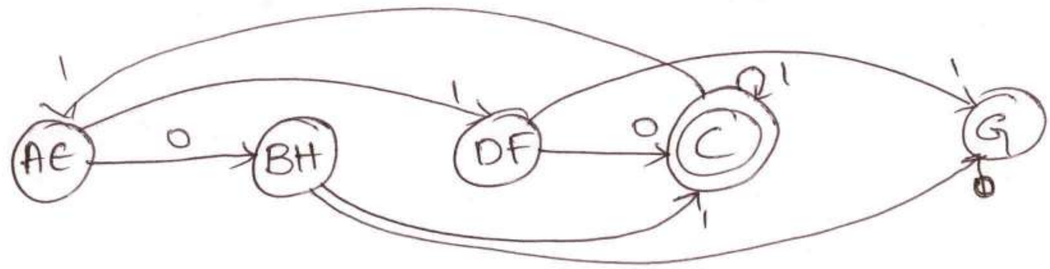
(B,0) – G (B,1) – C
(H,0) – G (H,1) – C

(D,0) – C (D,1) – G
(H,0) – G (H,1) – C } X

(E,0) – H (E,1) – F
(H,0) – G (H,1) – C } X

(F,0) – C (F,1) – G
(H,0) – G (H,1) – C } X

(G,0) – G (G,1) – E
(H,0) – G (H,1) – C } X



MREC Exam Cell

Minimized DFA

Chomsky Hierarchy of Languages

Regular Grammar:

MREC Exam Cell

Noam Chomsky gave a Mathematical Model of Grammar which is effective for writing computer Languages.

The four types of Grammar according to Noam Chomsky are:

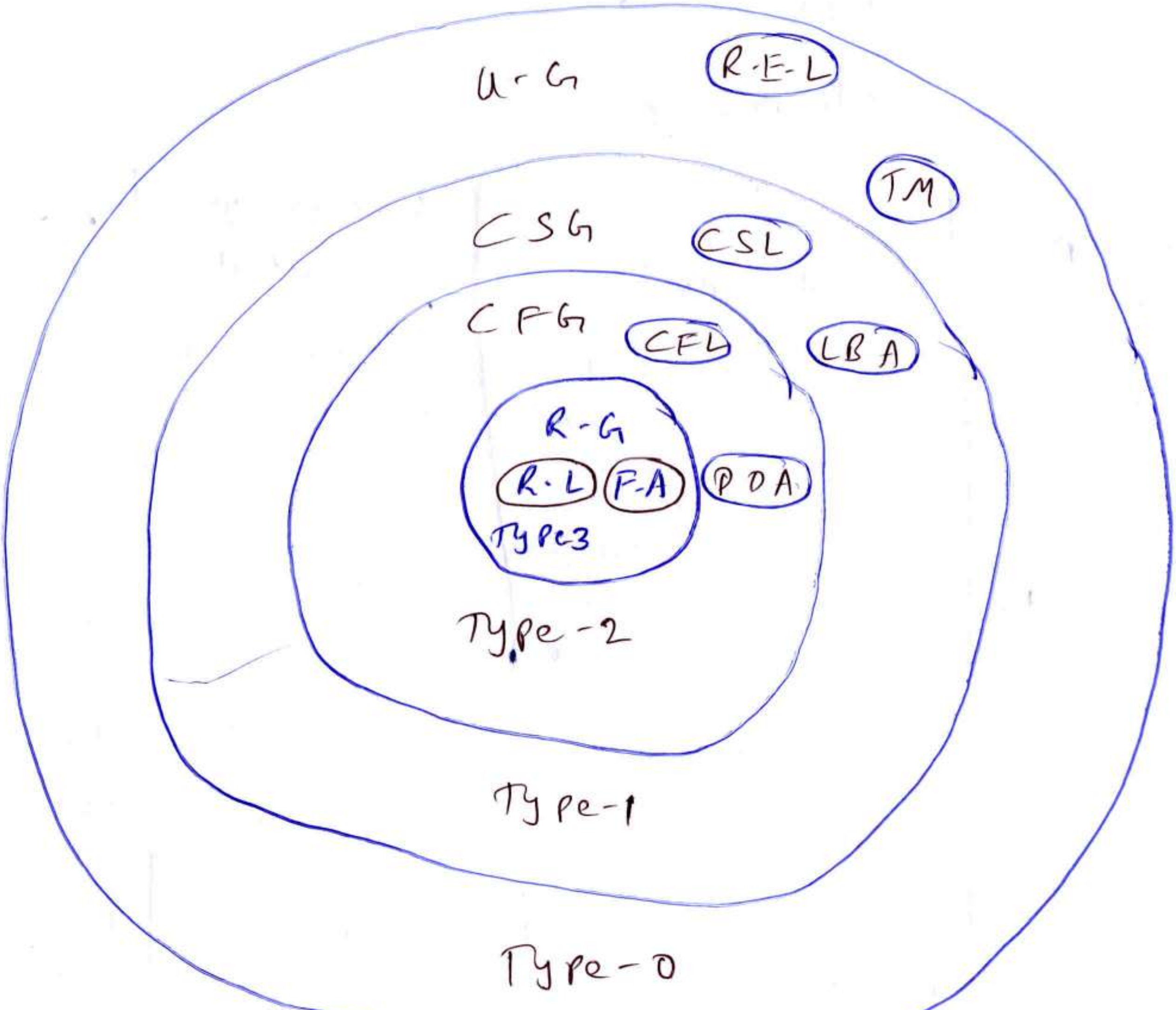
Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type-0	Unrestricted Grammar	Recursively Enumerable Language	Turing Machine
Type-1	Context Sensitive Grammar	Context Sensitive Language	Linear Bounded Automaton
Type-2	Context Free Grammar	Context Free Language	Push Down Automata
Type-3	Regular Grammar	Regular Language	Finite State Automaton

$\boxed{\begin{array}{l} \text{Type 3} \subset \text{Type 2, Type 1, Type 0} \\ \text{Type 2} \subset \text{Type 1, Type 0} \\ \text{Type 1} \subset \text{Type 0} \end{array}}$

$\boxed{Type_{i+1} \subset Type_i}$

MREC Exam Cell

→ If a Grammar is Regular Grammar then it will be C.F.G, C.S.G, U.G



Regular Grammar → Finite Automata → N.F.A & D.F.A

Context Free Grammar → Push Down Automata → NPDA & DPDA

Context Sensitive Grammar → Linear Bound Automata (LBA) → D.L.B.A & N.L.B.A

MREC Exam Cell

Unrestricted Grammar → Turing Machine → D.T.M & N.T.M

Expressive Power :- No. of languages accepted by a particular Machine is known as "Expressive power" of that Machine.

Languages Accepted by :-

Finite Automata = Regular Language

Push Down Automata = C.F.L, R.L

Linear Bound Automata = C.S.L, C.F.L, R.L

Turing Machine = R.E.L, C.S.L, C.F.L, R.L

Expressive power : T.M > L.B.A > P.D.A > F.A

$\boxed{\text{Expressive power of D.F.A = Expressive power of N.F.A}}$

MREC Exam Cell

$$\boxed{E(NPDA) > E(DPDA)}$$

$$\boxed{E(DTM) = E(NTM)}$$

→ The expressive power of DLBA and NLBA is unknown

The Default F.A is D.F.A
The Default P.D.A is NPDA
The Default T-M is D-T-M

Note

FA → Does not have Memory Element - No comparisions

$\boxed{PDA = FA + \text{stack}}$ → One Memory element

$\boxed{TM = FA + 2\ \text{stack}}$

FA + 2 stack (or)
3 stack
4 stack
has more power

$$\boxed{TM > PDA > FA}$$

Grammar :-

MREC Exam Cell

A Grammar 'G' can be formally described using 4 tuples

$\boxed{G = (V, T, S, P)}$ where

V = Set of Variables (or) Non-Terminal Symbols

T = set of Terminal symbols

S = Start symbol

P = Production rules for Terminals and Non-Terminals

→ A production rules has the form

$\boxed{\alpha \rightarrow \beta}$ where α and β are strings on (V∪T) and atleast one symbol of α belongs to V.

Type 0 (Most Powerful) - Unrestricted Grammar

$\boxed{\alpha \rightarrow \beta}$

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

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

Eg: $aAb \rightarrow bB$

$aA \rightarrow \epsilon$

Type 1 (Context Sensitive Grammar)

MREC Exam Cell

$$\boxed{\alpha \rightarrow \beta}$$

$$\boxed{\begin{array}{l} |\alpha| \leq |\beta| \\ \alpha, \beta \in (V+T)^+ \end{array}}$$

Eg: $aAb \rightarrow bbb$

$aA \rightarrow bbb$

Not Allowed

$\boxed{aAb \rightarrow bb}$ Not C.S.G

Type 2 (Context Free Grammar)

$$\boxed{A \rightarrow \alpha}$$

$$\boxed{\begin{array}{l} A \in V \\ \alpha \in (V+T)^* \end{array}}$$

Eg: $A \rightarrow \epsilon$

$A \rightarrow BCD$

(Can have many variables on R.H.S)

Type 3 (~~context~~ Regular Grammar)

Right Linear :	$A \rightarrow xB \mid x$
Left Linear :	$A \rightarrow Bx \mid x$

$$\boxed{\begin{array}{l} A, B \in V \\ x \in \Sigma^* \ (T^*) \end{array}}$$

what is Context ?

MREC Exam Cell

$a \, A \, b \to b\,b$

Left Context of A

Right Context of A

but in CFG we write production as

$V \to (V+T)^*$

$\epsilon \qquad \epsilon$

$\epsilon \, A \, \epsilon$ So Freeing the Context

No Context Left

No context Right

→ Means Left and Right context of this variable is ϵ.

→ That is why these are known as Context Free Grammar (CFG)

Linear Grammar: In any Grammar if there exist exactly one variable on the L.H.S and atmost one variable on R.H.S is known as "Linear Grammar"

Eg: $S \to aSb \mid bSa \mid \epsilon \to$ $\boxed{\text{Middle Linear}}$

MREC Exam Cell

$A \to aB \mid b \to$ $\boxed{\text{Right Linear}}$

$S \to aS \mid bS \mid a \to$ $\boxed{\text{Right Linear}}$

Context Free Language

→ In Formal Language theory, a Context Free Language is a Language generated by some Context Free Grammar.

→ The set of all CFL is identical to the set of languages accepted by Pushdown Automata.

→ CFG is defined by 4 tuples as

$G = \{V, \Sigma, S, P\}$ where

V = set of variables (or) Non-Terminal Symbols

Σ = set of Terminal symbols.

S = Start symbol

P = Production rules.

→ Production Rule for CFG is of the form

$A \to a$ where $a = \{V \cup \Sigma\}^*$ and $A \in V$

Eg:- For generating a Language that generates equal no of a's and b's in the form ~~of~~ $a^n b^n$, the C-F-G will be defined as

$$\boxed{G = \{ \{S, A\}, (a, b), S, (S \to aAb, A \to aAb \mid \epsilon) \}}$$

MREC Exam Cell

$S \to a\underline{A}b$ (by $S \to aAb$)

$\to aaAbb$ (by $A \to aAb$)

$\to aabb$ (by $A \to \epsilon$)

$\to a^2 b^2 \Rightarrow \boxed{a^n b^n}$

Method to find whether a string belongs to a Grammar or not

1) start with the start symbol and choose the closest production that matches to the given string

2) Replace the variables with its most appropriate production. Repeat the process until the string is generated or until no other productions are left.

Eg: verify whether the Grammar $S \to 0B \mid 1A$, $A \to 0 \mid 0S \mid 1AA \mid \epsilon$, $B \to 1 \mid 1S \mid 0BB$ generates string 00110101

MREC Exam Cell

$S \rightarrow 0B \quad (S \rightarrow 0B)$

$\rightarrow 00BB \quad (B \rightarrow 0BB)$

$\rightarrow 001B \quad (B \rightarrow 1)$

$\rightarrow 0011S \quad (B \rightarrow 1S)$

$\rightarrow 00110B \quad (S \rightarrow 0B)$

$\rightarrow 001101S \quad (B \rightarrow 1S)$

$\rightarrow 0011010B \quad (S \rightarrow 0B)$

$\rightarrow 00110101 \quad (B \rightarrow 1)$

Generated by the Grammar.

(Q) verify whether the Grammar $S \rightarrow aAb$, $A \rightarrow aAb/\epsilon$ generates string aabbb.

$S \rightarrow aAb$

$\rightarrow aaAbb \quad (A \rightarrow aAb)$

$\rightarrow aa\epsilon bb \quad (A \rightarrow \epsilon)$

$\rightarrow aaaAbbb \quad (A \rightarrow aAb)$

$aaabbb \quad (A \rightarrow \epsilon)$

$\rightarrow$ Cannot be generated by Grammar.

Derivation Tree (Parse Tree)

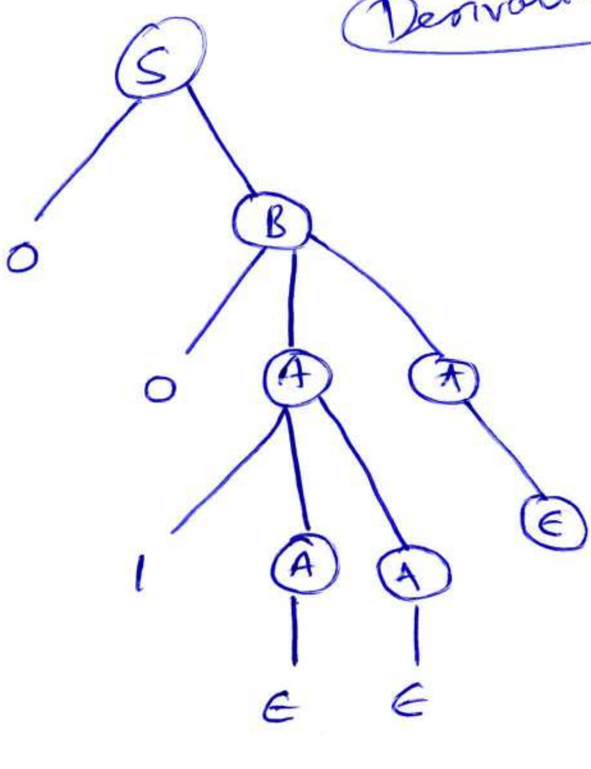
MREC Exam Cell

→ A Derivation Tree (or) Parse Tree is an ordered rooted tree that graphically represents the semantic information of strings derived from a Context Free Grammar.

Eg:- For the Grammar $G = \{V, T, P, S\}$ where

$\{S \rightarrow 0B,\ A \rightarrow 1AA \mid \epsilon,\ B \rightarrow 0AA\}$

Derivation Tree



$\boxed{0\ 0\ 1}$ → string

Root vertex: Must be labelled by start symbol

vertex: Labelled by Non-Terminal symbols

Leaves: Labelled by Terminal symbols (or) ϵ

Left Most Derivation

MREC Exam Cell

→ A Left Most Derivation is obtained by applying production to the leftmost variable in each step.

Eg: For generating the string aabaa from the Grammar $S \rightarrow aAS \mid aSS \mid \epsilon$,

$A \rightarrow SbA \mid ba$

$$S \rightarrow a\underline{S}S$$
$$\rightarrow aa\underline{A}SS \quad [S \rightarrow aAS]$$
$$\rightarrow aaba\underline{S}S \quad [A \rightarrow ba]$$
$$\rightarrow aabaa\underline{S}S \quad [S \rightarrow aSS]$$
$$\rightarrow aabaa\epsilon S \quad [S \rightarrow \epsilon]$$
$$\rightarrow \underline{aabaa} \quad [S \rightarrow \epsilon]$$

Always left Most Symbol is being replaced.

Right Most Derivation

MREC Exam Cell

→ A Right Most Derivation is obtained by applying production to the Right Most variable in each step.

Eg:, For generating the string aabaa from the Grammar $S \to aAS \mid aSS \mid \epsilon$,

$A \to SbA / ba$

$S \to aS\underline{S}$ $[S \to aSS]$

$S \to aSaA\underline{S}$ $[S \to aSS]$

$S \to aSaAaS\underline{S}$ $[S \to aSS]$

$S \to aSaAaS$ $[S \to \epsilon]$

$S \to aSa\underline{A}a$ $[S \to \epsilon]$

$S \to a\underline{S}abaa$ $[A \to ba]$

$S \to aabaa$ $[S \to \epsilon]$

→ Always Right Most symbol is being replaced.

Ambigous Grammar

MREC Exam Cell

→ A Grammar is said to be Ambigous if there exists two or) more derivations or) derivation trees for a string 'w' (that means two or) more Left Most Derivation or) Left Derivation trees]

Eg: $G = (\{S\}, \{a+b, +, *\}, P, S\}$ where

$P = \{S \to S+S \mid S*S \mid a \mid b\}$

string to be generated is $a+a*b$

$S \to S+S$
$\to a+S$
$\to a+S*S$
$\to a+a*S$
$\to a+a*b$

$S \to S*S$
$S \to S+S*S$
$S \to a+S*S$
$S \to a+a*S$
$S \to a+a*b$

→ The above grammar is Ambigous.

MREC Exam Cell

Eg 2 :- $S \rightarrow aB \mid bA$

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

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

string :- "aabbabba"

Left Most Derivation

$S \rightarrow a\underline{B}$

$\rightarrow aa\underline{B}B$

$\rightarrow aab\underline{S}B$

$\rightarrow aabb\underline{A}B$

$\rightarrow aabba\underline{B}$

$\rightarrow aabbab\underline{S}$

$\rightarrow aabbabb\underline{A}$

$\rightarrow aabbabba$

Right Most Derivation

$S \rightarrow a\underline{B}$

$\rightarrow aaB\underline{B}$

$\rightarrow aaBb\underline{S}$

$\rightarrow aaBbb\underline{A}$

$\rightarrow aa\underline{B}bba$

$\rightarrow aab\underline{S}bba$

$\rightarrow aabb\underline{A}bba$

$\rightarrow aabbabba$

Eg3 :- $S \rightarrow aAS \mid aSS \mid \epsilon$ $\quad A \rightarrow SbA \mid ba$ string:- "aabbaa"

Left Most Derivation

$S \rightarrow a\underline{A}S$

$\rightarrow a\underline{S}bAS$

$\rightarrow aa\underline{SS}bAS$

$\rightarrow aab\underline{A}S$

$\rightarrow aabba\underline{S}$

$\rightarrow aabbaa\underline{S}S$

$\rightarrow aabbaa\underline{S}$

$\rightarrow aabbaa$

Right Most Derivation

$S \rightarrow aA\underline{S}$

$\rightarrow aAaS\underline{S}$

$\rightarrow aAa\underline{S}$

$\rightarrow a\underline{A}a$

$\rightarrow aSb\underline{A}a$

$\rightarrow a\underline{S}bbaa$

$\rightarrow aa\underline{SS}bbaa$

$\rightarrow aa\underline{S}bbaa$

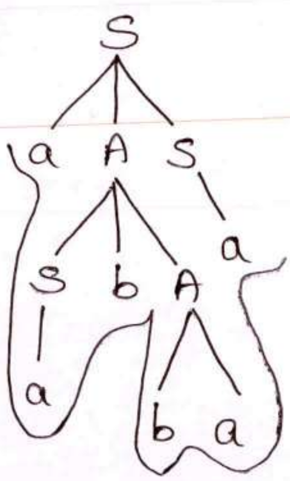
$\rightarrow aabbaa$

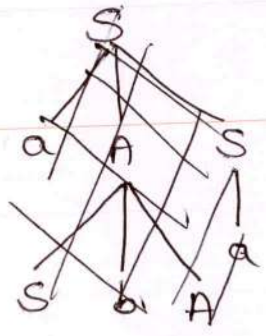
Examples on Parse Trees :-

MREC Exam Cell

$S \rightarrow aAS \mid a$ string :- "aabbaa"

$A \rightarrow SbA \mid SS \mid ba$

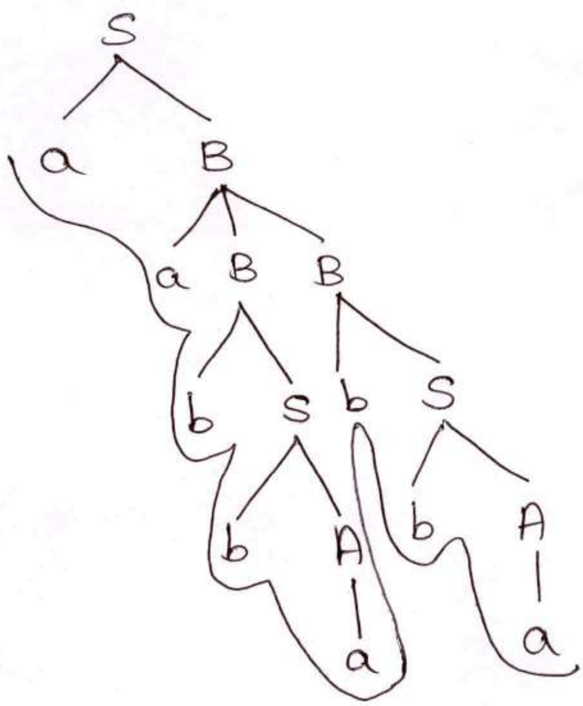




$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$ string :- "aabbabba"

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



$S \rightarrow AB \mid \varepsilon \quad A \rightarrow aB \quad B \rightarrow Sb \quad$ String :- aabbbb

MREC Exam Cell

$S \rightarrow AB$

$\rightarrow aBB$

$\rightarrow aSbB$

$\rightarrow aABbB$

$\rightarrow aaBBbB$

$\rightarrow aaSbBbB$

$\rightarrow aa\varepsilon bBbB$

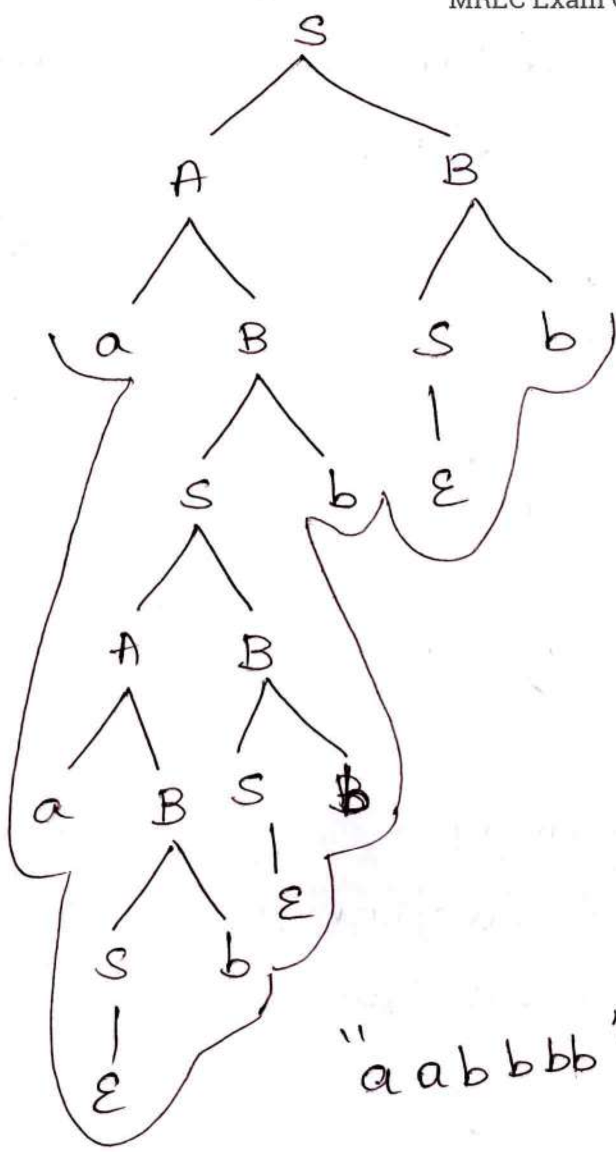
$\rightarrow aabSbbB$

$\rightarrow aab\varepsilon bbB$

$\rightarrow aabbbSb$

$\rightarrow aabbb\varepsilon b$

$\rightarrow aabbbb$



"aabbbb"

Examples on Ambiguity of Grammer :-

MREC Exam Cell

$E \rightarrow E+E \mid E*E \mid (E) \mid id$ string :- id + id * id

$E \rightarrow \underline{E}+E$	$E \rightarrow \underline{E}*E$
$\rightarrow id + \underline{E}*E$	$\rightarrow \underline{E}+E*E$
$\rightarrow id+id*\underline{E}$	$\rightarrow id+\underline{E}*E$
$\rightarrow id+id*id$	$\rightarrow id+id*E$
	$\rightarrow id+id*id$

$S \rightarrow aSbS \mid bSaS \mid \varepsilon$ string :- "abab"

$S \rightarrow a\underline{S}bS$	$S \rightarrow a\underline{S}bS$
$\rightarrow ab\underline{S}aSbS$	$\rightarrow a\varepsilon b\underline{S}$
$\rightarrow ab\varepsilon a\underline{S}bS$	$\rightarrow aba\underline{S}bS$
$\rightarrow aba\varepsilon b\underline{S}$	$\rightarrow aba\varepsilon bS$
$\rightarrow abab\varepsilon$	$\rightarrow abab\varepsilon$
$\rightarrow abab$	$\rightarrow abab$