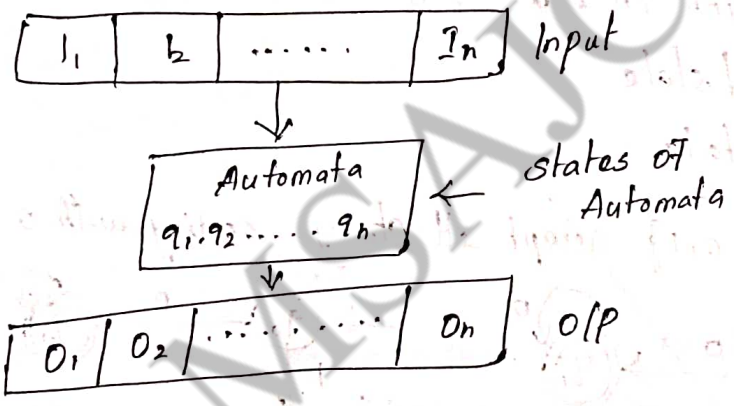
Automata and Regular Expression

Finite Automata :-

* Finite Automat (FA) is the simplest machine.
* FA are used to recognize patterns.
* It have 5 elements or tuples.
* It has a set of states and rules for moving from one state to another (or) stay in same state.
* Finite Automata have two states. Accept state and Reject state
* when the i/p string is processed successfully, & the automata reached its final state, then it will accept.

Features of Automation :



Formal Definition of FA :

A finite automata is a collection of 5-tuple $(Q, \Sigma, \delta, q_0, F)$

Q : Finite set of states

Σ : finite set of i/p symbol (Alphabets)

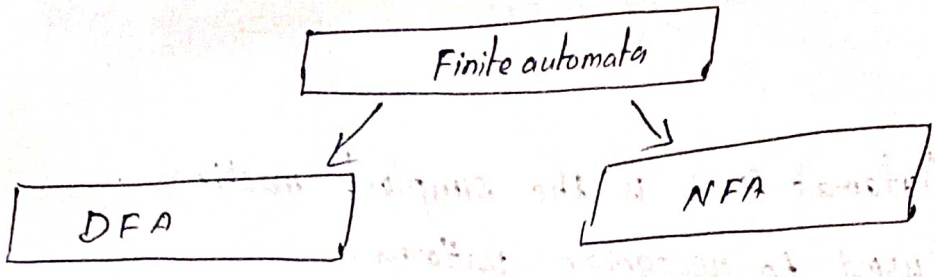
δ : Transition fn

q_0 : Initial state

F : Finial state.

Types of Automata :

* DFA (Deterministic Finite Automata)
* NFA (Non-Deterministic " ")



DFA :

* Deterministic Finite Automata
* Deterministic refers to the uniqueness of the computation.
* DFA does not accept the null move.
* It goes to one state only for a particular i/p character

DFA consist of 5 tuples :

Q : set of states

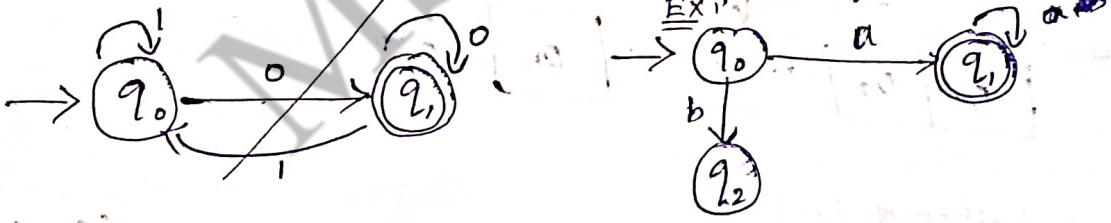
Σ : set of i/p symbols

δ : Transition fn $\delta : [\delta : Q \times \Sigma \to Q]$

q : Initial state

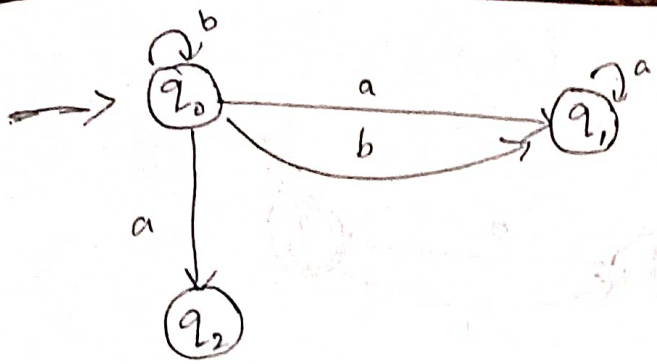
F : Final state

Ex : DFA with $\Sigma = \{0, 1\}$ accept ~~all strings ending with 0~~



NFA :

* Non-deterministic Finite Automata.
* It can accept the null move.
* Ability to transmit to any number of states for a particular i/p
* It have many path for a specific i/p from the current state to the nxt state
* It is easy to construct NFA, when compared to DFA.
* Every NFA is not DFA, But Each NFA can be translated into DFA.



* NFA also has five states same as DFA. but different transition fn.

$$\delta : Q \times \Sigma \to 2^Q$$

where,

Q : finite set of states

Σ : finite set of i/p symbol

q_0 : Initial state

F : Final state

δ : Transition fn.

Graphical Representation:-

=> The state is represented by vertices

=> The arc labeled with an i/p character show the transitions

=> The initial state is marked with an arrow.

=> The final state is denoted by the double circle.

Example 1:

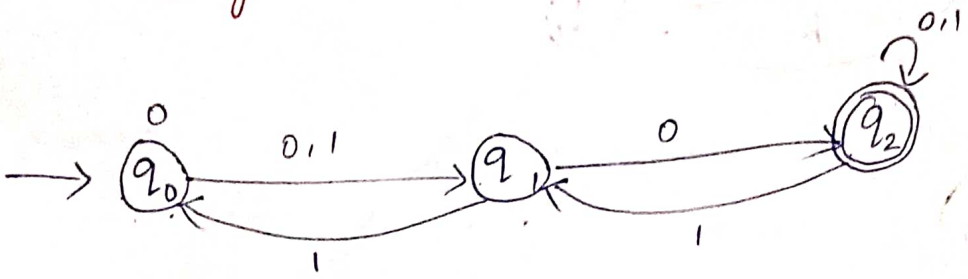
$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0, 1\}$

$q_0 = \{q_0\}$

$F = \{q_2\}$

Transition Diagram:-



Transition Table:-

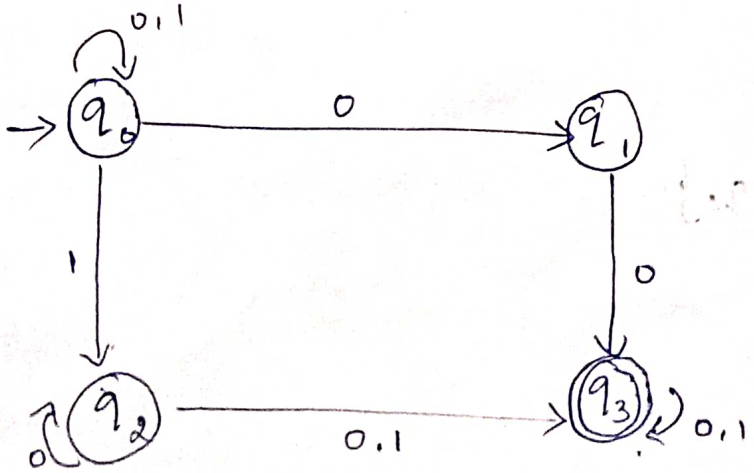
Present state	Next state Input 0	Input 1
→ q_0	q_0, q_1	q_1
q_1	q_2	q_0
* q_2	q_2	q_1, q_2

→

Ex :- 2

Design a NFA for the transition table as given below:

Present state	0	1
→ q_0	q_0, q_1	q_0, q_2
q_1	q_3	ε
q_2	q_2, q_3	q_3
* q_3	q_3	q_3



$\delta(q_0, 0) = \{q_0, q_1\}$

$\delta(q_0, 1) = \{q_0, q_2\}$

$\delta(q_1, 0) = \{q_3\}$

$\delta(q_1, 1) = \varepsilon$

$\delta(q_2, 0) = \{q_2, q_3\}$

$\delta(q_2, 1) = \{q_3\}$

$\delta(q_3, 0) = \{q_3\}$

$\delta(q_3, 1) = \{q_3\}$

→

Example 3: Design a NFA with $\Sigma = \{0,1\}$ accepts all string in which the third symbol from right end is always 0

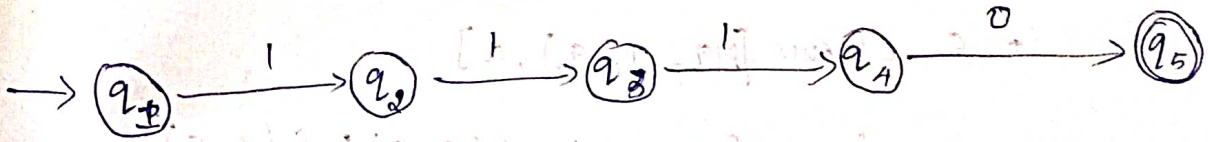
Right end third place always 0

It is NFA. Becoz in state q_0 with i/p 0 we can either go to state q_0 or q_1

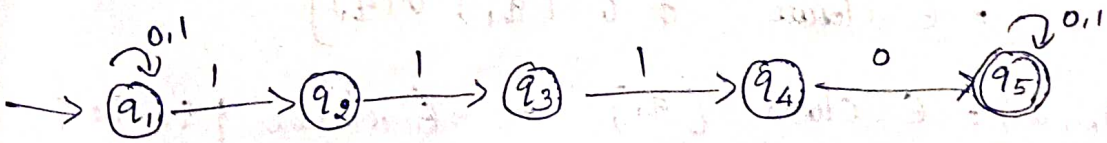
→

Example 4:

Design NFA in which all the string contain a substring 1110



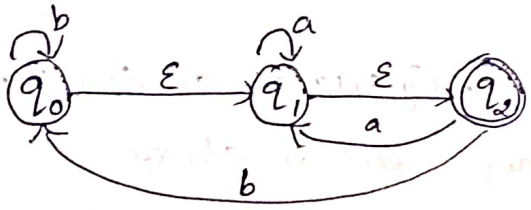
we will add the i/p 0's & 1's and 1110 string can be maintained.



Present state	0	1
→ q_1	q_1	q_1, q_2
q_2	ϕ	q_3
q_3	ϕ	q_4
q_4	q_5	ϕ
* q_5	q_5	q_5

→

1) Conversion of NFA with Epsilon to without ε :-



1. ε - closure
2. $\delta'(q,a) = \epsilon\text{-closure}(\delta(\epsilon\text{-cl}$ q
3. $F' = F \cup \{q_0\}$ if ε-closure q_0 is having Final

Step 1: ε-closure moves

ε - closure $(q_0) = \{q_0, q_1, q_2\}$

ε - closure $(q_1) = \{q_1, q_2\}$

ε - closure $(q_2) = \{q_2\}$

Step 2: δ' for ε-closure

$\delta'(q_0, a) = \varepsilon$ - closure $[\delta((\varepsilon\text{-closure}(q_0), a)]$

$= \varepsilon$ - closure $[\delta(q_0, q_1, q_2), a]$

$= \epsilon$ - closure $[\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)]$

$= \epsilon$ - closure $[\phi \cup (q_1) \cup (q_1)]$

$\delta'(q_0, a) = \epsilon$ - closure $[q_1] \Rightarrow$ ε-closure $\{q_1, q_2\}$ //

$\delta'(q_0, b) = \epsilon$ - closure $[\delta(\epsilon\text{-closure}(q_0), b)]$

$= \epsilon$ - closure $[\delta(q_0, q_1, q_2), b)]$

$= \varepsilon$ - closure $[\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)]$

$= \epsilon\text{-closure}[q_0 \cup \phi \cup q_0]$

$= \epsilon\text{-closure}(q_0)$

$\delta'(q_0, b) = \{q_0, q_1, q_2\}$ //

$\delta'(q_1, a) = \epsilon\text{-closure}[\delta(\epsilon\text{-closure}(q_1), a)]$

$= \epsilon\text{-closure}[\delta(q_1, q_2), a]$

$= \epsilon\text{-closure}[\delta(q_1, a) \cup \delta(q_2, a)]$

$= \epsilon\text{-closure}[q_1 \cup q_1]$

$= \epsilon\text{-closure}[q_1]$

$\delta'(q_1, a) = \{q_1, q_2\}$ //

$\delta'(q_1, b) = \epsilon\text{-closure}[\delta(\epsilon\text{-closure}(q_1), b)]$

$= \epsilon\text{-closure}[\delta(q_1, q_2), b]$

$= \epsilon\text{-closure}[\delta(q_1, b) \cup \delta(q_2, b)]$

$= \epsilon\text{-closure}[\phi \cup q_0]$

$\delta'(q_1, b) = \{q_0, q_1, q_2\}$ //

$\delta'(q_2, a) = \epsilon\text{-closure}[\delta(\epsilon\text{-closure}(q_2), a)]$

$= \epsilon\text{-closure}[\delta(q_2, a)]$

$= \epsilon\text{-closure}\ q_1$

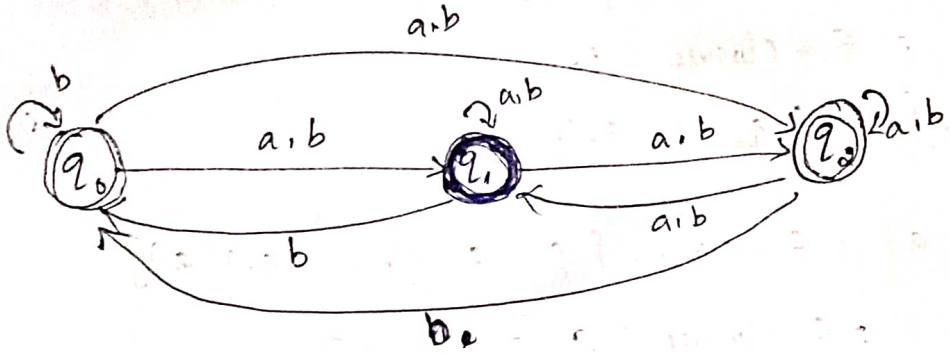
$\delta'(q_2, a) = \{q_1, q_2\}$ //

$\delta'(q_2, b) = \epsilon\text{-closure}[\delta(\epsilon\text{-closure}(q_2), b)]$

$= \epsilon\text{-closure}[\delta(q_2, b)]$

$= \epsilon\text{-closure}[q_0]$

$\delta'(q_2, b) = \{q_0, q_1, q_2\}$ //



$F \cup \{q_0\}$ if ϵ-closure (q_0) is having F as member

$F' = F \cup \{q_0\}$

$= \{q_0, q_2\}$

→

Ex :- 2

step 1:

ε - closure q_0, q_1, q_2

ε - closure $q_0 = \{q_0\}$

ϵ - closure $q_1 = \{q_1, q_2\}$

ϵ - closure $q_2 = \{q_2\}$

step 2: δ' transition on each i/p symbol.

$\delta'(q_0, a) = \epsilon$ - closure $(\delta(\epsilon\text{-closure}(q_0), a))$

$= \epsilon$ - closure $(\delta(q_0, a))$

$= \epsilon$ - closure q_1

$\delta'(q_0, a) = \{q_1, q_2\}$

$\delta'(q_0, b) = \epsilon$ - closure $(\delta(\epsilon\text{-closure}(q_0), b))$

$= \epsilon$ - closure $(\delta(q_0, b))$

$= \epsilon$ - closure (ϕ)

$\delta'(q_0, b) = \phi$

$\delta'(q_1, a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), a))$

$= \epsilon\text{-closure}(\delta(q_1, q_2), a))$

$= \epsilon\text{-closure}(\delta(q_1, a) \cup \delta(q_2, a))$

$= \epsilon\text{-closure}(\phi \cup \phi)$

$\delta'(q_1, a) = \phi$

$\delta'(q_1, b) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), b))$

$= \epsilon\text{-closure}(\delta(q_1, q_2), b))$

$= \epsilon\text{-closure}(\delta(q_1, b) \cup \delta(q_2, b))$

$= \epsilon\text{-closure}(\phi \cup q_2)$

$= \epsilon\text{-closure}(q_2)$

$\delta'(q_1, b) = \{q_2\}$

$\delta'(q_2, a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), a))$

$= \epsilon\text{-closure}(\delta(q_2), a))$

$= \epsilon\text{-closure}(\delta(q_2, a))$

$= \epsilon\text{-closure}(\phi)$

$\delta'(q_2, a) = \phi$

$\delta'(q_2, b) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), b))$

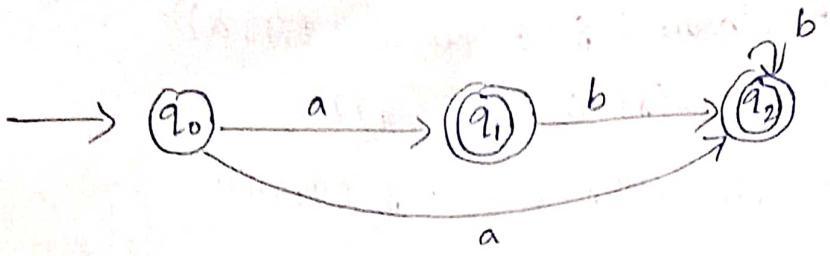
$= \epsilon\text{-closure}(\delta(q_2), b))$

$= \epsilon\text{-closure}(\delta(q_2, b))$

$= \epsilon\text{-closure}(q_2)$

$\delta'(q_2, b) = q_2$

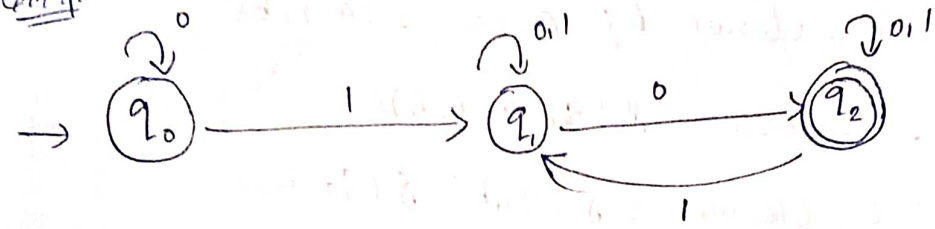
states	a	b
$\rightarrow q_0$	$\{q_1, q_2\}$	ϕ
$* q_1$	ϕ	$\{q_2\}$
$* q_2$	ϕ	$\{q_2\}$



→

Conversion from NFA to DFA :

Step:



Construct the transition table.

state	0	1
→ q_0	q_0	q_1
q_1	$\{q_1, q_2\}$	$\{q_1\}$
* q_2	$\{q_2\}$	$\{q_1, q_2\}$

δ' transition for all states.

$\delta'(q_0, 0) = [q_0]$

$\delta'(q_0, 1) = [q_1]$

$\delta'(q_1, 0) = [q_1, q_2]$ → New state.

$\delta'(q_1, 1) = [q_1]$

$\delta'((q_1, q_2), 0) = \delta'(q_1, 0) \cup \delta'(q_2, 0)$

$= \{q_1, q_2\} \cup \{q_2\}$

$= [q_1, q_2]$ //

$\delta'((q_1, q_2), 1) = \delta'(q_1, 1) \cup \delta'(q_2, 1)$

$= \{q_1\} \cup \{q_1, q_2\}$

$= [q_1, q_2]$ //

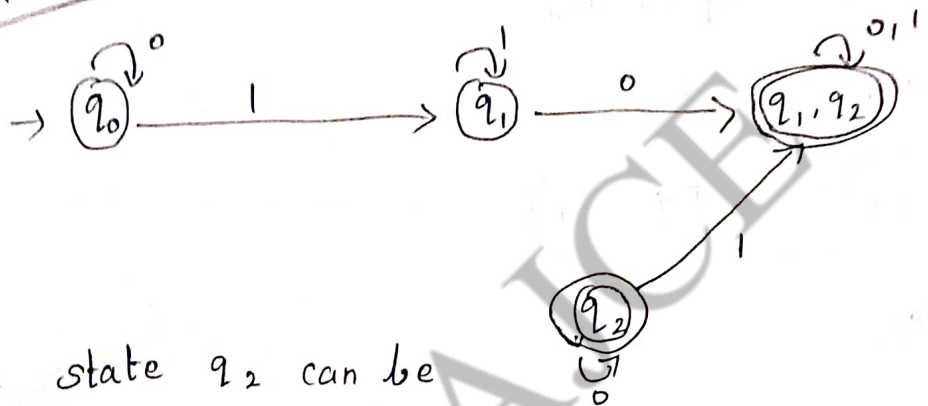
$\delta'(q_2, 0) = [q_2]$

$\delta'(q_2, 1) = [q_2, q_1]$

The state $[q_1, q_2]$ is the final state. bcoz it contains a final state q_2.

state	0	1
→ q_0	q_0	q_1
q_1	$[q_1, q_2]$	q_1
* q_2	q_2	$[q_1, q_2]$
* $[q_1, q_2]$	$[q_1, q_2]$	$[q_1, q_2]$

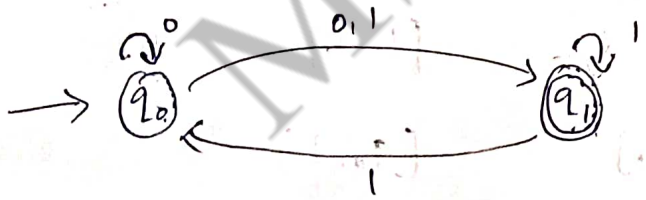
Transition Diagram:



The state q_2 can be eliminated bcoz q_2 is an unreachable state.

→

Ex:2



state	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_1\}$
* q_1	$\{\phi\}$	$\{q_0, q_1\}$

$\delta'(q_0, 0) = \{q_0, q_1\}$ → new state Generated.

$= [q_0, q_1]$

$\delta'(q_0, 1) = \{q_1\}$

$= [q_1]$

$$\delta'([q_0,q_1],0) = \delta'(q_0,0) \cup \delta'(q_1,0)$$
$$= \{q_0,q_1\} \cup \phi$$
$$= \{q_0,q_1\}$$
$$= [q_0,q_1]$$

$$\delta'([q_0,q_1],1) = \delta'(q_0,1) \cup \delta'(q_1,1)$$
$$= \{q_1\} \cup \{q_0,q_1\}$$
$$= \{q_0,q_1\}$$
$$= [q_0,q_1]$$

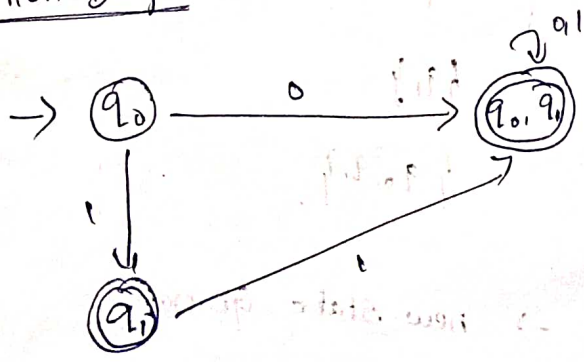
$$\delta'(q_1,0) = \phi$$

$$\delta'(q_1,1) = [q_0,q_1]$$

Transition Table:

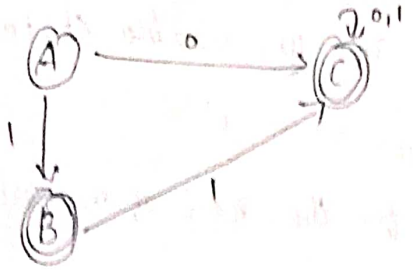
state	0	1
→ q_0	$[q_0,q_1]$	$[q_1]$
* q_1	ϕ	$[q_0,q_1]$
* $[q_0,q_1]$	$[q_0,q_1]$	$[q_0,q_1]$

Transition Diagram:



we can change the name of the states of DFA

$A = [q_0]$
$B = [q_1]$
$C = [q_0,q_1]$



→ Minimization of DFA:

* Minimization of DFA means reducing the number of states from given FA.

Steps:

Step 1: Remove all the states that are unreachable from the initial state

step 2: Draw the transition table for all pair of states.

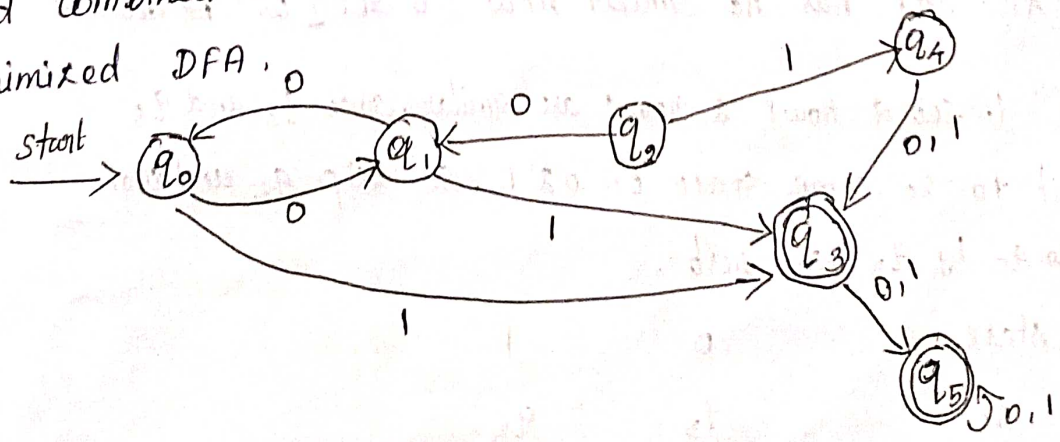
step 3: Now split the transition table into two tables T1 & T2. T1 contain all final states and T2 contains non-final states

step 4: Find similar rows and find the two states which have the same value and remove one of them.

step 5: Repeat step 3 until we find no similar rows available in the transition table T1.

step 6: Repeat step 3 & 4 for Table T2 also.

step 7: Now combine the reduced T1 & T2 tables. and combined transition table is the transition table of minimized DFA.



Soln:

Step 1: In the DFA q_2 & q_4 are the unreachable states so remove them (Again Draw Digram remove q_2 & q_4)

Step 2: Draw the transition table for the rest of the states

state	0	1
$\rightarrow q_0$	q_1	q_3
q_1	q_0	q_3
* q_3	q_5	q_5
* q_5	q_5	q_5

Step 3: Now Divide rows of transition table into two table

1. which start from non-final states

State	0	1
q_0	q_1	q_3
q_1	q_0	q_3

2, Another start from final states

state	0	1
q_3	q_5	q_5
q_5	q_5	q_5

Step 4: set1 has no similar row so set 1 will be the same

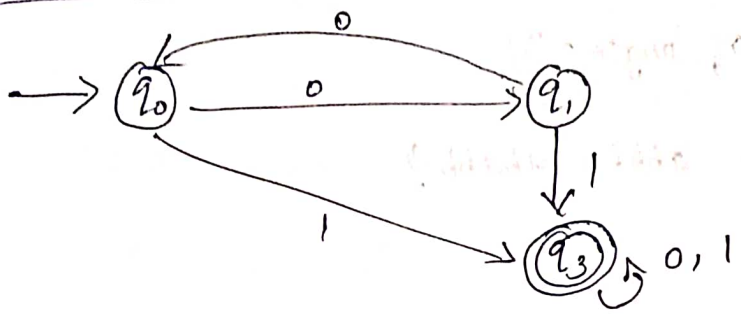
Step 5: In set 2 row1 & row2 are similar, since q_3 and q_5 transit to the same state on 0 & 1. So skip q_5 and then replace q_5 by q_3 in the rest.

State	0	1
q_3	q_3	q_3

Step 6: Now combine set 1 & set 2

State	0	1
→ q_0	q_1	q_3
q_1	q_0	q_3
* q_3	q_3	q_3

Minimized DFA:



→

Need For Automata Theory:

* Automata theory is also known as Theory of computation.

* It is the study of abstract machines and the computation problems that can be solved using these machines.

* The abstract machine is called the automata.

* The automation consists of states and transitions. The state is represented by circles, and the transition is represented by arrows.

* Automata is the kind of machine which takes some string as i/p & this i/p goes through a finite number of states and may enter in the final state.

Symbols:

It is denoted by Σ

Ex:-

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

$\Sigma = \{A, B, C, D\}$

$\Sigma = \{0, 1, 2\}$

$\Sigma = \{0, 1, \ldots 5\}$

Example :- 1

$L = \{$set of string of length 2$\}$

1) $\{a, aa, aaa, abb, abbb, ababb\}$

Infinite Language.

2) $\{aa, bb, ba, bb\}$

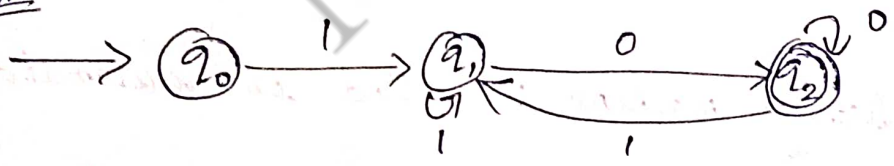
Finite Language.

→

Examples of DFA:

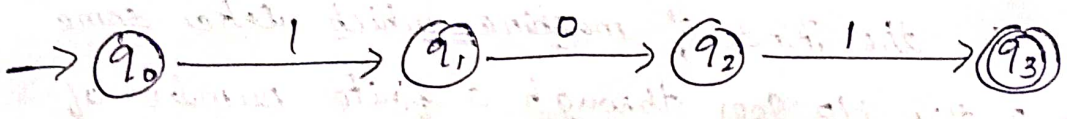
1) Design FA with $\Sigma = \{0, 1\}$ accepts those string which starts with 1 and end with 0

Soln:

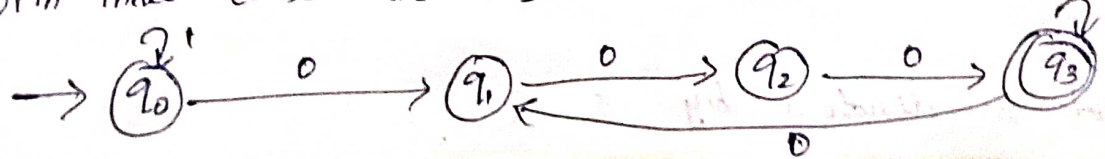


Example 2:

Design a FA with $\Sigma = \{0, 1\}$ accepts the only i/p 101

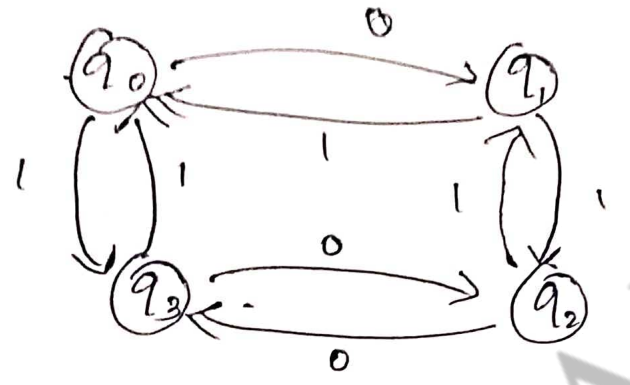


Ex:-3 Design a FA with $\Sigma = \{0, 1\}$ accepts the set of all strings with three consecutive 0's



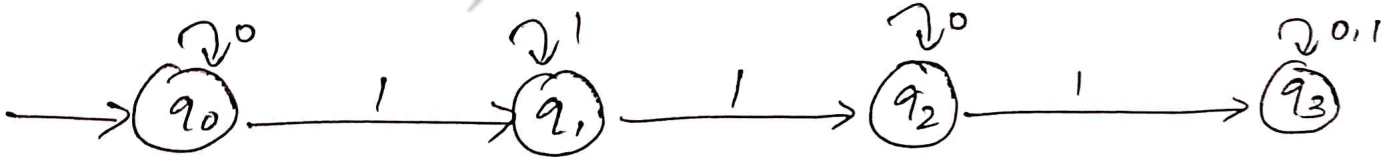
Ex:4 Design FA with $\Sigma = \{0, 1\}$ accepts even no. of 0's & even no. of 1's

Sol:



→

Ex5: Design a DFA $L(M) = \{ w \mid w \in \{0,1\}^* \}$ and w is a string that does not contain consecutive 1's



Pushdown Automata:

* A pushdown automation is a way to implement a context-free grammer in a similar way we design DFA for a R.G
* A DFA can remember a finite amount of info'
* A PDA can remember a infinite amount of info'

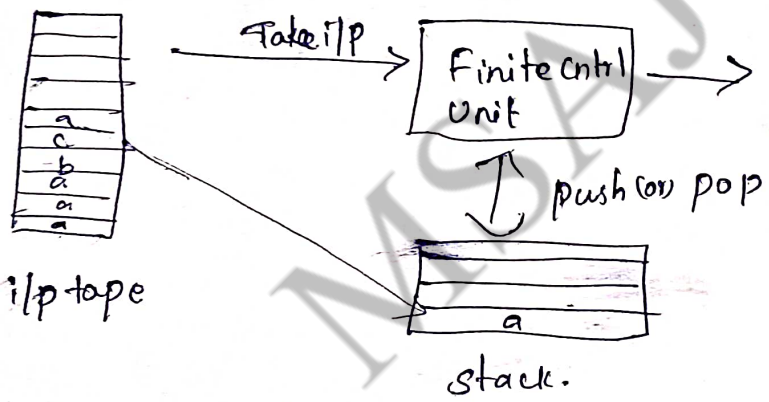
Basically a PDA is,

"Finite state Machine" + "a stack"

⇒ PDA has three components.

* an i/p tape * a cntrl unit * A stack with infinite size.

⇒ A PDA may (or) maynot read i/p symbol, but it has to read the top of the stack in every transaction.



⇒ A PDA can be formally described as 7-tuples $(Q, \Sigma, \delta, S, q_0, I, F)$

Q - is finite no of state

Σ - i/p alphabets

S - is a stack symbol

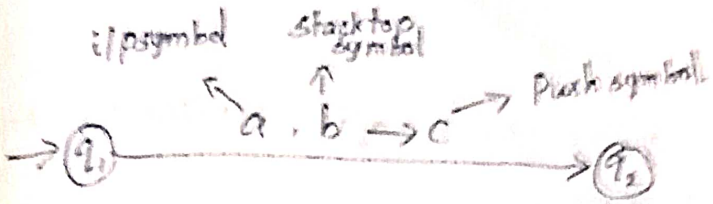
δ - is the transaction: $Q \times (\Sigma \cup \{\} \in \{\}) \times S \times Q \times S^*$

q_0 - is the Initial state $(q_0 \in Q)$

I - is the initial stack top symbol $(I \in S)$

F - is a set of accept state (or) Final state $(F \in Q)$

⇒ Transaction in a PDA from a state q_1 to state q_2 labled as $a, b \to c$



ie) at state q_1 if an i/p string 'a' is Encountered & top of symbol of stack is 'b' then we have to pop 'b' push 'c' on the top of the stack & move to state q_2

Terminologies Related to PDA:

Instantaneous Description (ID):

The instantaneous description (ID) of a PDA is represented by a triples (q, w, s) where

- → q is the state
- → w is unconsumed i/p
- → s is the stack contents

ID is an informal notation of how a PDA Compute an i/p string & make a decision that string is accepted (or) rejected.

Twinstile notation:

It is used for connecting pairs of ID's that represent one (or) many moves of a PDA. The process of transaction is denoted by the twinstile symbol "$\vdash$"

- $\vdash$ It represents one move
- $\vdash^*$ sign describes a sequence of moves

Ex: $(p, b, T) \vdash (q, w, a)$
↳ unconsumed Data.

while taking a transaction from p to q the i/p symbol 'b' is Consumed & top of the stack 'T' is represented

Twinstile notation.

PDA Acceptance:

Two different way

Final state Acceptability:

⇒ A PDA accepts a string, when after reading the entire string the PDA in final state.

⇒ from the starting state we can make moves the end up in a final state with any stack values

⇒ The stack values are irrelevent as long as we endup in a final

For a PDA $(Q, \Sigma, S, \delta, q_0, I, F)$ the lang' accepted by the set of final state F is:

$$L(PDA) = \{ w \mid (q_0, w, I) \xrightarrow{\text{ID (triples)}} \vdash^* (q, \epsilon, x), q \in F \}$$

w: unconsumed data; q should be in Final st

for any i/p string x

Empty stack acceptability:

Here a PDA accepts a string when after reading the entire string the PDA has empty its stack.

For PDA $(Q, \Sigma, S, \delta, q_0, I, F)$ the lang' accepted by empty stack

$$L(PDA) = \{ w \mid (q_0, w, I) \vdash^* (q, \epsilon, \epsilon), q \in Q \}$$

ϵ: Empty stack; q should be in any one of stat

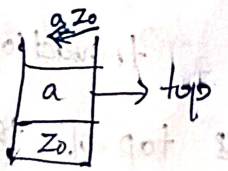
→

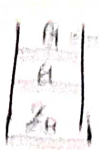
Ex: Construct PDA that accepts $L = \{ a^n b^n \mid n \geq 1 \}$

Soln: $a^n b^n$ = Every strings contains n. no of a's followed by n - no. of b's

If $n = 3$ $L = \{aaabbb\}$ $\boxed{Z_0}$ top

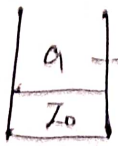
$\delta(q_0, a, Z_0) = (q_0, aZ_0)$



$\delta(q_0, a, a) = (q_0, aaZ_0)$ =  → top

$\delta(q_0, a, a) = (q_0, aaaZ_0)$ =  → top

$\delta(q_0, b, a) = (q_1, \epsilon)$ =  → top (ϵ → pop operation)

$\delta(q_0, b, a) = (q_1, \epsilon)$ =  → top

$\delta(q_1, b, a) = (q_1, \epsilon)$ =  → top

$\delta(q_1, \epsilon, Z_0) = (q_2, \epsilon)$

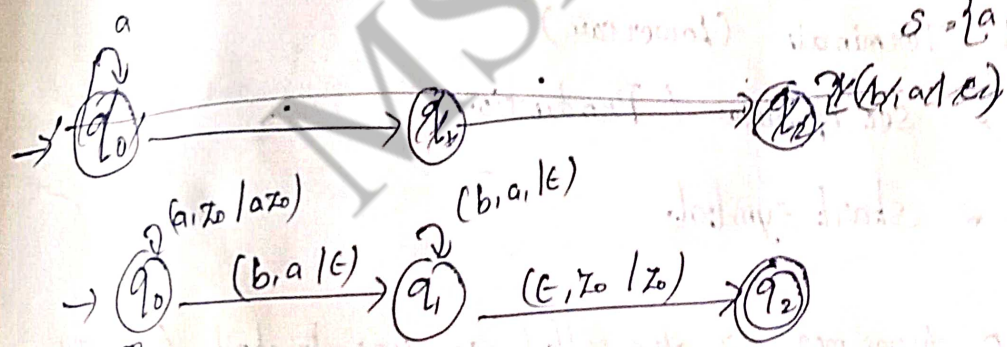
 Empty stack

$PDA = \{Q, \Sigma, S, \delta, q_0, T, F\}$

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{a, b\}$ $F = \{q_2\}$

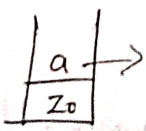
$S = \{a, Z_0\}$



Q) Give a PDA to accept the following language

$L = \{a^n b^{2n} / n \geq 1\}$ by final state.

{ abb , aa bbbb ... }

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$  → top

$\delta(q_0, a, a) = (q_0, aa)$ 

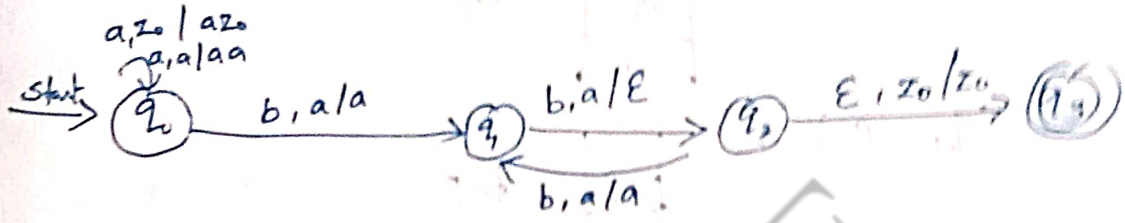
$\delta(q_0, b, a) = (q_1, a)$

$\delta(q_1, b, a) = (q_2, \varepsilon)$

$\delta(q_2, b, a) = (q_1, a)$

$\delta(q_2, \varepsilon, z_0) = (q_3, z_0)$

PDA, $P = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \{q_3\})$



→

Types of Grammer:

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

V = Non Terminals (Upper case)

T = Terminals (lower case)

P = Set of rules / Productions

S = start symbol.

1. Type 0 Grammer is also called as unrestricted Grammer
2. Type 1 Grammer " " " Context sensitive Grammer
3. Type 2 " " " Context free Grammer
4. Type 3 " " " Regular Grammer

Type 0 :: (URG)

* Recursively Enumerable lang
* It accepted by turing Machine.

$\alpha \longrightarrow \beta$

There is no restriction on α and β

$\alpha, \beta \in (V \cup T)^*$ any combination of Terminals and non-Terminals atleast one - non Terminal is present on the left hand side.

Ex: $aA \rightarrow bBcD$

$D \rightarrow F$

Type 1 (CSG)

* Context Sensitive lang'
* when the automata that accept context Sensitive lang' is Linear Bounded Automata.

$\alpha \rightarrow \beta \qquad \alpha, \beta \in (V \cup T)^*$

$|\alpha| \leq |\beta|$ [The length of α is less than (or) equal to the length of β]

[Left hand side we have Non Terminal]

$aA \rightarrow bBc$ (A → Non Terminal.)

$Aa \rightarrow b$ → This not accepted.

Type 2 (CFG)

* Context Free Lang'
* Context free Lang' is accepted by pushdown automata.

$\alpha \longrightarrow \beta$, $|\alpha| \leq |\beta|$, [$\alpha \in V$] $\therefore |\alpha| = 1$

($\alpha \in V$ → Non Terminal.)

Ex: $S = ab$

$S = abA$.

Type 3 (RG)

* Regular Language.
* Accepted by Finite Automata.

$\alpha \rightarrow \beta$

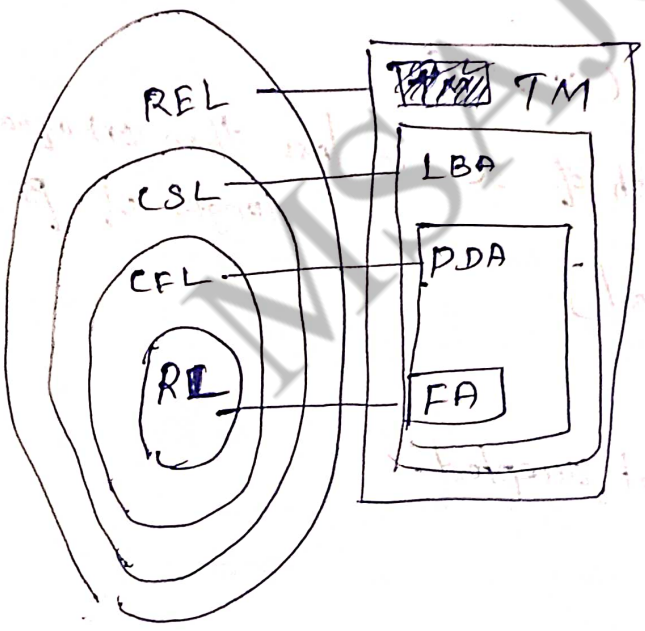
Left hand side α is Non-Terminal and the length is always 1 ($\because |\alpha| = 1$)

Right hand side is terminal followed by non terminal.

$S \rightarrow a \mid bA$ [$\because$ Right Linear Grammer]

Left linear grammer non terminal followed by terminal

$S \rightarrow a \mid Ab$



$\alpha \rightarrow \beta$

Type 0 : $\alpha \neq NULL$

Type 1 : $|\alpha| \leq |\beta|$

Type 2 : $A \in V$

Type 3 : $A \rightarrow aB$ or $A \rightarrow a$

1) $S \rightarrow AcaB$
$Bc \rightarrow acB$
$cB \rightarrow DB$
$aD \rightarrow Db$
} Type 1

2) $S \rightarrow xa$
$x \rightarrow a$
$x \rightarrow ax$
$x \rightarrow abc$
$x \rightarrow \epsilon$
} Type 2 (type 1.0 also).

3) $x \rightarrow \epsilon$
$x \rightarrow a \mid ay$
$Y \rightarrow b$
} Type 3

DERIVATION TREES:

* It is a graphical representation for the derivation of the gn production rules for a gn CFG
* It is also called parse tree

Properties of Derivation tree:

* The root node is always a node indicating start symbol.
* The derivation is read from left to right
* The leaf nodes are always terminal nodes
* The interior nodes are always the non-terminal nodes.

Ex: Construct the derivation tree for the string aabbabba from the CFG gn by $S \to aB / bA$

$A \to a / aS / bAA$

$B \to b / bS / aBB$

Soln: Left Most Derivation.

$S \underset{lm}{\Rightarrow} aB$

$S \underset{lm}{\Rightarrow} aaBB$

$S \underset{lm}{\Rightarrow} aabB$

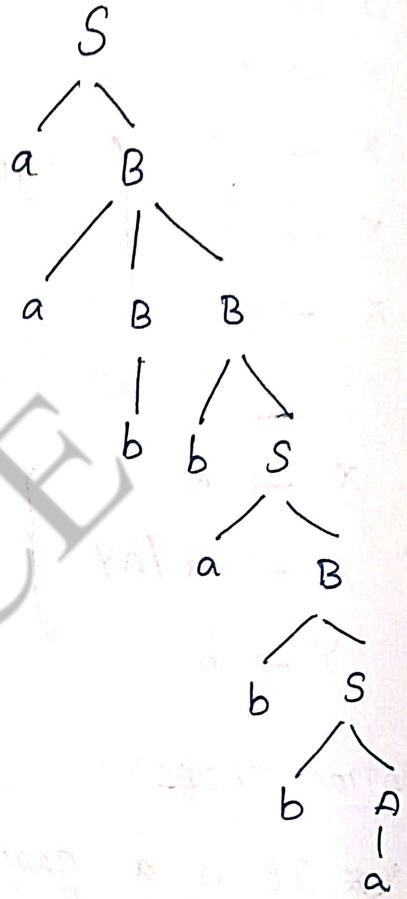
$S \underset{lm}{\Rightarrow} aabbS$

$S \underset{lm}{\Rightarrow} aabbaB$

$S \underset{lm}{\Rightarrow} aabbabS$

$S \underset{lm}{\Rightarrow} aabbabbA$

$S \underset{lm}{\Rightarrow} aabbabba$



Right Most Derivation

$S \underset{rm}{\Rightarrow} aB$

$S \underset{rm}{\Rightarrow} aaBB$

$S \underset{rm}{\Rightarrow} aaBbS$

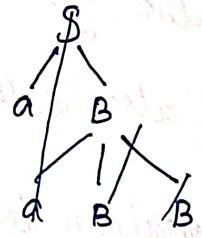
$S \underset{rm}{\Rightarrow} aaBbaA$

$S \underset{rm}{\Rightarrow} aaBbabS$

$S \underset{rm}{\Rightarrow} aaBbabbA$

$S \underset{rm}{\Rightarrow} aaBbabba$

$S \underset{rm}{\Rightarrow} aabbabba$



Right most Derivation :-

$S \underset{rm}{\rightarrow} aB$

$S \underset{rm}{\Rightarrow} aaBB$

$S \underset{rm}{\Rightarrow} aaBbS$

$S \underset{rm}{\Rightarrow} aaBbbA$

$S \underset{rm}{\Rightarrow} aaBbba$

$S \underset{rm}{\Rightarrow} aabSbba$

$S \underset{rm}{\Rightarrow} aabbAbba$

$S \underset{rm}{\Rightarrow} aabbabba$

→ 1) Consider the CFG for the lang' having any number of a's over the set of $\Sigma = \{a\}$

Soln:- $R.E = a^*$

$L = \{a, aa, aaa \ldots\}$

$S \rightarrow aS$

$S \rightarrow \in$

For Ex:- $\{aaaaa\}$

$S \rightarrow aS$

$S \rightarrow aaS$

$S \rightarrow aaaS$

$S \rightarrow aaaaS$

$S \rightarrow aaaaaS \quad (\because S \rightarrow \in)$

$S \rightarrow aaaaa$

→ 2) Construct the CFG for the lang' L which has all the strings which are all palindrome over $\Sigma = \{a, b\}$

$L = \{aba, aabbaa, baab \ldots\}$

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow a$

$S \rightarrow b$

$S \rightarrow \epsilon$

For ex:- abaaba

$S \rightarrow aSa$

$S \rightarrow abSba$

$S \rightarrow abaSaba \quad (\because S \rightarrow \epsilon)$

$S \rightarrow abaaba$

→ 3) Construct a CFG for the lang' $L = a^n b^{2n}$ where $n \geq 1$

Soln:- $L = \{ \underset{n=1}{abb}, \underset{n=2}{aabbbb}, \underset{n=3}{aaabbbbbb} \cdots \}$

The grammer could be 'aabbbb'

$S \rightarrow aSbb$

$S \rightarrow abb$

$S \rightarrow \epsilon$

$S \rightarrow aSbb$

$S \rightarrow aabbbb$

→ (4)

Ambiguous Grammer

A Grammer is said to be Ambiguous if there exists two or more derivation tree for a string (nxt page)

1) Check whether the following grammer is ambiguous or not $E \rightarrow E+E \mid E-E \mid E*E \mid a \mid \epsilon$. string is (a*a+a

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

Left derivation

$E \underset{lm}{\Rightarrow} E+E$

leftmost derivation 2

$E \underset{lm}{\Rightarrow} E*E$

$\Rightarrow_{lm}$ E * E + E

$\Rightarrow_{lm}$ a * E + E

$\Rightarrow_{lm}$ a * a + E

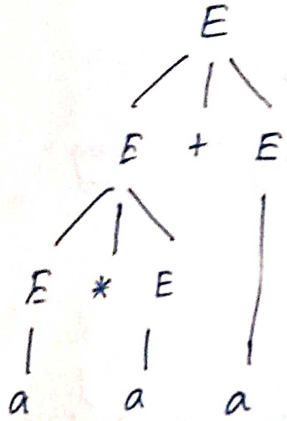
$\Rightarrow_{lm}$ a * a + a

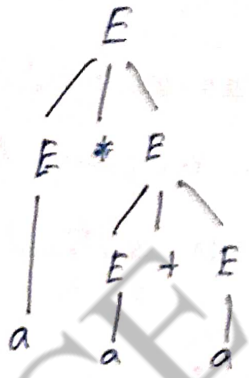
$\Rightarrow_{lm}$ a * E + E

$\Rightarrow_{lm}$ a * a + E

$\Rightarrow_{lm}$ a * a + a

Parse tree





Right Most Derivation 1

E $\Rightarrow_{rm}$ E + E

$\Rightarrow_{rm}$ E + a

$\Rightarrow_{rm}$ E * E + a

$\Rightarrow_{rm}$ E * a + a

$\Rightarrow_{rm}$ a * a + a

RMD 2

E $\Rightarrow_{rm}$ E * E

$\Rightarrow_{rm}$ E * E + E

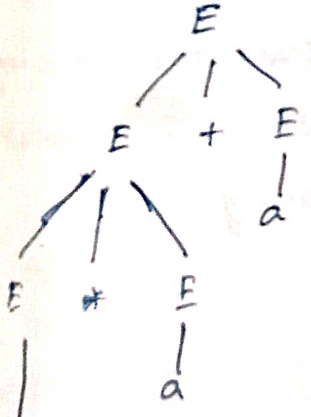
$\Rightarrow_{rm}$ E * E + a

$\Rightarrow_{rm}$ E * a + a

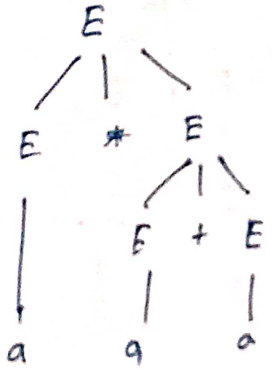
$\Rightarrow_{rm}$ a * a + a

Parse tree

RD 1



RD 2



* A grammer is said to be ambiguous if there exists more than one left most derivation (or) more than one right most derivation (or) more than one parse tree for gn i/p string

* If the grammer is not ambiguous, then we call unambig

* If the grammer 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 grammer without ambiguity.

→

Ex: let us consider a grammer with production rule

$E \rightarrow I$

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

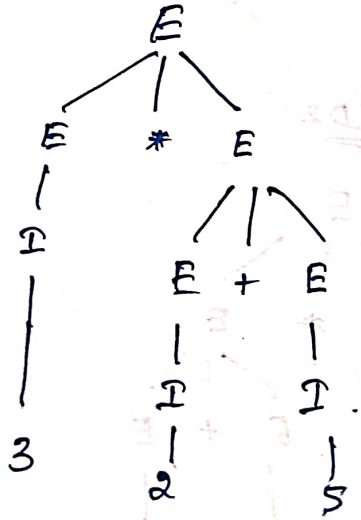
$I \rightarrow \epsilon \mid 0 \mid 1 \mid 2 \mid \ldots 9$

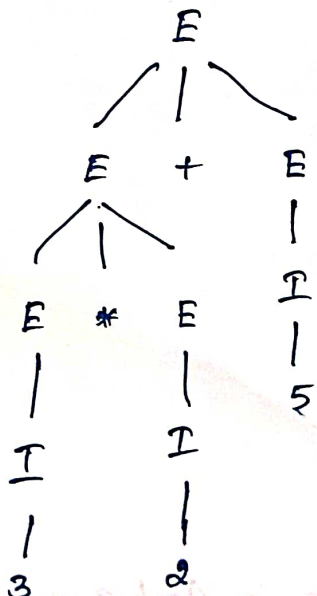
$G = \{V, T, P, S\}$

$V = \{I, E\}$

$T = \{+, *, (,), \epsilon, 0, 1, \ldots\}$

For ex. string 3 * 2 + 5 the above grammer can generate two parse tree by leftmost derivation.





Since these are two parse tree for sing string the grammer ambiguous.

Ex: Check whether the grammar is ambiguous or not

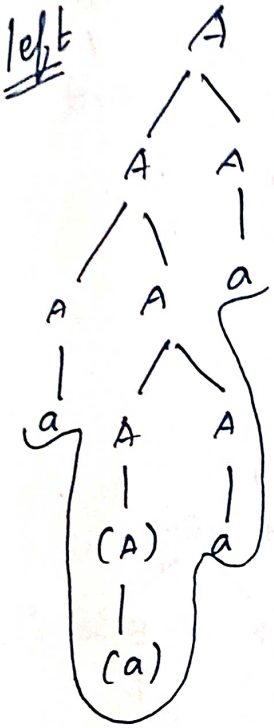
$A \rightarrow AA$

$A \rightarrow (A)$

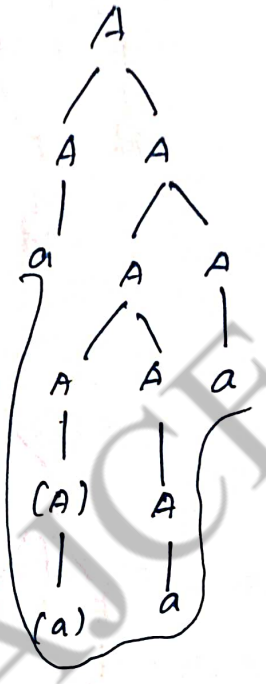
$A \rightarrow a$

for the string a(a)aa ; It can generate 2 parse tree

Left

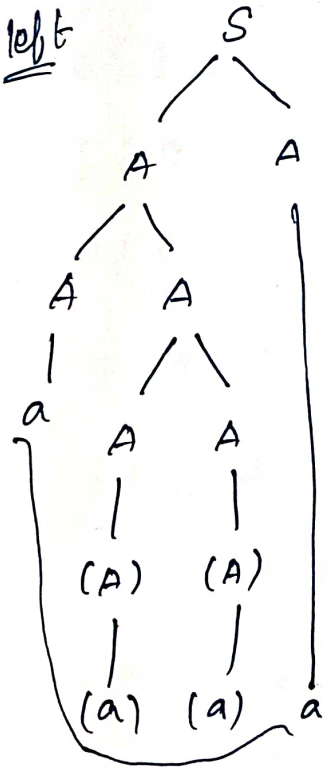


Right

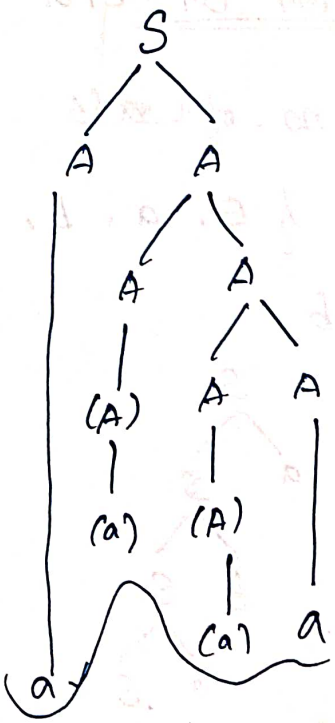


"a(a)(a)a"

Left



Right



For a grammar that can be produced diff' string and diff' that is called ambiguous.

Ex:- Check whether the gn grammer is ambiguous (or) not

$E \to E + E$

$E \to E - E$

$E \to id$

Soln:- "id + id - id"

1) LM

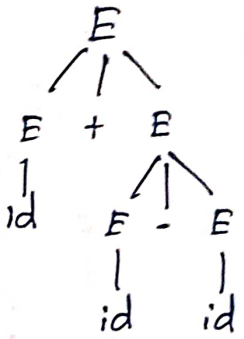
$E \to E + E$

$\Rightarrow id + E$

$\Rightarrow id + E - E$

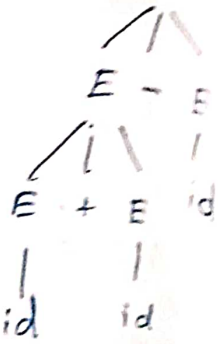
$\Rightarrow id + id - E$

$\Rightarrow id + id - id$



2) RM

$E \to E + E$



2)

$E \Rightarrow E - E$

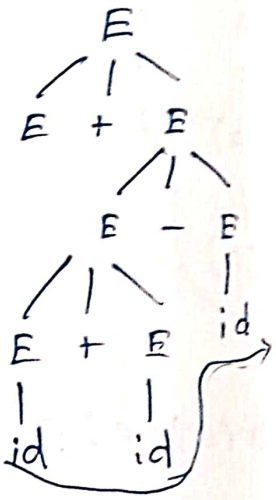
$\Rightarrow E + E - E$

$\Rightarrow id + E - E$

$\Rightarrow id + id - E$

$\Rightarrow id + id - id$

Parse tree:-



→ Grammer is ambiguous.

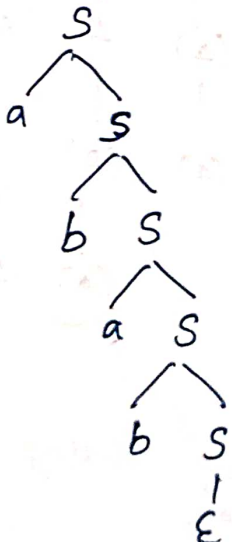
Construction of CFG

A) Any no. of a's and b's

Am:- $L = \{ \epsilon, a, b, aa, ab, bb, ba \ldots \}$

abab

$S \to aS / bS / \epsilon$



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

5) One occurance of 000

Ans: $S \rightarrow A000A$

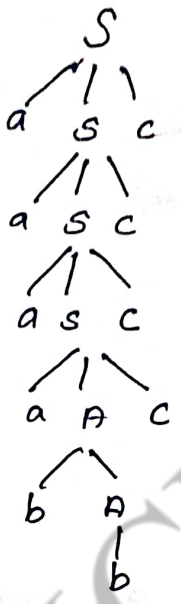
$A \rightarrow 0A \mid 1A \mid \varepsilon$

6) $L = \{ a^n b^m c^n \mid n, m \geq 1 \}$

Ans: $S \rightarrow aSc \mid aAc$

$A \rightarrow bA \mid b$

$n = 4 \quad m = 2$

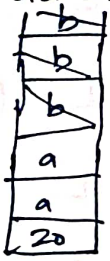


→ PDA

Construct PDA for $L = \{ a^n b^m c^m d^n \mid n \geq 1, m \geq 1 \}$

$n = 2 \quad m = 3$

aa bbb ccc dd



$\delta(q_0, a, Z_0) = (q_0, aZ_0)$

$\delta(q_0, a, a) = (q_0, aa)$

$\delta(q_0, b, a) = (q_1, ba)$

$\delta(q_1, b, b) = (q_1, bb)$

$\delta(q_1, c, b) = (q_2, \varepsilon)$

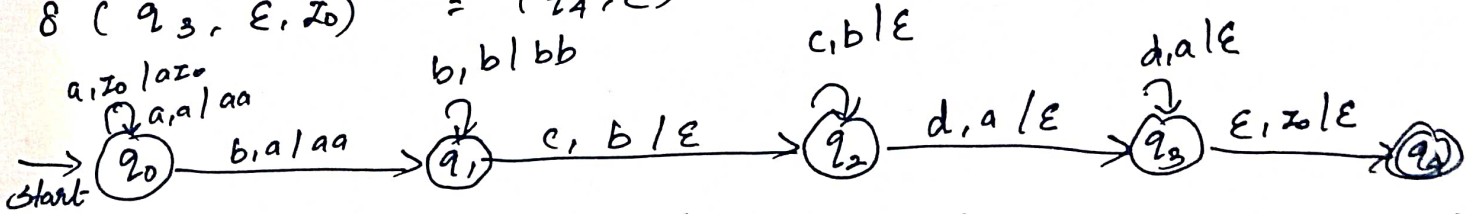
$\delta(q_2, c, b) = (q_2, \varepsilon)$

$\delta(q_2, c, b) = (q_2, \varepsilon)$

$\delta(q_2, d, a) = (q_3, \varepsilon)$

$\delta(q_3, d, a) = (q_3, \varepsilon)$

$\delta(q_3, \varepsilon, Z_0) = (q_4, \varepsilon)$



PDA, $P = (\{q_0 \ldots q_4\}, \{a, b, c, d\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_4\})$

Instantaneous Description (ID)

Check the Acceptance of string.

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

$L = \{ ab, aabb, aaabbb \cdots \}$

aaabbb

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$

$\delta(q_0, a, a) = (q_0, aaZ_0)$

$\delta(q_0,$

$(q_0, aaabbb, Z_0) \vdash (q_0, aabbb, aZ_0)$

$\vdash (q_0, abbb, aaZ_0)$

$\vdash (q_0, bbb, aaaZ_0)$

$\vdash (q_1, bb, aaZ_0)$

$\vdash (q_1, b, aZ_0)$

$\vdash (q_1, \varepsilon, Z_0)$

$\vdash (q_2, \varepsilon)$

Accepted.

Chomsky's Normal Form (CNF)

CNF stands for chomsky Normal form. A CFG is in CNF if all production rules satisfy one of the following conditions:

* start symbol generating ϵ

$$A \to \epsilon$$

* A non-terminal generating two non-terminals

$$S \to AB$$

* A non-terminal generating a terminal

$$S \to a$$

For Ex:

$G_1 =$ $S \to AB$, $S \to C$, $A \to a$, $B \to b$

$\left.\begin{matrix} S \to \epsilon \\ S \to AB \\ S \to a \end{matrix}\right\}$ Rule.

G_1 satisfy the rule specified for CNF, so it is in CNF.

$G_2 =$ $S \to aA$, $A \to a$, $B \to c$

G_2 doesnot satisfy the rules specified for CNF as $S \to aA$ contains terminal followed by non-terminal so G_2 is not a CNF.

Steps for converting CFG into CNF:-

Step 1: Eliminate start symbol from the RHS

If the start symbol S is at the right hand side of any production, create a new production as:

$$S_1 \to S$$

where S1 is the new start symbol

step 2: In the grammer, remove the null, unit and useless productions.

step 3: Eliminate terminals from the R.H.S of the production if they exists with other non-terminals (or) terminals

Eg: $S \to aA$ $\quad S \to RA$, $R \to a$ $\}$ $\to S \to aA$ can be decomposed

step 4: Eliminate R.H.S with more than two non-terminals

Eg: $S \to ABS$

$S \to RS$

$R \to AB$

Ex: Convert the given CFG to CNF, Consider the gn grammar as,

$S \to a \mid aA \mid B$

$A \to aBB \mid \epsilon$

$B \to Aa \mid b$

Soln:

Step 1: $S_1 \to S$ (New start symbol)

$S \to a \mid aA \mid B$

$A \to aBB \mid \epsilon$

$B \to Aa \mid b$

step 2:

$A \to \epsilon$

According to the rule only start symbol contain ϵ (Null) Production. but the grammar having ϵ (Null) production so, we have to remove ϵ.

$S1 \rightarrow S$

$S \rightarrow a \mid aA \mid B$

$A \rightarrow aBB$

$B \rightarrow Aa \mid b \mid \in$

$A \rightarrow \in$ ↓ replace a A means place one 'a' to the grammar.

whenever we have $\in$ value we replace A with $\in$ & write that terminal / non terminal.

Simplify CFG →
- Removal of $\in$ production
- Removal of unit production [one non terminal is deriving another non terminal $S \rightarrow S$]
- Removal of useless symbol

To eliminate unit production, replace S & B production

$S1 \rightarrow a \mid aA \mid \cancel{B} \mid \cancel{b} \mid Aa$ [B can be removed and by place B_value]

$S \rightarrow a \mid aA \mid \cancel{\in} Aa \mid b$

$A \rightarrow aBB$

$B \rightarrow Aa \mid b \mid a$

Step 3:

$S1 \rightarrow aA \mid Aa$

$S \rightarrow aA \mid Aa$ [These are not CNF]

$A \rightarrow aBB$

$B \rightarrow Aa$

$X \rightarrow a \Rightarrow S1 \rightarrow a \mid XA \mid Ax \mid b$

$S \rightarrow a \mid XA \mid Ax \mid b$

$A \rightarrow XBB$

$B \rightarrow Ax \mid b \mid a$

$X \rightarrow a.$

$S_0 \rightarrow a \mid XA \mid Ax \mid b$

$S \rightarrow a \mid XA \mid Ax \mid b$

$A \rightarrow RB$

$R \rightarrow XB$

$B \rightarrow AX \mid b \mid a$

$X \rightarrow a$

The grammar in CNF.

Greibach Normal form

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

* A start symbol generating ϵ

$S \rightarrow \epsilon$

* A non-terminal generating a terminal

$A \rightarrow a$

* A non-terminal generating a terminal which is followed by any number of non-terminals

$S \rightarrow aASB$

$S \rightarrow aAB$

Ex:

$G1 = S \rightarrow aAB \mid aB$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

G1 satisfy the rules specified for GNF. So grammar is in GNF

$G2 = S \rightarrow aAB \mid aB$

$A \rightarrow aA \mid \epsilon$

$B \rightarrow bB \mid \epsilon$

G2 does not satisfy the rules so not in GNF.

Only the start symbol has to generate G, so grammar is not in GNF

Steps for converting CFG into GNF

Step 1: Convert the grammar into CNF

If the gn grammar is not in CNF, convert it into CNF

Step 2: If the grammar exists left recursion, Eliminate it

Step 3: In the grammar, convert the given production rule into GNF form.

If any production rule in the grammar is not in GNF form we have to convert CNF.

Ex:

$S \rightarrow XB \mid AA$

$A \rightarrow a \mid SA$

$B \rightarrow b$

$X \rightarrow a$.

Soln: The gn grammar G is already in CNF & there is no left recu

The production rule $A \rightarrow SA$ is not in GNF,

So we substitute $S \rightarrow XB \mid AA$ in the production rule

$A \rightarrow SA$ as

$S \rightarrow XB \mid AA$

$A \rightarrow a \mid XBA \mid AAA$

$B \rightarrow b$

$X \rightarrow a$

The production rule $S \rightarrow XB$ & $B \rightarrow XBA$ is not in GNF

So we substitute $X \rightarrow a$ in the production rule as,

$S \rightarrow aB \mid AA$

$A \rightarrow a \mid aBA \mid AAA$

$B \rightarrow b$

$X \rightarrow a$

e will remove left recursion ($A \rightarrow AAA$) we get

$S \rightarrow aB \mid AA$

$A \rightarrow aC \mid aBAC \mid a \mid aBA$

$C \rightarrow AAC \mid AA$

$B \rightarrow b$

$X \rightarrow a$

$C \rightarrow AA$

[Introducing new Variable to remove left recursion one 'A' in the Production is replace with new Variable C without new variable] with variable / without variable
AAC / AA
AAC → will be replace

The production rule $S \rightarrow AA$ is not GNF

So we substitute $A \rightarrow aC \mid aBAC \mid a \mid aBA$ in $S \rightarrow AA$
$C \rightarrow AA$.

$S \rightarrow aB \mid aCA \mid aBACA \mid aA \mid aBAA$

$A \rightarrow aC \mid aBAC \mid a \mid aBA$

$C \rightarrow AAC$

$C \rightarrow aCA \mid aBACA \mid aA \mid aBAA$

$B \rightarrow b$

$X \rightarrow a$

The production rule $C \rightarrow AAC$ is not in GNF, so we substitute $A \rightarrow aC \mid aBAC \mid a \mid aBA$ in production rule $C \rightarrow AAC$ as :

$C \rightarrow aCAC \mid aBACAC \mid aAC \mid aBAAC$

$S \rightarrow aB \mid aCA \mid aBACA \mid aA \mid aBAA$

$A \rightarrow aC \mid aBAC \mid a \mid aBA$

$C \rightarrow aCA \mid aBACAC \mid aAC \mid aBAAC$

$C \rightarrow aCA \mid aBACA \mid aA \mid aBAA$

$B \rightarrow b$

$X \rightarrow a$ This is the GNF Grammar G1

Turing Machine

=> TM has infinite size tape and it is used to accept Recursive Enumerable languages

* TM can Move in both directions, and it also doesn't accept ∈

* If the string inserted in not in lang', machine will halt in non-final state.

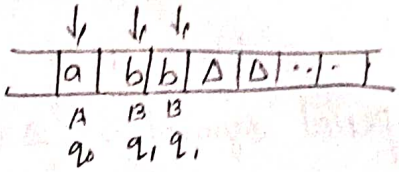
=> TM is a Mathematical model which consist of an infinite length tape divided into cells on which ip is given

=> It consist of the head which reads the input tape.

=> A state register stores the state of TM.

=> After reading an ip symbol, it is replaced with another symbol, to internal state is changed and it moves from one cell to the right or left.

> If the TM reaches the final states then the string accepted, otherwise rejected



ing Machine Formal Definition:

TM can be formally described as 7-tuples

Q, X, Σ, δ, q_0, B, F) where,

Q - finite set of states

X - Is the tape alphabets

Σ - Is the i/p alphabets

δ - Is the transition function

$\delta : Q \times X \rightarrow Q \times X \times \{\text{left shift, Right}\}$

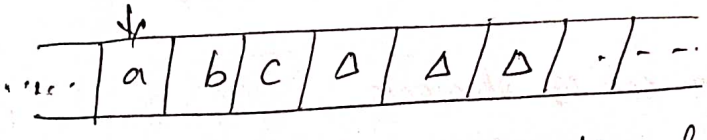
q_0 - Is the Initial state

B - Is the Blank symbol

F - Is the set of final states.

sic Model of TM:

The i/p tape is having an infinite no. of cells, each containing one i/p symbol. The Empty tape is filled blank characters



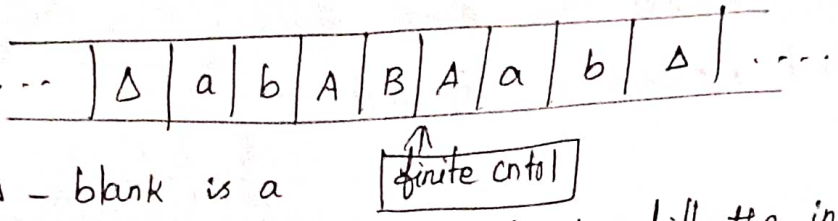
I/p tape

The finite control and the tape head which is responsible reading the current i/p symbol.

tape head can move to left to right

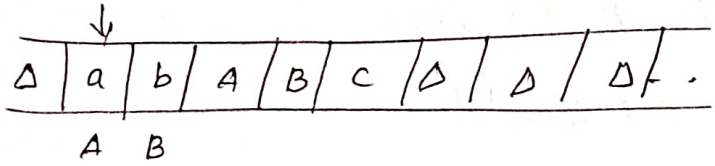
a finite set of states through which m/c has to undergo

Finite set of symbols called External symbols which are in building the logic of TM

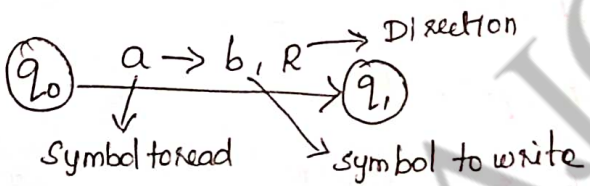


Δ - blank is a special symbol, $\Delta \notin \Sigma$ used to fill the infinite tape.

Operations on the tape:



* Read / scan the symbol and below tape head
* Update / write a symbol below tape head
* Move the tape head are step left
* Move the tape head are step right.



→

Languages accepted by Turing Machine

=> The TM accepts all the lang' Even though they are recursively Enumerable

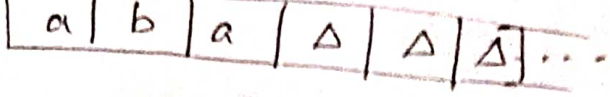
=> Recursive means repeating same set of rules for any no of times.

=> Enumerable means a set of elements

=> TM also accept the computable fun, such as addition, multiplication, subtraction, division, and many more.

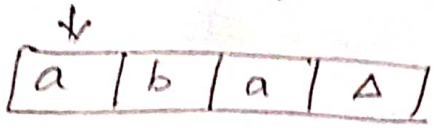
Eg: Construct a TM which accepts the lang of aba over $\Sigma = \{a, b\}$

Soln: we will assume that an i/p tape the string 'aba' is placed

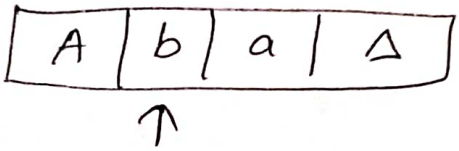


If the tape head is 'read out' 'aba' string then TM will halt after reading Δ

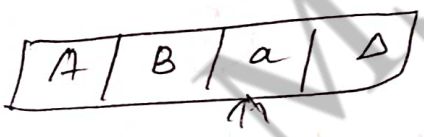
⇒ Initially state is q_0 & head point to 'a' as



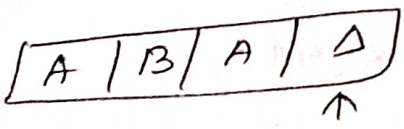
⇒ The move will be $\delta(q_0, a) = \delta(q_1, A, R)$ which means it will go to state q_1, replaced 'a' by 'A' & head will move to right as:



⇒ The move will be $\delta(q_1, b) = \delta[q_2, B, R)$ which means it will go to state q_2, replaced 'b' by 'B' & head will move to right as:



⇒ The move will be $\delta(q_2, a) = \delta(q_3, A, R)$ which means it will go to state q_3, replaced 'a' by 'A' and head will move to right as:

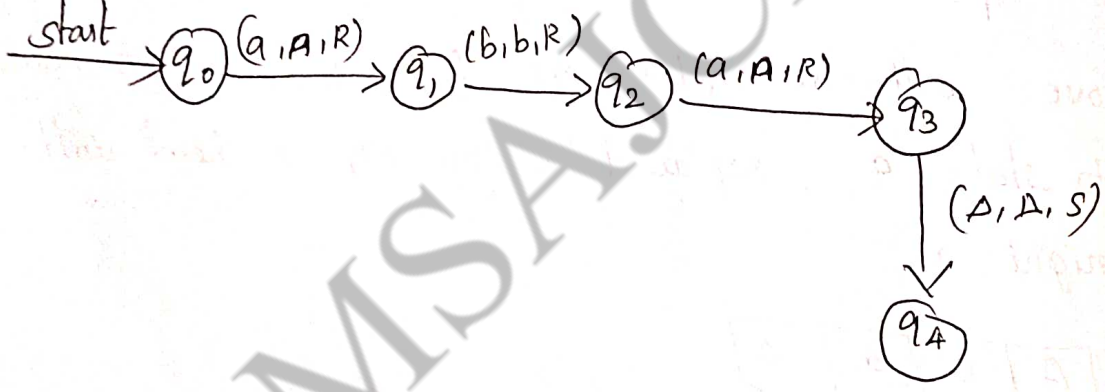


⇒ The move $\delta(q_3, A) = (q_4, \Delta, S)$ which means it will go to q_A which is HALT state which is accept state for any T.M

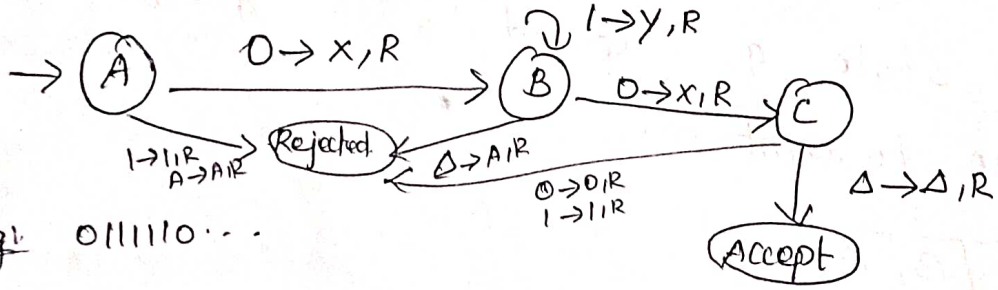
Representation of Transition Table:

states	a	b	Δ
q_0	(q_1, A, R)	-	-
q_1	-	(q_2, B, R)	-
q_2	(q_3, A, R)	-	-
q_3	-	-	(q_4, Δ, S)
q_4	-	-	-

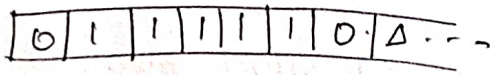
Transition Diagram:



Ex: Design a TM which recognize the lang' $L = 01^*0$

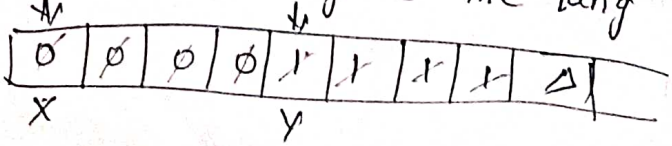


Eg: 0111110...



→

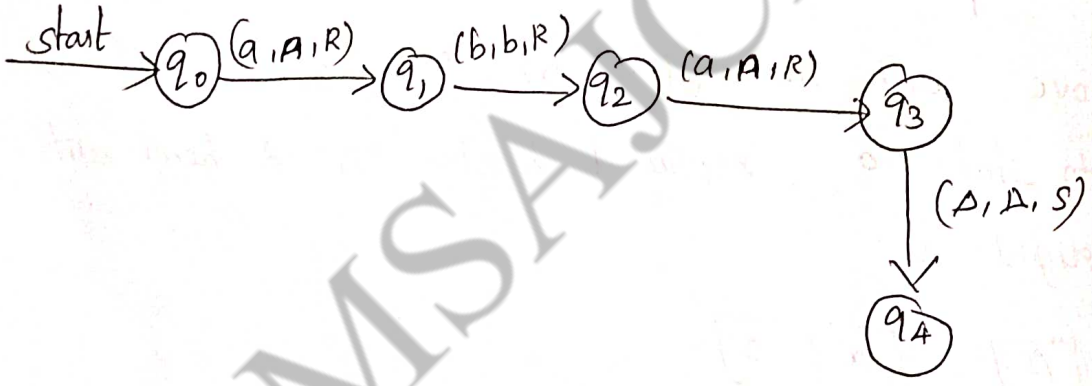
Ex: Design a TM which recognize the lang $L = 0^N 1^N$



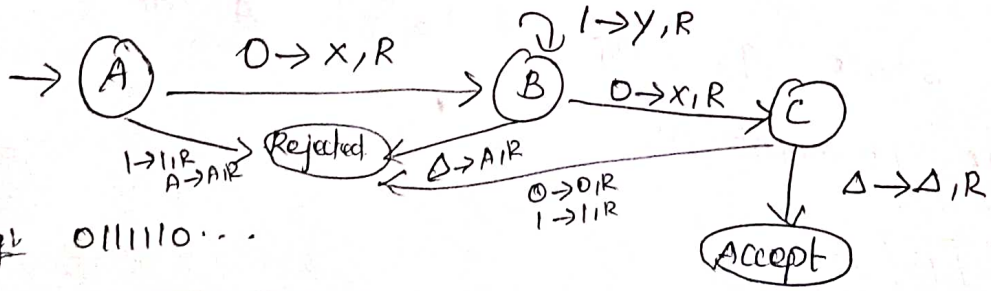
Representation of Transition Table:

states	a	b	Δ
q_0	(q_1, A, R)	-	-
q_1	-	(q_2, B, R)	-
q_2	(q_3, A, R)	-	-
q_3	-	-	(q_4, Δ, S)
q_4	-	-	-

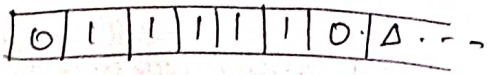
Transition Diagram:



Ex: Design a TM which recognize the lang' $L = 0\,1^*\,0$

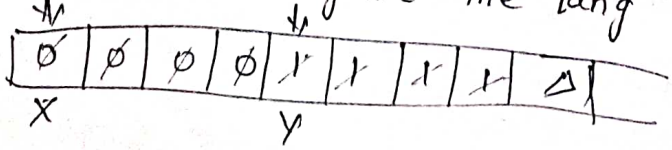


Eg: 0111110...



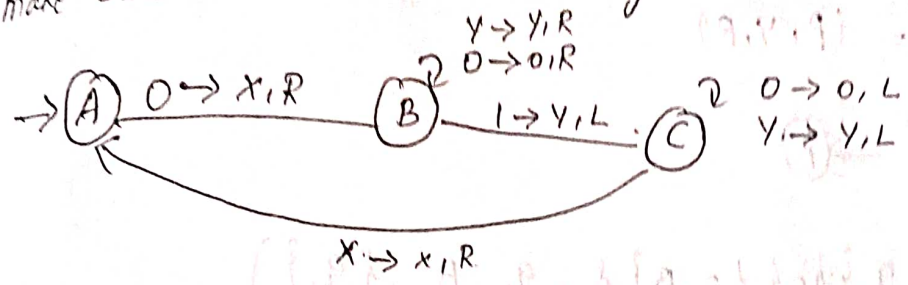
→

Ex: Design a TM which recognize the lang $L = 0^N 1^N$



Algorithm:

- change "0" to "x"
- Move Right to first "1" If none: Reject
- change "1" to "y"
- Move left to left most '0'
- Repeat the above steps until no more 0's
- make sure no more 1's remaining.



→

Instantaneous Description:

$X_1 X_2 \ldots X_{i-1}\ q\ X_i\ X_{i+1} \ldots X_n$

i) $\delta(q, X_i) = (P, Y, L)$

$X_1 X_2 \ldots X_{i-1}\ q\ X_i\ X_{i+1} \ldots X_n \vdash X_1 X_2 \ldots X_{i-2}\ P\ X_{i-1}\ Y\ X_{i+1} \ldots X_n$

ii) $\delta(q, X_i) = (P, Y, R)$

$X_1 X_2 \ldots X_{i-1}\ q\ X_i\ X_{i+1} \ldots X_n \vdash X_1 X_2 \ldots X_{i-1} Y P\ X_{i+1} \ldots X_n$

Let $M = (Q, \Sigma, F, \delta, q_0, B, F)$ be a TM

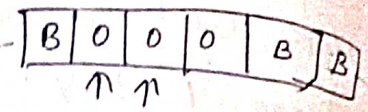
$L(M) = \{ w \text{ in } \Sigma^* \mid q_0 w \vdash^* \alpha P \beta \}$

α & β - Tape string

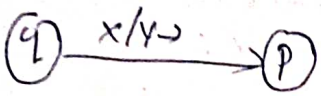
P - Final state.

1) Construct a TM for the fn $f(x) = x+1$

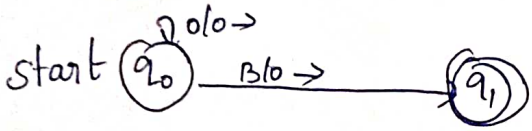
State	Tape symbols	
	0	B
q_0	$(q_0, 0, R)$	$(q_1, 0, R)$
q_1	–	–



$\delta(q, x) = (p, Y, R)$



TM, $M = (\{q_0, q_1\}, \{0\}, \{0, B\}, \delta, q_0, B, \{q_1\})$



$q_0 000 \vdash 0 q_0 00$B

$\vdash 00 q_0 0$B

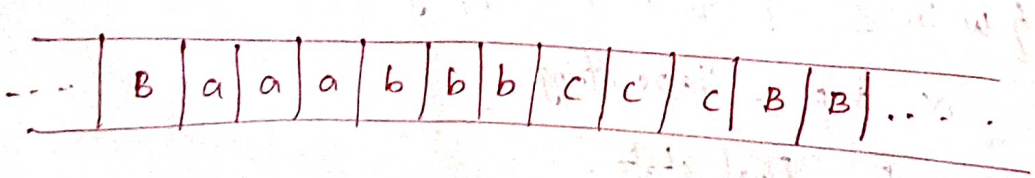
$\vdash 000 q_0$B

$\vdash 0000 q_1$B

q_1 is final state, TM halts, that accept the i/p string.

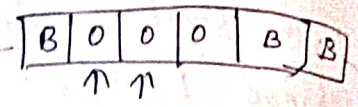
→ Construct a TM to accept $L = \{a^n b^n c^n \mid n \geq 1\}$

$L = \{abc, aabbcc, aaabbbccc, \ldots\}$

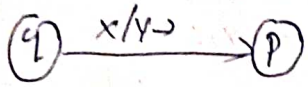


1) Construct a TM for the fn $f(x) = x+1$

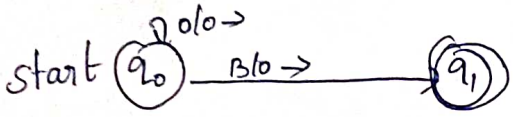
State	Tape symbols 0	B
q_0	$(q_0, 0, R)$	$(q_1, 0, R)$
q_1	–	–



$\delta(q, x) = (p, Y, R)$



TM, $M = (\{q_0, q_1\}, \{0\}, \{0, B\}, \delta, q_0, B, \{q_1\})$



$q_0000 \vdash 0q_000B$

$\vdash 00q_00B$

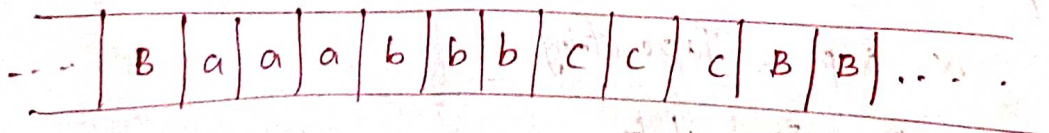
$\vdash 000q_0B$

$\vdash 0000q_1B$

q_1 is final state, TM halts, that accept the i/p string.

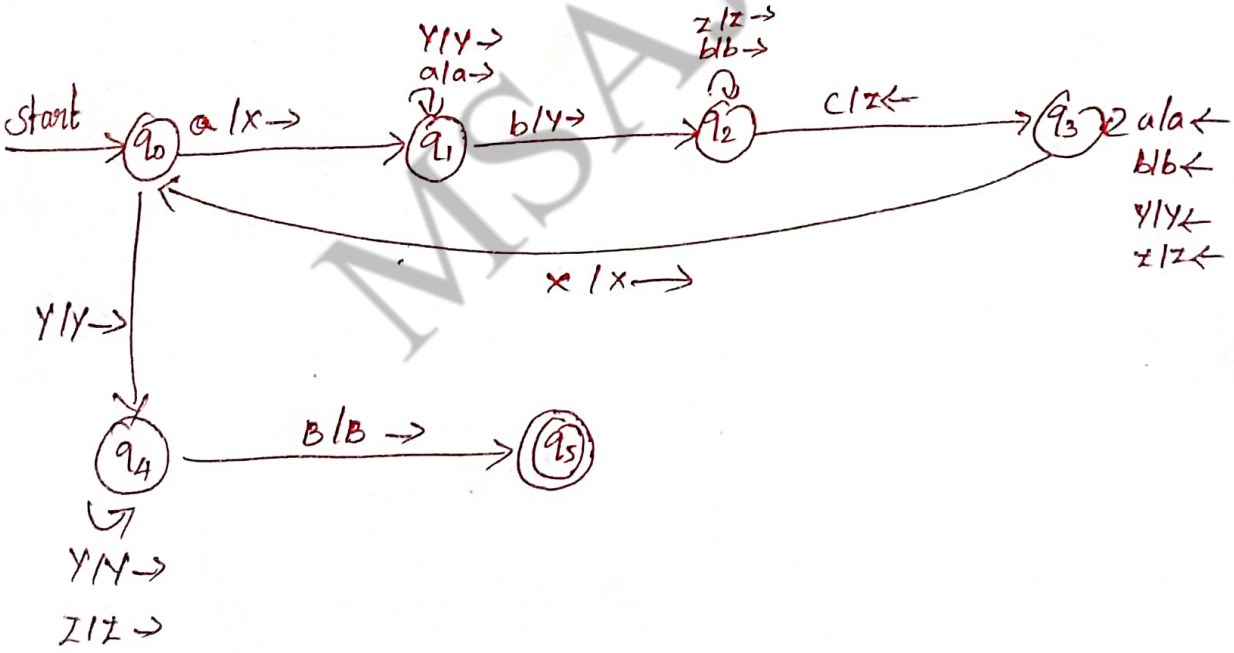
→ Construct a TM to accept $L = \{a^n b^n c^n \mid n \geq 1\}$

$L = \{abc, aabbcc, aaabbbccc, \ldots\}$



State	a	b	c	x	y	z	B
q_0	(q_1, X, R)	–	–	–	(q_4, Y, R)	–	–
q_1	(q_1, a, R)	(q_2, Y, R)	–	–	(q_1, Y, R)	–	–
q_2	–	(q_2, b, R)	(q_3, Z, L)	–	–	(q_2, Z, R)	–
q_3	(q_3, a, L)	(q_3, b, L)	–	(q_0, X, R)	(q_3, Y, L)	(q_3, Z, L)	–
q_4	–	–	–	–	(q_4, Y, R)	(q_4, Z, R)	$(q_5, B R)$
q_5	–	–	–	–	–	–	–

TM, $M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a, b, c\}, \{a, b, c, x, y, z, B\}, \delta, q_0, B, \{q_5\})$

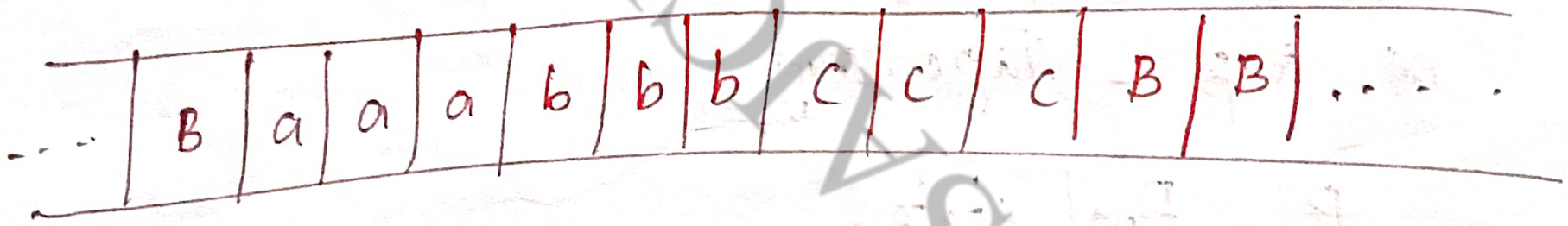


W = aa

q_1 b

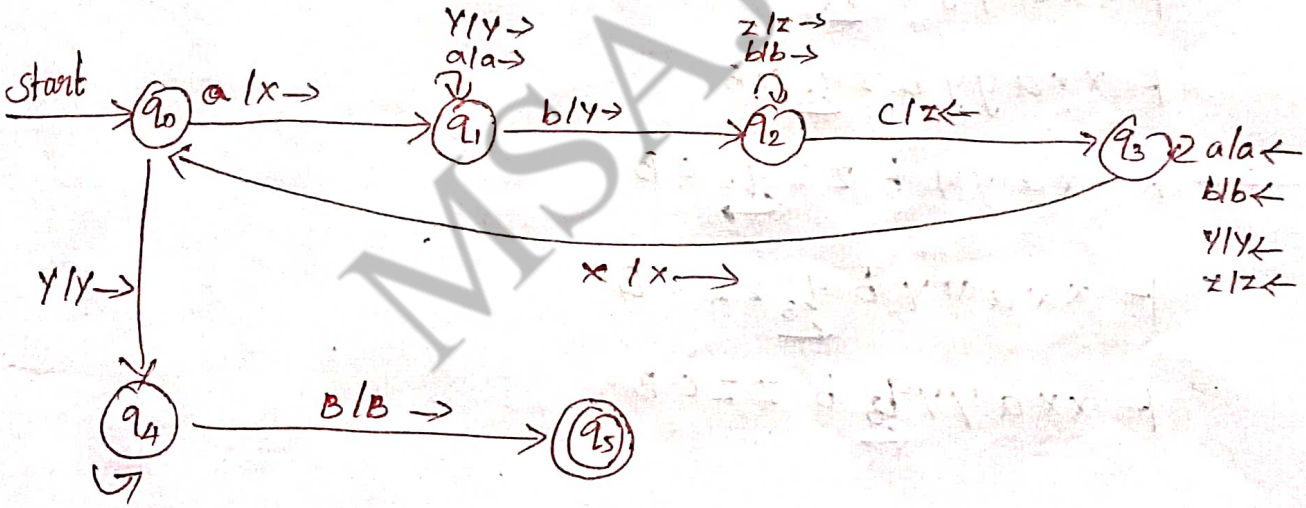
→ Construct a TM to accept $L = \{a^n b^n c^n \mid n \geq 1\}$

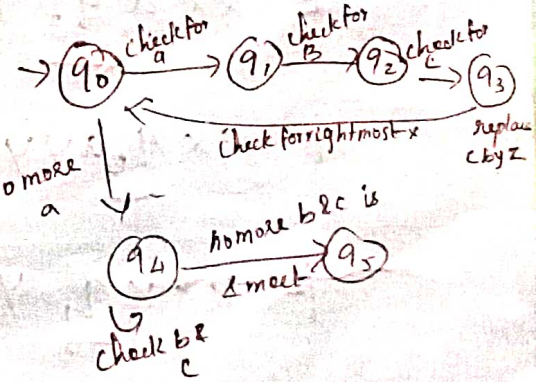
$L = \{abc, aabbcc, aaabbbccc, \ldots\}$



State	a	b	c	x	y	z	B
q_0	(q_1, X, R)	–	–	–	(q_4, Y, R)	–	–
q_1	(q_1, a, R)	(q_2, Y, R)	–	–	(q_1, Y, R)	–	–
q_2	–	(q_2, b, R)	(q_3, Y, L)	–	–	(q_2, Z, R)	–
q_3	(q_3, a, L)	(q_3, b, L)	–	(q_0, X, R)	(q_3, Y, L)	(q_3, Z, L)	–
q_4	–	–	–	–	(q_4, Y, R)	(q_4, Z, R)	(q_5, B, R)
q_5	–	–	–	–	–	–	–

TM, $M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a, b, c\}, \{a, b, c, X, Y, Z, B\}, \delta, q_0, B, \{q_5\})$





W = aaabbbccc

$\vdash \underset{\rightarrow}{q_0}$ aaabbbccc B

$\vdash$ X $\underset{\rightarrow}{q_1}$ aabbbccc B

$\vdash$ Xaa $\underset{\rightarrow}{q_1}$ bbbccc B

$\vdash$ Xaay $\underset{\rightarrow}{q_2}$ bbccB

$\vdash xaaYbbq_2CCB$

$\vdash xaaYbbZq_3CCB$

$\vdash xaaYbbq_3ZCCB$

$\vdash xaaYq_3bbZCCB$

$\vdash xaaq_3YbbZCCB$

$\vdash Xq_0aaYbbZCCB$

$\vdash XXq_1aYbbZCCB$

$\vdash XXaq_1YbbZCCB$

$\vdash XXaYq_1bbZCCB$

$\vdash XXaYYq_2bZCCB$

$\vdash XXaYYbq_2ZCCB$

$\vdash XXaYYbZq_2CCB$

$\vdash XXaYYbZZq_3CB$

$\vdash XXaYYbq_3ZZCB$

$\vdash XXaYYq_3bZZCB$

$\vdash XXaq_3YYbZZCB$

$\vdash XXq_0aYYbZZCB$

$\vdash XXXq_1YYbZZCB$

$\vdash XXXYYq_1bZZCB$

$\vdash XXXYYYq_2ZZCB$

$\vdash XXXYYYZZq_2CB$

$\vdash$ XXX YYY Z ZZ q_3 B

$\vdash$ XXX YYY Z Z q_3 Z B

$\vdash$ XXX YYY Z q_3 Z Z B

$\vdash$ XXX YYY q_3 Z Z Z B

$\vdash$ XXX YY q_3 Y Z Z Z B

$\vdash$ XXX Y q_3 Y Y Z Z Z B

$\vdash$ XXX q_6 Y Y Y Z Z Z B

$\vdash$ XX q_3 X Y Y Y Z Z Z B

$\vdash$ X q_3 X X Y Y Y Z Z Z B

$\vdash$ X X X Y Y q_4 Y Z Z Z B

$\vdash$ X X X Y q_4 Y Y Z Z Z B

$\vdash$ X X X q_4 Y Y Y Z Z Z B

$\vdash$ X X X Y Y Y q_4 Z Z Z B

$\vdash$ X X X Y Y Y Z q_4 Z Z B

$\vdash$ X X X Y Y Y Z Z q_4 Z B

$\vdash$ X X X Y Y Y Z Z Z q_4 B

$\vdash$ X X X Y Y Y Z Z Z B q_5

Since we have reach the final state.

$\therefore$ The given string is accepted.

→

1) Design a TM to accept a lang' $L = \{0^n 1^n \mid n \geq 1\}$

↳ (pg No: 215 to 216)

2) Design a TM for a lang $L = \{$The set of all string with equal no. of 0's & 1's$\}$. → (pg. No 219)

1) Soln:

The TM for the lang' has n no. of 0's & 1's

step2: the M moves towards if to find the right most 'Y' on moves M one cell right

step2.1:

If its a '0' the step 1 repeated

step2.2:

If its 'y' then go to step 3.

step 3:

It search for '1' in the remaining cells towards right. If 1 is found then string must be reflected else M can reach the final state upon reading 'B' & accept the string.

The required TM for gn lang is

$M = (Q, \Sigma, \vdash, \delta, q_0, B, F)$

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

$\Sigma = \{0, 1\}$

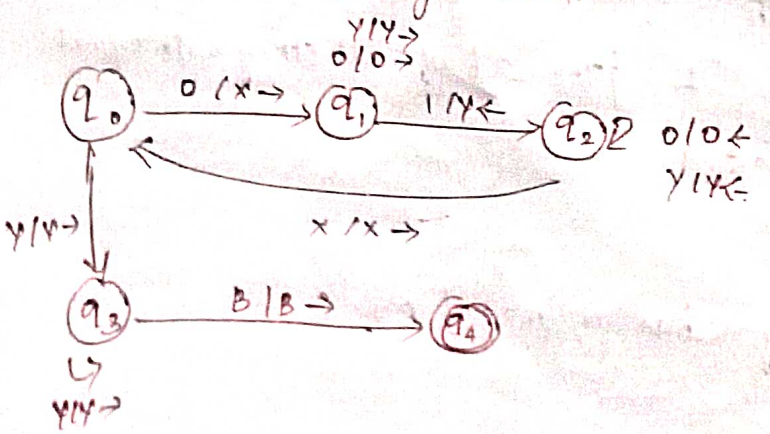
$\vdash = \{0, 1, X, Y, B\}$

$q_0 = \{q_0\}$

$B = \{B\}$

$F = \{q_4\}$

we take string 0011



$w = 0011$

$\vdash \underset{\rightarrow}{q_0}\ 0011B$

$\vdash X \underset{\rightarrow}{q_1} 011B$

$\vdash X0 \underset{\rightarrow}{q_1} 11B$

$\vdash X0Y \underset{\leftarrow}{q_2} 1B$

$\vdash X0 \underset{\leftarrow}{q_2} Y1B$

$\vdash X \underset{\rightarrow}{q_0} 0Y1B$

$\vdash XX \underset{\rightarrow}{q_1} Y1B$

$\vdash XXY \underset{\leftarrow}{q_2} YB$

$\vdash XX \underset{\leftarrow}{q_2} YYB$

$\vdash XX \underset{\rightarrow}{q_0} YYB$

$\vdash XXY \underset{\rightarrow}{q_3} YB$

$\vdash XXYY \underset{\rightarrow}{q_3} B$

$\vdash XXYYB \underset{\rightarrow}{q_4}$