Central Concept Of Automata Theory

Symbol : Symbol is an user defined entity which we want to consider as a part of language i.e, either symbols either letters or numbers etc.

Alphabets : It is the finite set of symbols, which is denoted by Σ (sigma).

Ex- $\Sigma(0,1)$ is known as binary alphabets.
$\Sigma(a,b)$ - Alphabet with set of letters
$\Sigma(0,1,2,3)$ - Alphabet with set of numbers
$\Sigma(0,1,b,+,-)$ - Alphabet with different type of symbol.

String : String is a finite set of alphabet or it is also known as sequence of symbol.

Ex: 0101 is a string from binary alphabet
ababa is a string from alphabet with letters.

Empty String : It is the string with 0 occurance of symbols.
→ It is denoted by 'ε'.

Power of Alphabet : If one alphabet is given then we can findout set of all strings of a certain length from that alphabet by using an exponential symbol i.e. Σ^K.

Ex: $\Sigma(0,1)$ given (alphabet)

$\Sigma^0 = (\ )$, ϵ length is 0

$\Sigma^1 = (0,1)$ length is 1

$\Sigma^2 = (00, 01, 10, 11)$ length is 2.

$\Sigma^3 = (000, 001, 100, 101, 010, 011, 111)$ length is 3.

$\Sigma^n = 2^n$

Length of String:

Length of the string is the no. of symbols present in the string and it is denoted by $|\omega|$.

→ ω denotes string

|| denotes length of string

Ex: $|0101| = 4$ $|abac| = 4$

$|011| = 2$ $|a| = 1$

Language: It is defined as a set of all strings over an alphabet.

→ It is denoted by Σ^*

Ex: $L = \Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 100, 101, 010, 011, 111, 110, 0000, - - - - - -\}$

If we express the language without ϵ then it will express in terms of Σ^+.

Σ^* - collection of all type of string with epsilone.

→ Σ^* is also known as the Kleene Closure and it always ∞.

→ Σ^+ is known as positive closure i.e, it is the collection of all strings without 'ϵ'.

$$\Sigma^* = \epsilon + \Sigma^+$$

$$\Sigma^+ = \Sigma^* - \epsilon$$

where ϵ = null string or empty string
It also denoted as λ.

Automation:

→ Automation is a system in which we transmit our input and it perform some function without participation of man.

→ Automation is of 2 types.

(I) If o/p depends only on i/p then it is known as automation without memory.
ex- automated door

(II) If o/p depends on i/p as well as the states then it is known as automation with memory.
ex- automated washing machin

o/p is only depends on i/ps as well as the states is also known as Moore Machine

o/p depends on i/ps as well as states and i/p at any instance of time is known as Mealy Machine.
ex: automated washing mach

Finite Automata :

→ It is a mathematical model or a machine in which consist of O/p and i/p.

→ It is the collection of states.

→ Finite automata is always represented by the transition diagram.

→ Transition diagram (Table / function) is the finite directed graph in which each node ~~and~~ or and vertex will represent as a state and the directed edges indicate that transition function of that state and the edges are always levelled with inputs.

→ In this model the final state is always represent by double circle. [◎]

→ The initial state always represented by single arrow symbol with string start [start]

Single circle used for normal set or non-finite set.

A finite automata can be defined as five no. of tuples.

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

Q: set of all states

Σ: inputs

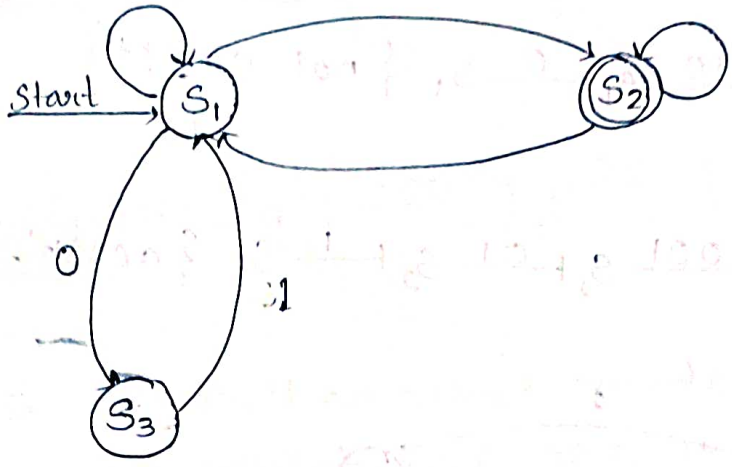
q_0: initial states

F: Final states

δ: transition functions.

4.1.23

Ex:



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

$Q = S_1, S_2, S_3$

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

$q_0 = S_1$

$F = S_2$

$\delta =$

	0	1
S_1	$\{S_1, S_3\}$	S_2
S_2	S_2	S_1
S_3	Null	S_1

Acceptibelity of string by using Finite Automata

$110 = \delta(S_1, 110) = \delta(S_2, 10) = \delta(S_1, 0) = S_1$ {not accept}

$1001 = \delta(S_1, 11001) = \delta(S_2, 1001) = \delta(S_1, 001) =$

$\delta(S_1, 01) = \delta(S_1, 1) = S_2$ {accept}

Otherwise :

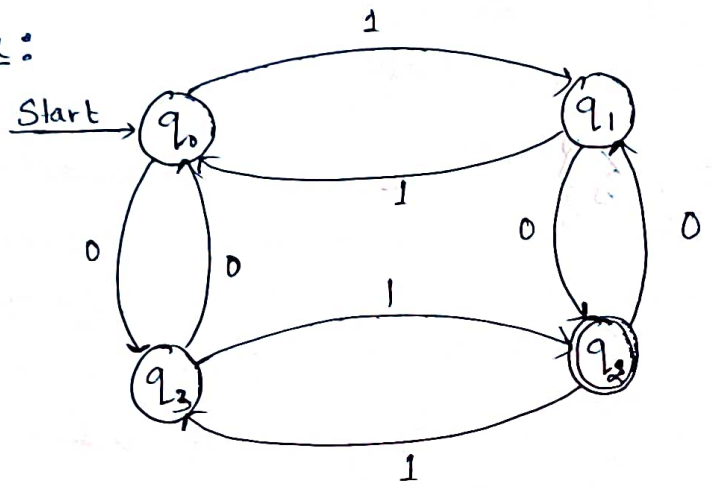
110

$S_1 \vdash^{110} S_2 \vdash^{10} S_1 \vdash^{0} S_1$ { not accept }

11001

$S_1 \vdash^{1001} S_2 \vdash^{001} S_2 \vdash^{01} S_2 \vdash^{1} S_1$ { accept }

Ex :



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

$Q = q_0, q_1, q_2, q_3$

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

$q_0 = q_0$

$F = q_2$

δ :

	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_1	q_3
q_3	q_0	q_2

→ 1100

$q_0 \xrightarrow{1100} q_1 \xrightarrow{100} q_0 \xrightarrow{00} q_3 \xrightarrow{0} q_0$ {not accept

→ 11100

$q_0 \xrightarrow{11100} q_1 \xrightarrow{1100} q_0 \xrightarrow{100} q_1 \xrightarrow{00} q_2 \xrightarrow{0}$

DFA : It is a mathematical model or a machine in which we are using 5 no. of tuples

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

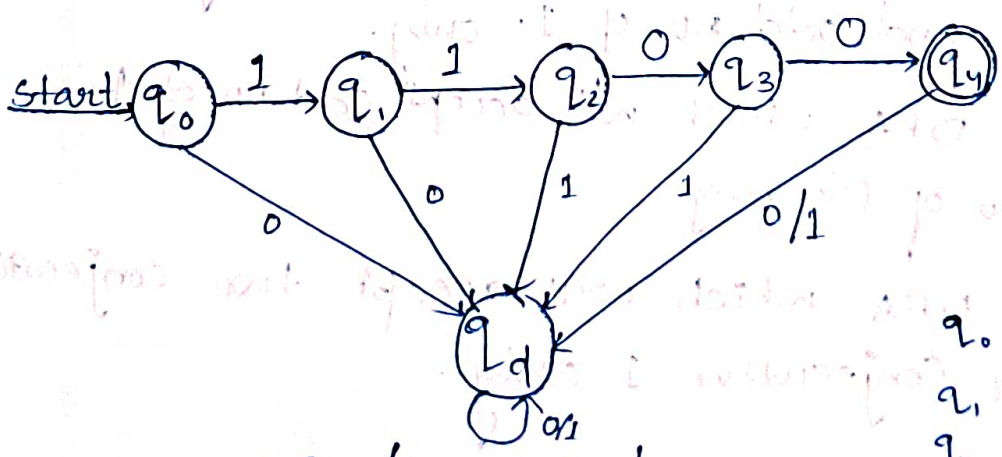
where $F \subseteq Q$ i.e, in DFA we can consider final state more than one.

The basic rules for DFA :

Rule 1 : Each state does not contain more than one transition for each input.

Rule 2 : Each and every state must use all the inputs.

Q) Design a DFA which accept a string 1100 only.

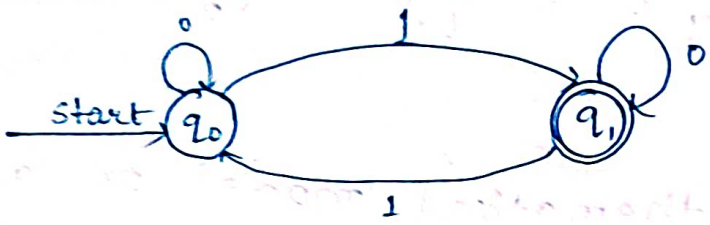


q_d : dead state / trap state / dummy state

	0	1
q_0	q_d	q_1
q_1	q_d	q_2
q_2	q_3	q_d
q_3	q_4	q_d
q_4	q_d	q_d

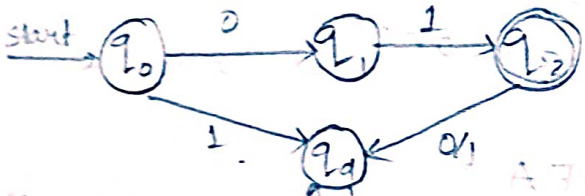
Q) Design a DFA which accept the string with odd no. of 1 over an alphabet {0,1}

000110001111000

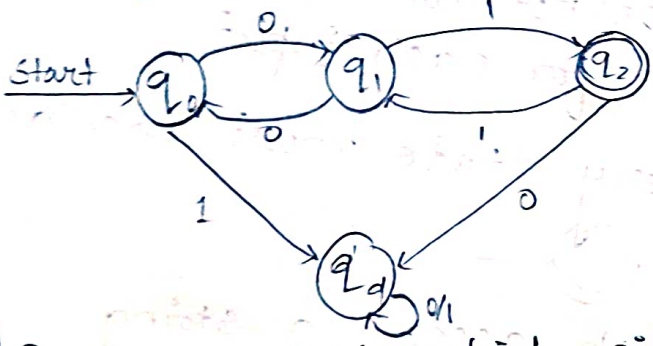


Q) Design a DFA which accept set of all string contains odd no. of 0's and odd no. of 1s only

01



000111



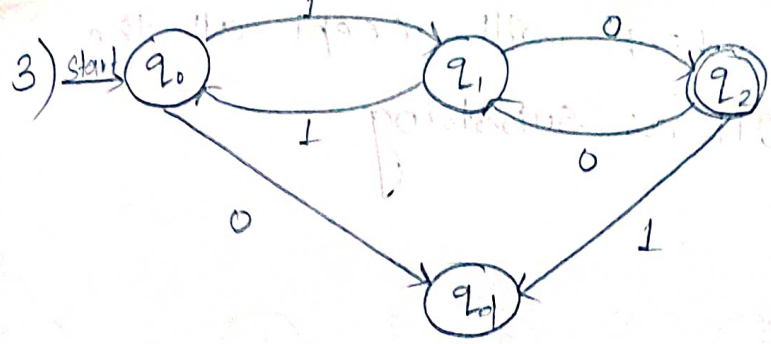
Q) Design a DFA which will accept the string 1010 only.

Q) Design a DFA which will accept the string odd no. of 0's and odd no. of 1's only.

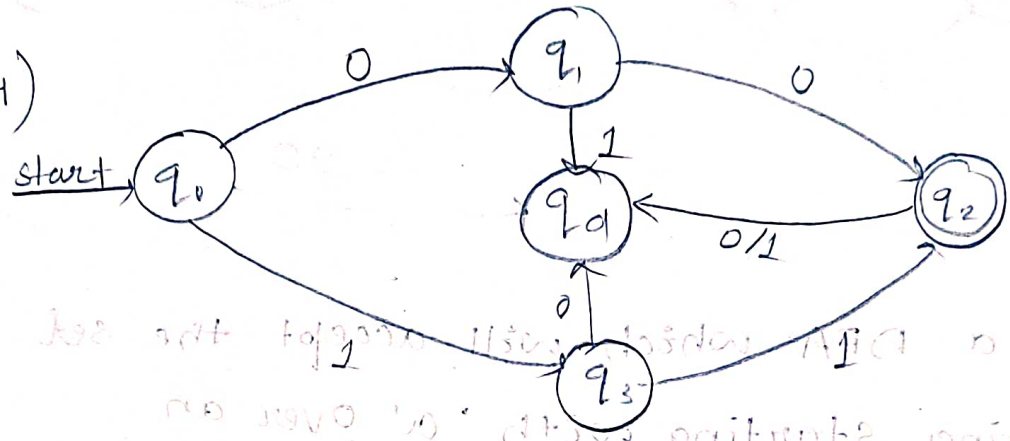
Q3) Design a DFA which will accept odd no. of 1's and odd no. of 0's only.

Q4) Design a DFA which will accept two consecutive 0's or two consecutive 1 only.

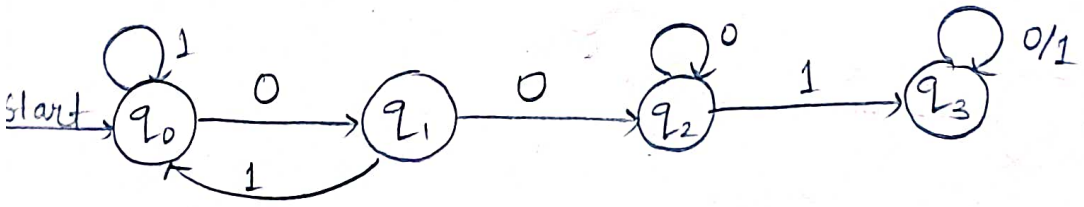
3)



4)



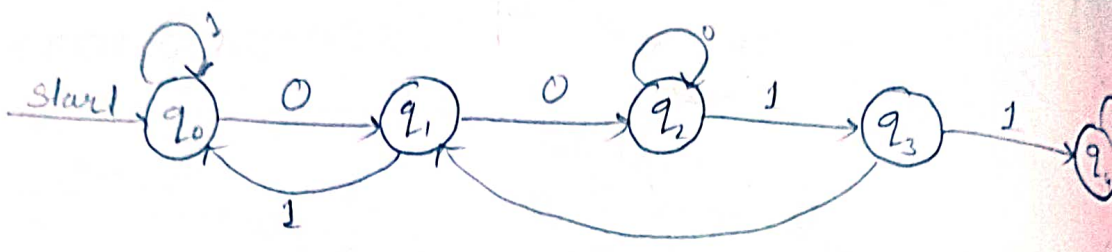
Q) Design a DFA which will accept all the string that contain 001 as the sub string



111001
1001
0001
00001
0011
0010
01001
000111

001000
0011111
001001

Q) Design a DFA which will accept all the string with 0011 as substring

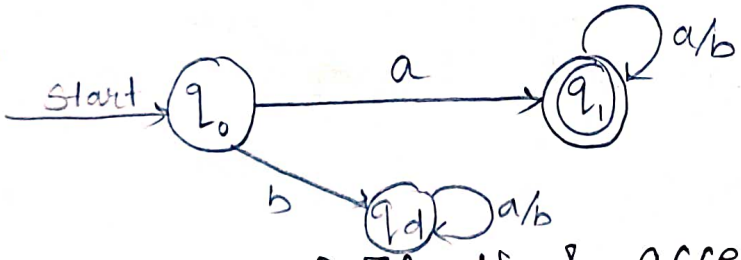


00

Q1) Design a DFA which will accept the set of string starting with 'a' over an alphabet: (a, b).

Q) Design a DFA which will accept the set of string ending with 'a' over an alphabet, Σ:(a, b).

1.

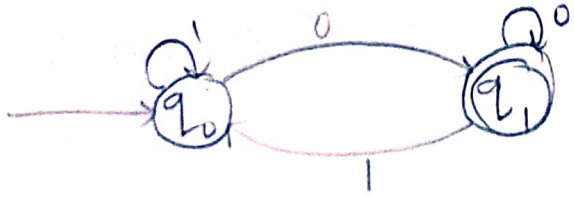


3) Design a DFA that accept all the even integers in the form of binary string.

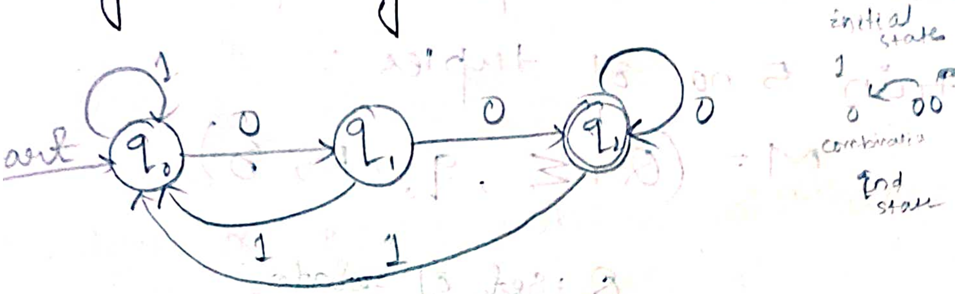
4) Design a DFA that will accept all the odd integers in the form of binary string.

0 2 4 6 8 10

0 10 100 110 1000 1010



Design a DFA which accept set of all strings ending with '00'.



NFA (Nond-

It is the also contains input, output & states

→ It contain 5 no. of tuples :

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

Q : set of states

Σ = input alphabet

q_0 = initial state.

F : It is a final state of set.

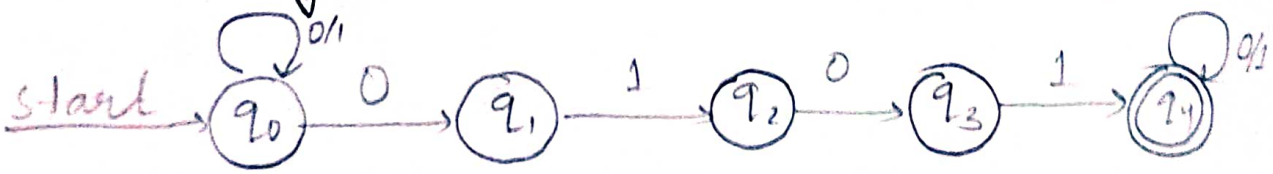
δ = transition func^n.

Basic rules for NFA :

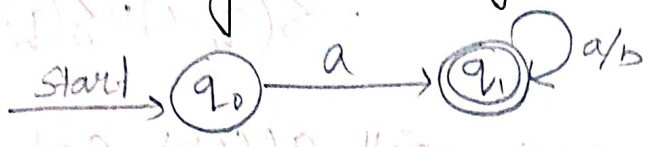
1. Each state contain more than one transition for the i/p.
2. It is not compulsory that each state will use all the inputs.

Q// Design a NFA which will accept the string 0101 as string only.

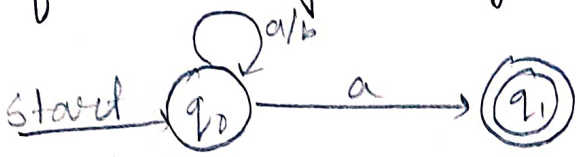
Q// Design a NFA which will accept set of all string contain 0101 as a substring.



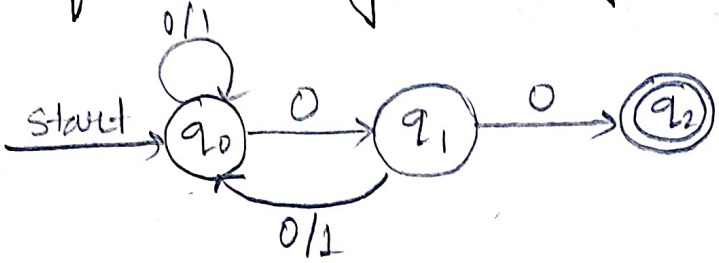
Q// Design a NFA which will accept set of all string starting with 'a'.



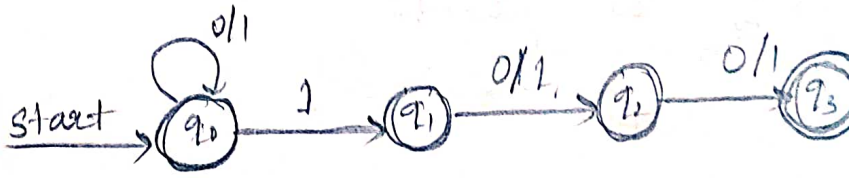
Q// Design a NFA which will accept se of all string ending with 'a'.



Q// Set of all strings ending will 00.



Q// Set of all strings containing 3rd symbo from right hand side is 1. 01



Q// Convertion of DFA from NFA :

Let M is a NFA i.e,

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

hen its equivalent DFA is $M' = \{Q', \Sigma, q_0'$

then, Q' = Set of states and it always represent in form of 2^Q

q_0' = initial state of DFA

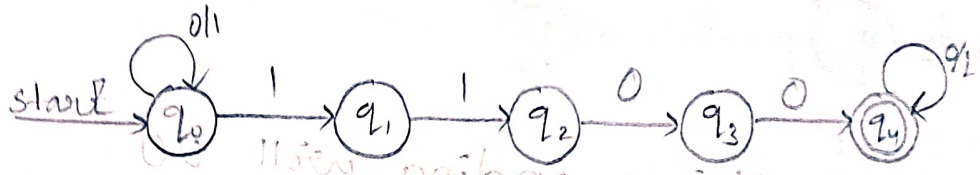
F' = subset of all subset of Q whi

are having that state of the given NFA.

δ = always represent $\delta'((q_0, q_1, q_2 \ldots$

$= \delta(q_0, 0) \cup \delta(q_1,$

1) Design a NFA which will accept set all string contain a 1100 as a substring then convert that NFA to a DFA.



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

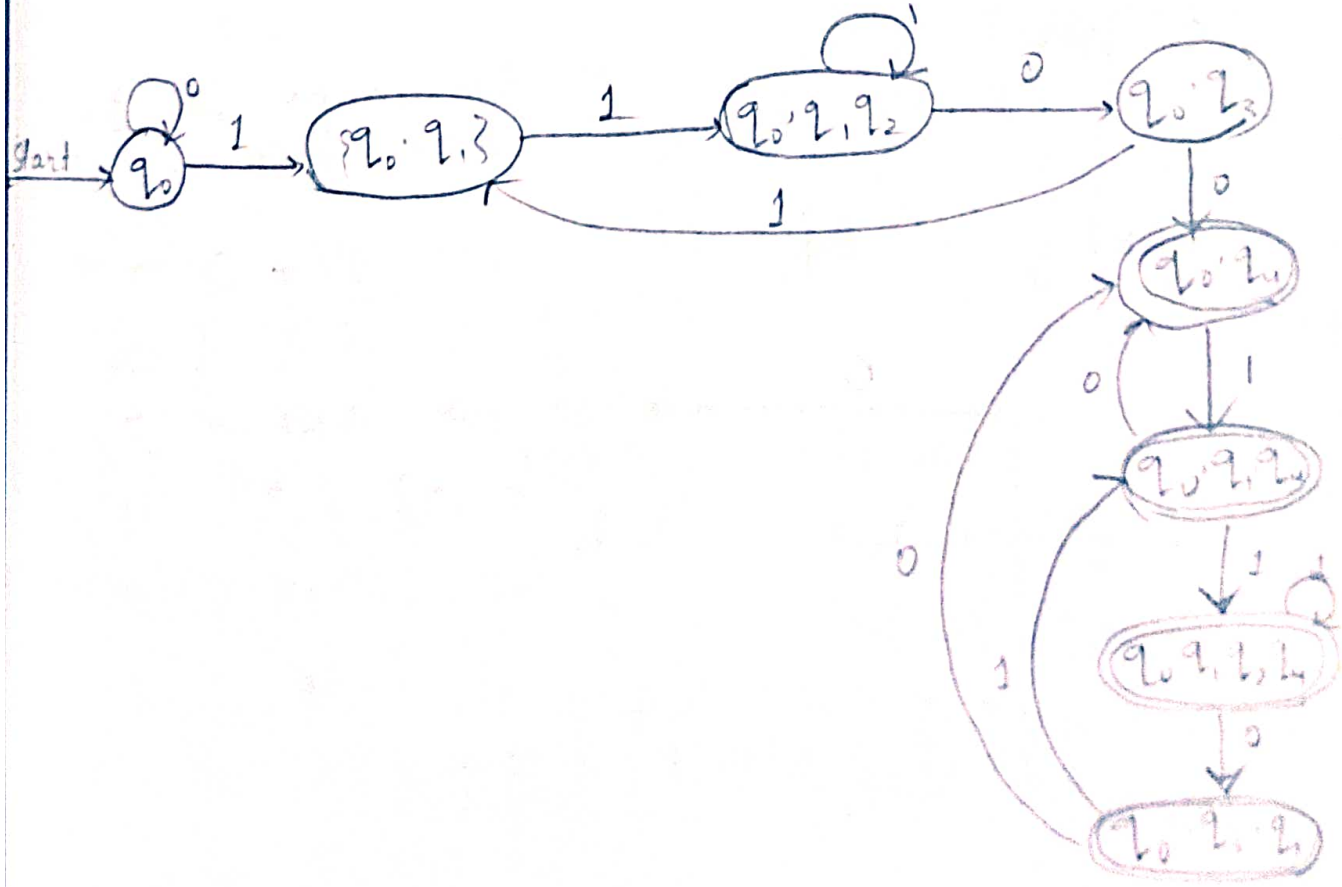
$q_0 = q_0$

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

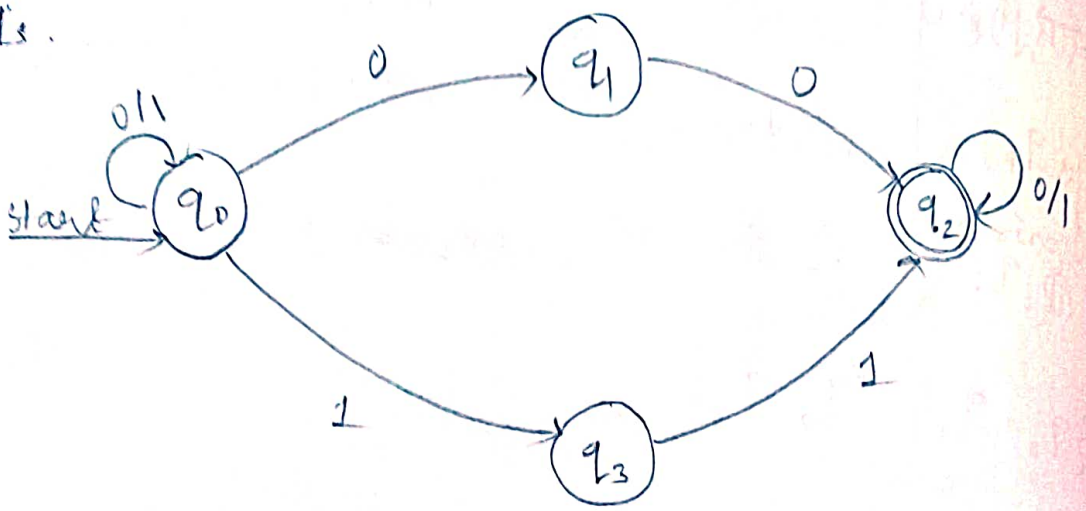
$F = \{q_4\}$

δ	0	1
$\rightarrow q_0$	q_0	$\{q_0, q_1\}$
q_1	ϕ	q_2
q_2	q_3	ϕ
q_3	q_4	ϕ
(q_4)	q_4	q_4

	0	1
$\rightarrow q_0$	$\{q_0\}$	$\{q_0 q_1\}$
$\{q_0 q_1\}$	$\{q_0\}$	$\{q_0 q_1 q_2\}$
$\{q_0 q_1 q_2\}$	$\{q_0 q_3\}$	$\{q_0 q_1 q_2\}$
$\{q_0 q_3\}$	$\{q_0 q_4\}$	$\{q_0 q_1\}$
$\{q_0 q_4\}$	$\{q_0 q_4\}$	$\{q_0 q_1 q_4\}$
$\{q_0 q_1 q_4\}$	$\{q_0 q_4\}$	$\{q_0 q_1 q_2 q_4\}$
$\{q_0 q_1 q_2 q_4\}$	$\{q_0 q_3 q_4\}$	$\{q_0 q_1 q_2 q_4\}$
$\{q_0 q_3 q_4\}$	$\{q_0 q_4\}$	$\{q_0 q_1 q_4\}$



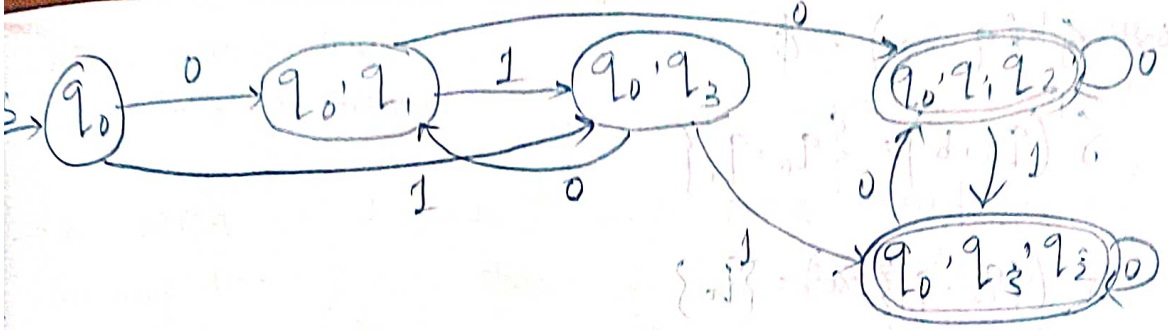
Q/ Design a NFA which will accept set of two consecutive 0s or and two consecutive 1s.



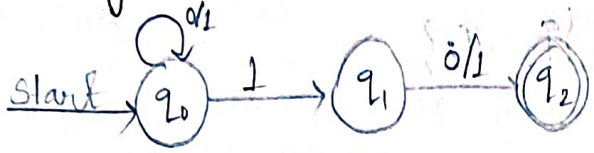
$Q = \{q_0, q_1, q_2, q_3\}$

	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	$\{q_2\}$	ϕ
q_2	$\{q_2\}$	$\{q_2\}$
q_3	ϕ	$\{q_2\}$

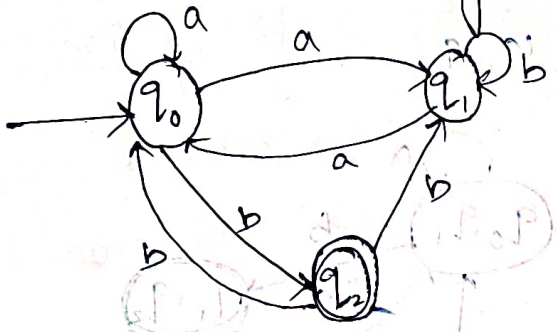
	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_3\}$
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_3\}$
$\{q_0, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_3, q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_3, q_2\}$
$\{q_0, q_3, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_3, q_2\}$



Q/ Design a NFA which will accept all the binary string in which second bit is always '1' from right hand side.



Q/ Construct a DFA diagram for NFA given below



Let the equivalent DFA, $M_2 = (Q', \Sigma, q_0, f, \delta')$
for a given NFA, $M_1 = (Q, \Sigma, q_0, f, \delta)$

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

Σ of DFA = Σ of NFA

q_0 of DFA = q_0 of NFA

$F' = \{q_2\}, \{q_1, q_2\}$

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

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

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

$= \{q_0, q_1\}$

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

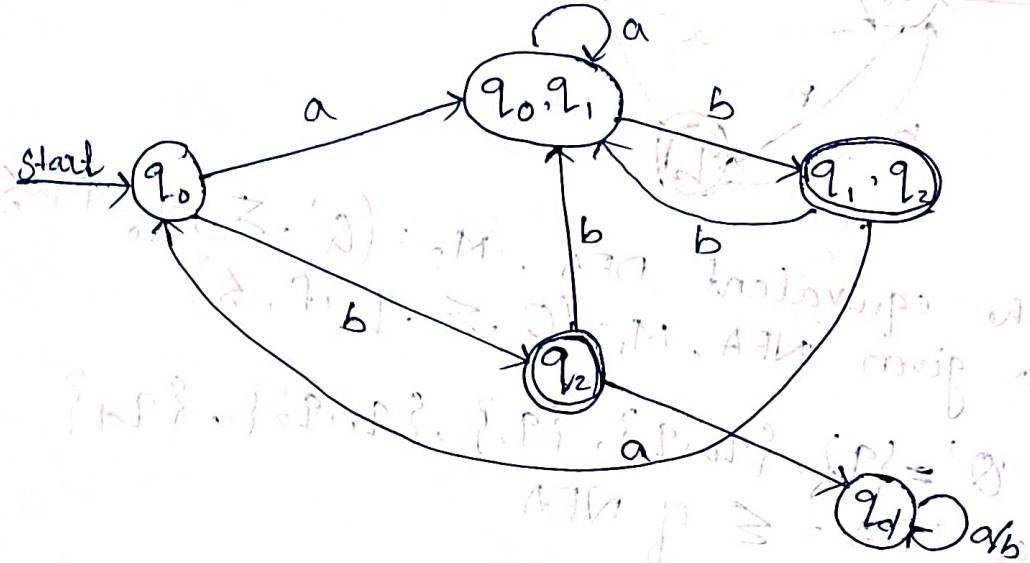
$\delta'(q_2, a) = \phi$

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

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

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

	a	b
q_0	$\{q_0, q_1\}$	$\{q_2\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_2, q_2\}$
$\{q_2\}$	ϕ	$\{q_0, q_1\}$
$\{q_1, q_2\}$	$\{q_0\}$	$\{q_1, q_0\}$



Difference b/w

NFA

- In NFA multiple transition are allowed.
- There is are multiple path from current state to next state for a specific input.

DFA

- In DFA multiple transition are not allow.
- There is only one path from current state to next state for a specific input.

→ NFA allow the null move or ∈ move.

→ In NFA dead state / dummy state / trap state is not allowed.

→ Design and const. is little bit easy.

→ In NFA. the transition func^n always return multiple value.

→ NFA ~~are~~ does not used in digital computer.

→ DFA does not allow the NULL move or ∈ move.

→ In DFA. dead state / dummy state / trap state ~~are~~ is allowed.

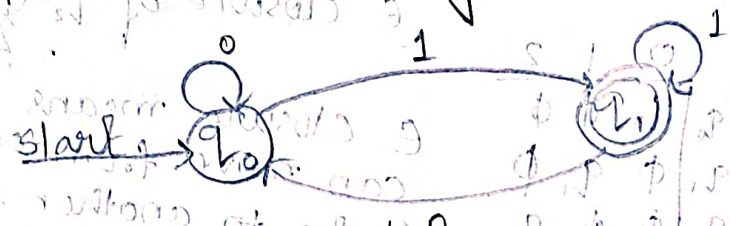
→ Design and const. is little bit diff. for DFA.

→ In DFA the transition func^n always return a single value.

→ DFA used in digital computer.

Acceptibility of a String :

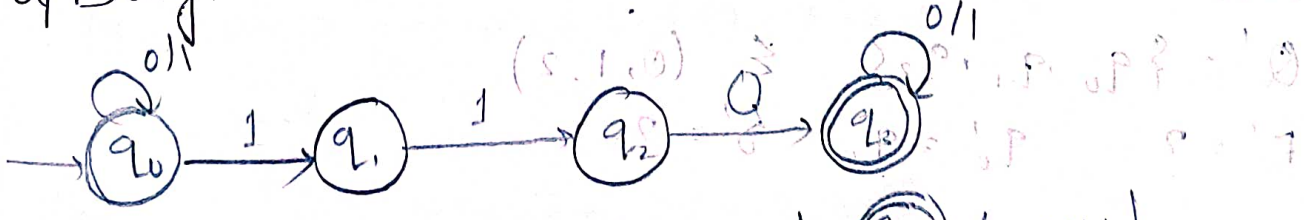
Q/ Design a DFA which will accept all the odd no's in binary form.



$q_0 \xrightarrow{101} q_1 \xrightarrow{01} q_0 \xrightarrow{1} q_1$ (final state) (accept)

$\delta(q_0, 101) = \delta(q_1, 01) = \delta(q_0, 1) = 1.$

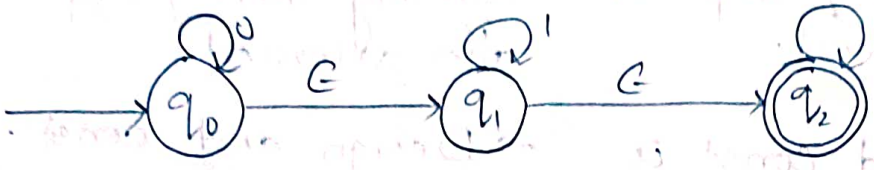
Q/ Design a NFA which accept set of string 110 as subst



$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3$ (accept)

$q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0$

Q/ Convert NFA with ϵ moves to its equivalent NFA without ϵ for the figure given below.



For the conversion of NFA with ϵ transition to the NFA without ϵ we must follow one rule i.e,

$$\hat{\delta} = \epsilon \text{ closure } (\delta(\hat{\delta}(q,\epsilon),a))$$

For this diagram, if we will express NFA with ϵ in the form of 'M' then

$M' = \{Q, \Sigma, q_0, F, \delta\}$

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

$\Sigma = (0, 1, 2)$

$q_0 = q_0$

$F = q_2$

$\delta = ---$

δ	0	1	2
q_0	q_0	ϕ	ϕ
q_1	ϕ	q_1	ϕ
q_2	ϕ	ϕ	q_2

then its equivalent NFA without ϵ

$M' = \{Q', \Sigma, q_0', F', \delta'\}$

the the transition function of NFA without ϵ we will find out

$Q' = \{q_0, q_1, q_2\}$ $\quad \Sigma = (0, 1, 2)$

$F' = ?$ $\quad q_0' = q_0$ $\quad \delta' = ?$

ϵ closure of $q_0 = (q_0 q_1$

ϵ Closure of $q_1 = (q_1 q$

ϵ Closure of $q_2 = (q_2)$

→ ϵ closure is denoted by $\hat{\delta}$ i.e, we can say that ϵ closure of $q_1 = (q_1, q_2, q_2) = \hat{\delta}(q_0, \epsilon)$

ϵ closure of $q_1 = (q_2 q_2)$
$= \delta(q_1, \epsilon)$

ϵ closure of $q_2 = (q_2)$
$= \delta(q_2, \epsilon$

ϵ closure means we can move from one state to another state without reading any input. or another meaning is there is some transition from one state to another state with empty string.

$'(q_0, 0) = \epsilon$ closure $(\delta(\hat{\delta}(q_0, \epsilon), 0))$

$= \epsilon$ closure $(\delta(q_0, q_1, q_2)\ 0) = \delta(q_0, 0) \cup (q_0, 0) -$

$= \epsilon$ closure $(q_0 \cup \phi \cup \phi) = (q_0, q_1, q_2)$

$'(q_0, 1) = \epsilon$ closure $(\delta(\hat{\delta}(q_0, \epsilon), 1))$

$= \epsilon$ closure $(\delta(q_0, q_1, q_2), 1)$

$= \epsilon$ closure $(\phi \cup q_1 \cup \phi) = \epsilon$ closeure of q_1.

$= (q_1, q_2)$

$'(q_0, 2) = \epsilon$ closure $(\delta(\hat{\delta}(q_0, \epsilon), 2))$

$= \epsilon$ closure $(\delta(q_0, q_1, q_2)2)$

$= \epsilon$ closure $(\phi \cup \phi \cup q_2)$

$= (q_2)$

$'(q_1, 0) = \epsilon$ closure $(\delta(\hat{\delta}(q_1, \epsilon), 0))$

$= \epsilon$ closure $(\delta(q_1, q_2), 0)$

$= \epsilon$ closure $(\phi \cup \phi) = \phi$

$(q_1, 1) = \epsilon$ closure $(\delta(\hat{\delta}(q_1, \epsilon), 1))$

$= \epsilon$ closure $(\delta(q_1, q_2), 1)$

$= \epsilon$ closure $(q_1 \cup \phi) = (q_1, q_2)$

$(q_1, 2) = \epsilon$ closure $(\delta(\hat{\delta}(q_1, \epsilon) 2))$

$= \epsilon$ closure $(\delta(q_1, q_2), 2)$

$= \epsilon$ closure $(\phi \cup q_2) = (q_2)$

$'(q_2, 0) = \epsilon$ closure $(\delta(\hat{\delta}(q_2, \epsilon) 0))$

$= \phi$

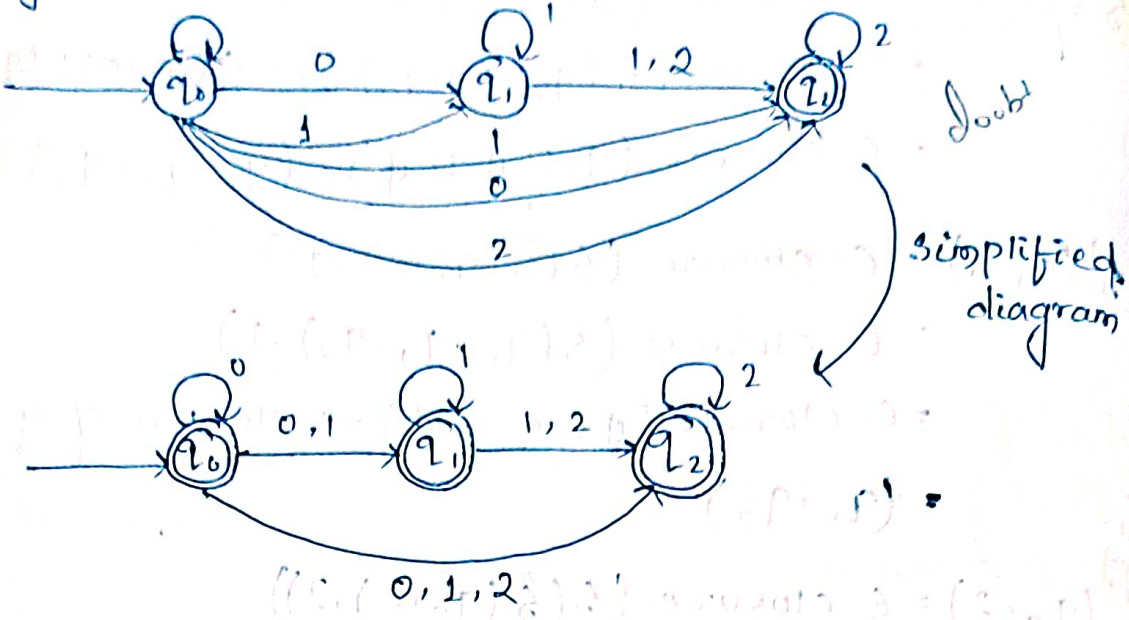
$'(q_2, 1) = \epsilon$ closure $(\delta(\hat{\delta}(q_2, \epsilon) 1))$

$= \phi$

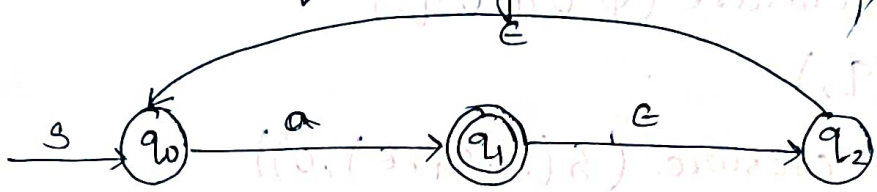
$'(q_2, 2) = \epsilon$ closure $(\delta(\hat{\delta}(q_2, \epsilon) 2))$

$= q_2$

Using δ', we will draw NFA without ϵ



2. Convert the following ϵ NFA to DFA



Step 1 : convert ϵ NFA to without ϵ NFA.

Step 2 : convert NFA to DFA.

ϵ Closure of $q_0 = (q_0)$

ϵ closure of $q_1 = (q_1, q_2, q_0)$

ϵ closure of $q_2 = (q_2, q_0)$

If NFA with ϵ is $M = (Q, \Sigma, q_0, F, \delta)$

$Q = (q_0, q_1, q_2)$

$\Sigma = a$

$q_0 = q_0$

$F = q_1$

$\delta =$ [ ]

If equivalent NFA without ϵ is M'

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

$Q' = (q_0, q_1, q_2)$, $q_0' = q_0$,

$\Sigma = a$, $F' = q_1$

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

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

$= (q_0, q_1, q_2)$

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

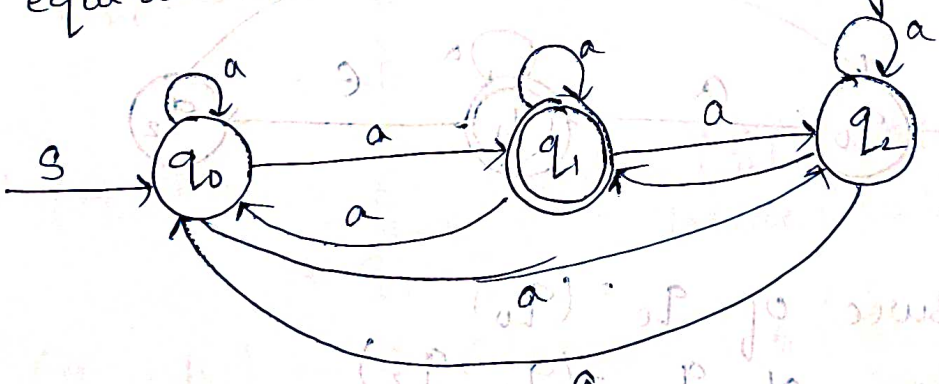
$= \epsilon$ closure $(\delta(q_1, q_2, q_0), a)$

$= (q_1, q_0, q_2)$

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

$= (q_0, q_1, q_2)$

It's equivalent NFA without ϵ diagram is

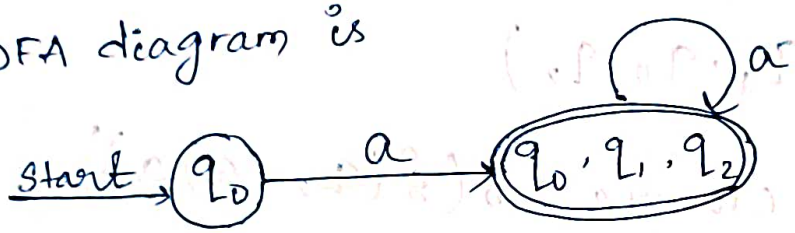


δ	a
$\rightarrow q_0$	(q_0, q_1, q_2)
(q_1)	(q_0, q_1, q_2)
q_2	(q_0, q_1, q_2)

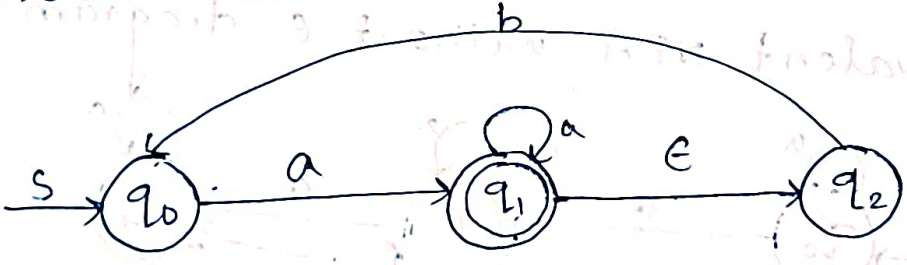
It's equivalent DFA ~~table~~ transition table is

	a
$\rightarrow q_0$	(q_0, q_1, q_2)
(q_0, q_1, q_2)	(q_0, q_1, q_2)

It's DFA diagram is



Q) Convert this NFA to equivalent DFA



ϵ closure of $q_0 = (q_0)$

ϵ closure of $q_1 = (q_1, q_2)$

ϵ closure of $q_2 = (q_2)$

If NFA with ϵ is $M = (Q, \Sigma, q_0, F, \delta)$

$Q = (q_0, q_1, q_2)$

$\Sigma = (a, b)$

$q_0 : q_0$

$F = q_1$

δ'	a	b
q_0	q_1	ϕ
q_1	q_1	ϕ
q_2	ϕ	q_0

If equivalent NFA without ϵ is M'

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

$Q' = (q_0, q_1, q_2)$, $q_0' = q_0$

$\Sigma = a, b$, $F' =$

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

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

$= \epsilon$ closure (q_1)

$= q_1, q_2$

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

$= \phi$

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

$= \epsilon$ closure $(q_1 \cup \phi)$

$= q_1, q_2$

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

q_1, q_2

$\phi \cup q_0$

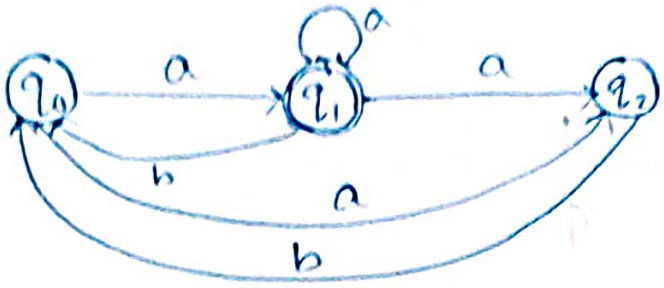
$= \phi$

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

$= \phi$

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

It's equivalent NFA without ε is.

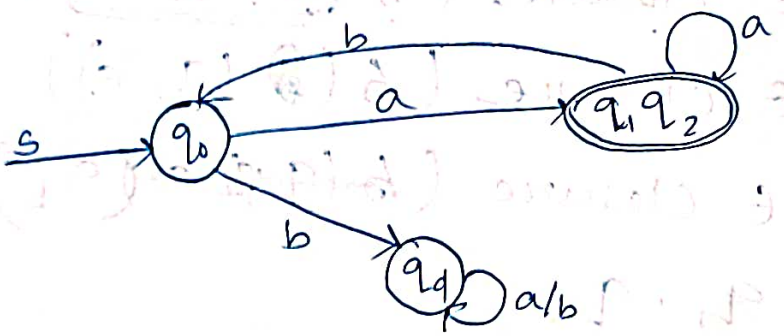


δ-table of above NFA:

	a	b
q_0	q_1, q_2	ϕ
q_1	q_1, q_2	q_0
q_2	ϕ	q_0

Its equivalent DFA transition table is

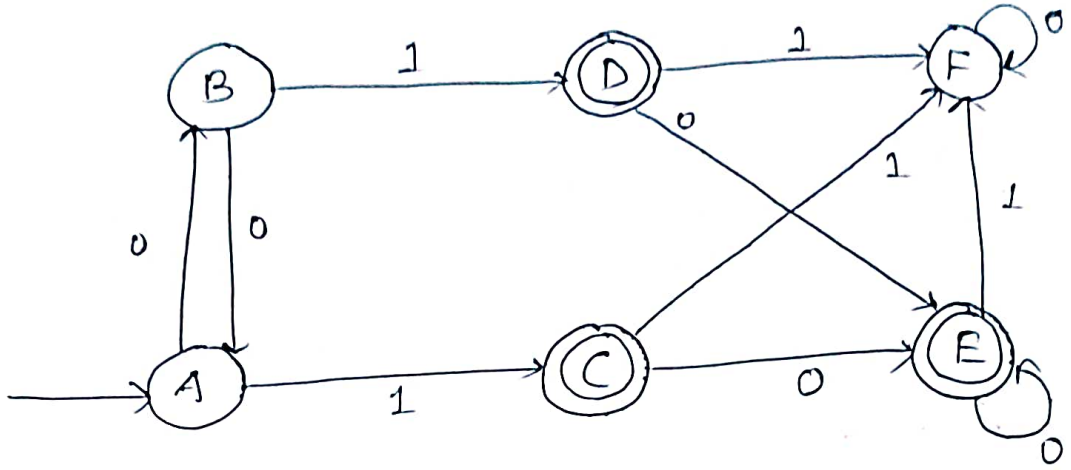
	a	b
q_0	$\{q_1, q_2\}$	ϕ
$q_1 q_2$	$\{q_1, q_2\}$	q_0



- In minimization of finite automata we decrease the n- no. of steps.
- In minimization we can directly delete the unreachable state.

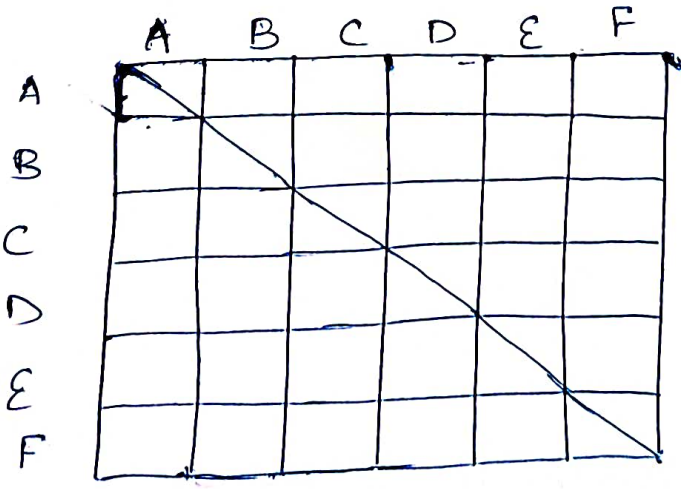
ii) Myhill Nerode Theorem ⟶ Table filling method

Q) Construct the minimization for the given figure.



	0	1
→ A	B	C
B	A	D
C	E	F
D	E	F
E	E	F
F	F	F

Step1 :



I divide this table into two parts and consider the lower part

Step 2:

Step 3:

BA $\delta(B,0) = A$, $\delta(A,0) = B$] × $\delta(B,1) = D$, $\delta(A,1) = C$] ×

DC $\delta(D,0) = E$, $\delta(C,0) = E$] × $\delta(D,1) = F$, $\delta(C,1) = F$] ×

EC $\delta(E,0) = E$, $\delta(C,0) = E$] × $\delta(E,1) = F$, $\delta(E,1) = F$] ×

ED $\delta(E,0) = E$, $\delta(D,0) = E$] × $\delta(E,1) = F$, $\delta(D,1) = F$]

FA $\delta(F,0) = F$, $\delta(A,0) = B$] ✓ $\delta(F,1) = F$, $\delta(A,1) = C$] ✓

FB

$\delta(F,0) = F$]
$\delta(B,0) = A$] ✓

$\delta(F,1) = F$]
$\delta(B,1) = D$] -

Step 4

Repeat these steps atleast two steps until you are getting no changes.

BA

$\delta(B,0) = A$
$\delta(A,0) = B$

$\delta(B,1) = D$
$\delta(A,1) = C$

DC

$\delta(D,0) = \varepsilon$
$\delta(C,0) = \varepsilon$

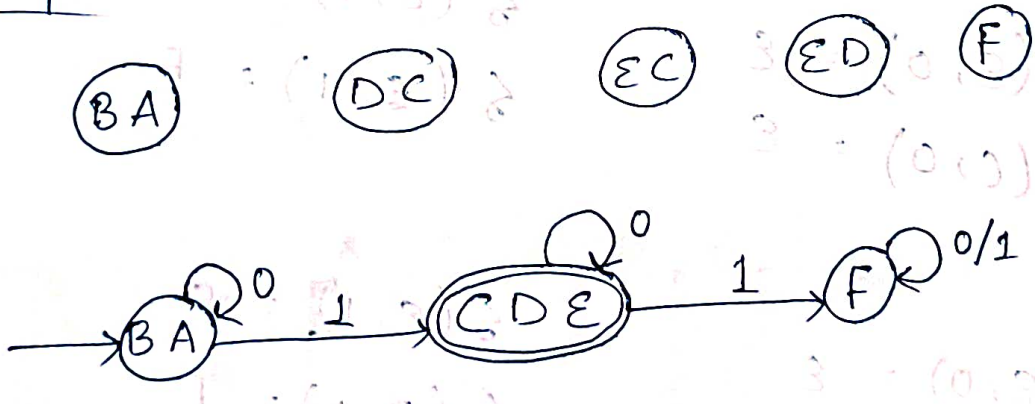
$\delta(D,1) = F$
$\delta(C,1) = F$

εC

$\delta(\varepsilon,0) = \varepsilon$
$\delta(C,0) = \varepsilon$

$\delta(\varepsilon,1) = F$
$\delta(C,1) = F$

Step 5



Regular Expression

If a lang. is accepted by a that lang. is a regular lang. or regular expression.

→ In regular expression we mostly use 3 symbol : + • *

→ + indicate union
• indicate concadination
* indicate Kleene closure

eg: $(01)^*$

$(01)^0 = \epsilon$
$(01)^1 = 01$
$(01)^2 = 0101$
$(01)^3 = 010101$

→ If we will write $(r_1 + r_2)$ i.e, union of two regular expression.

$(r_1 \cdot r_2)$ i.e, concadination of two regular expression r_1 and r_2 or r_1 followed by r_2.

(r_1^*) means zero or some occurance of r_1.

2) Describe the following

i) L_1 = Set of all strings 0's and 1's ending with 00.

→ $(0+1)^* 00$

ii) L = Set of all strings of 0's and 1's starting with one 0 and ending with one 1

$$0\,(0+1)^*\,1$$

iii) $L = (\epsilon, aa, aaaa, aaaaaa, \ldots)$

$$(aa)^*$$

Q) Define the regular expression to denote any number of 0's followed by 1's followed by 2's.

$$0^* \cdot 1^* \cdot 2^*$$

Q) Write the regular expression for the set of all string ~~follo~~ contain 1100 as substring.

$$(0+1)^* 1100 (0+1)^*$$

Convertion of regular Expression to finite Automata:

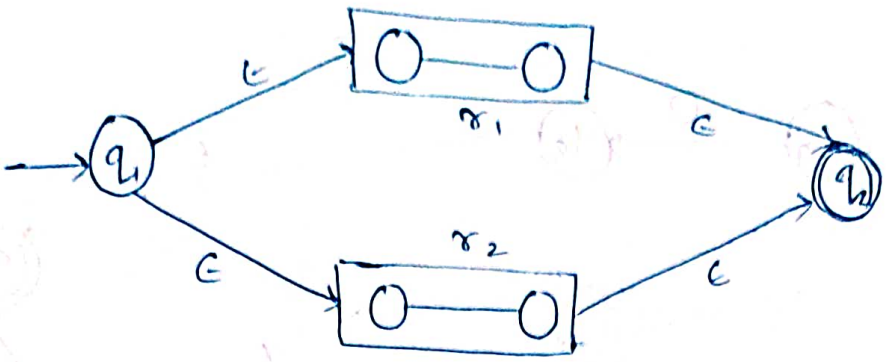
For this Conversion we must follow three types of model for the union (+), ~~condat~~ concadination (·) & Kleene Closure (*)

The model for union $(r_1 + r_2)$

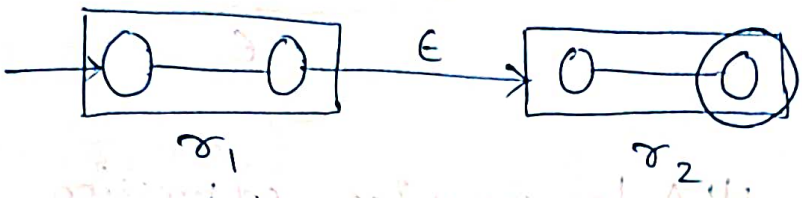
The model for concadination $(r_1 \cdot r_2)$

The model for Kleene closure (r_1^*)

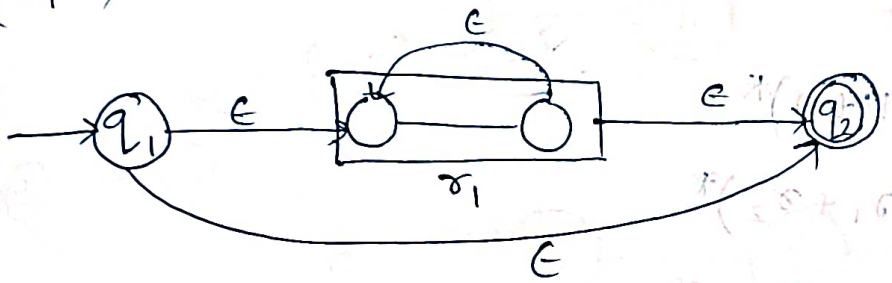
$(r_1 + r_2)$



$(r_1 \cdot r_2)$



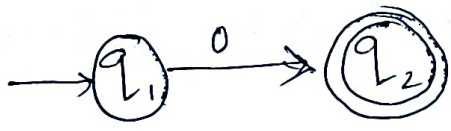
(r_1^*)



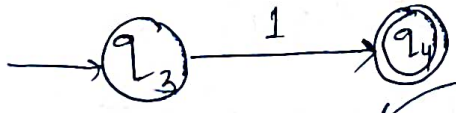
Q) Construct a ϵ NFA for the regular expression for $0 + 0.1^*$

Step 1:

0

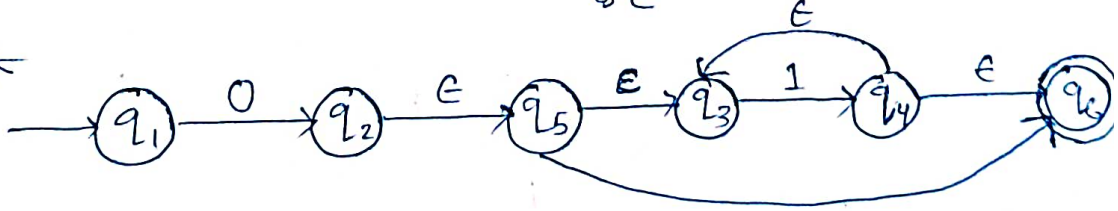


1

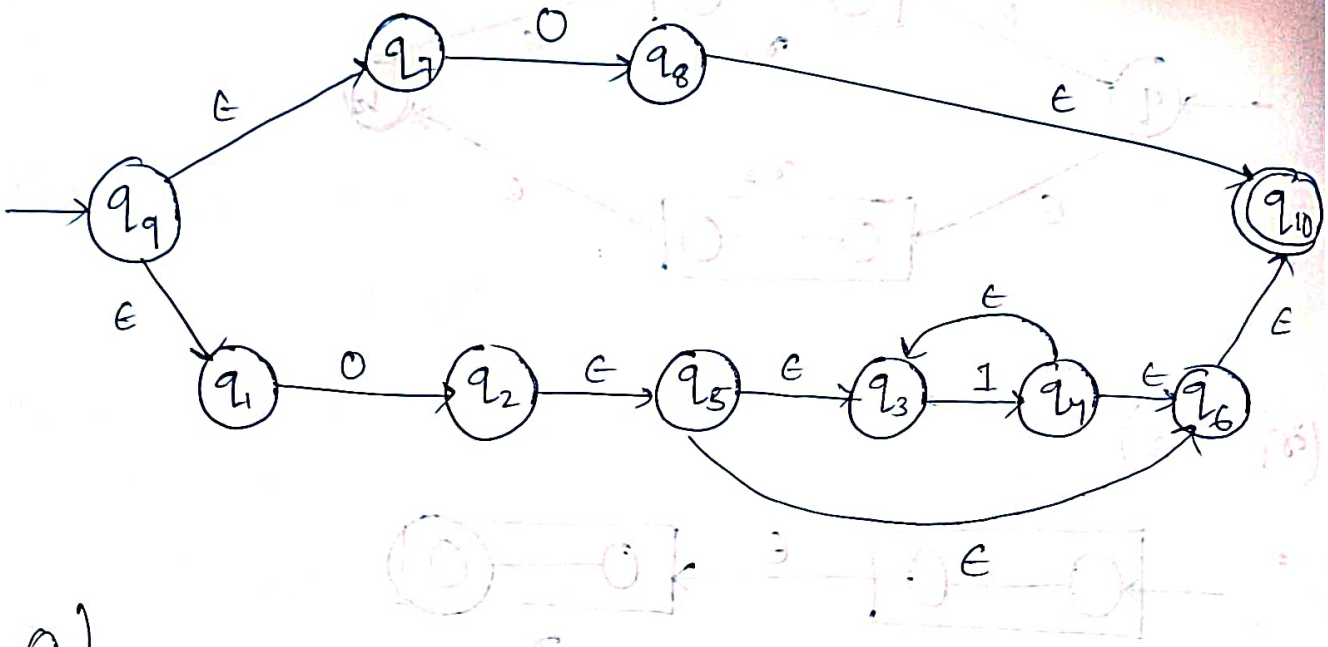


1^*

$0 \cdot 1^*$



$0 + 0.1^*$



Q) Construct a NFA for regular expression

$r = (11+0)^*$

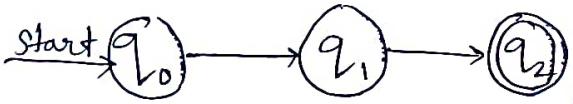
Step 1 : $r = (11+0)^*$

$r = (r_1 + r_2)^*$

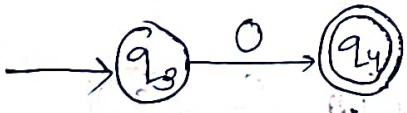
$r_1 = 11$, $r_2 = 0$

Step 2 :

$r_1 = 11$

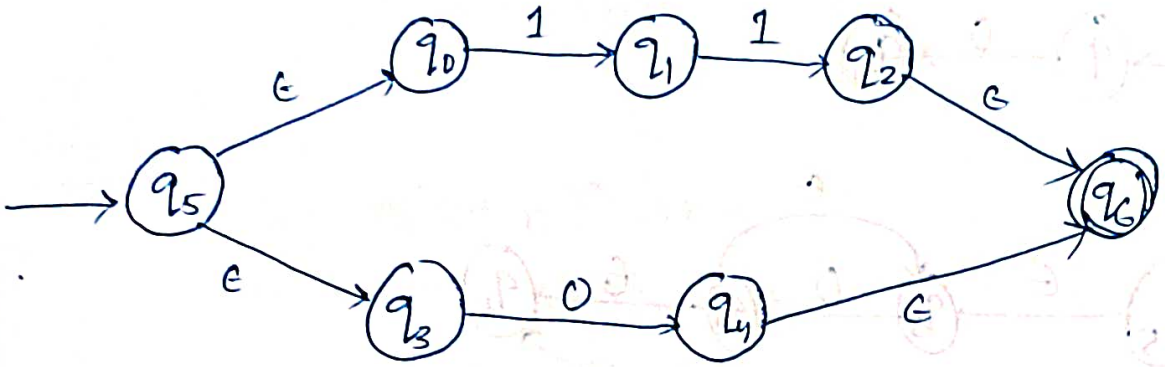


$r_2 = 0$



Step 3 :

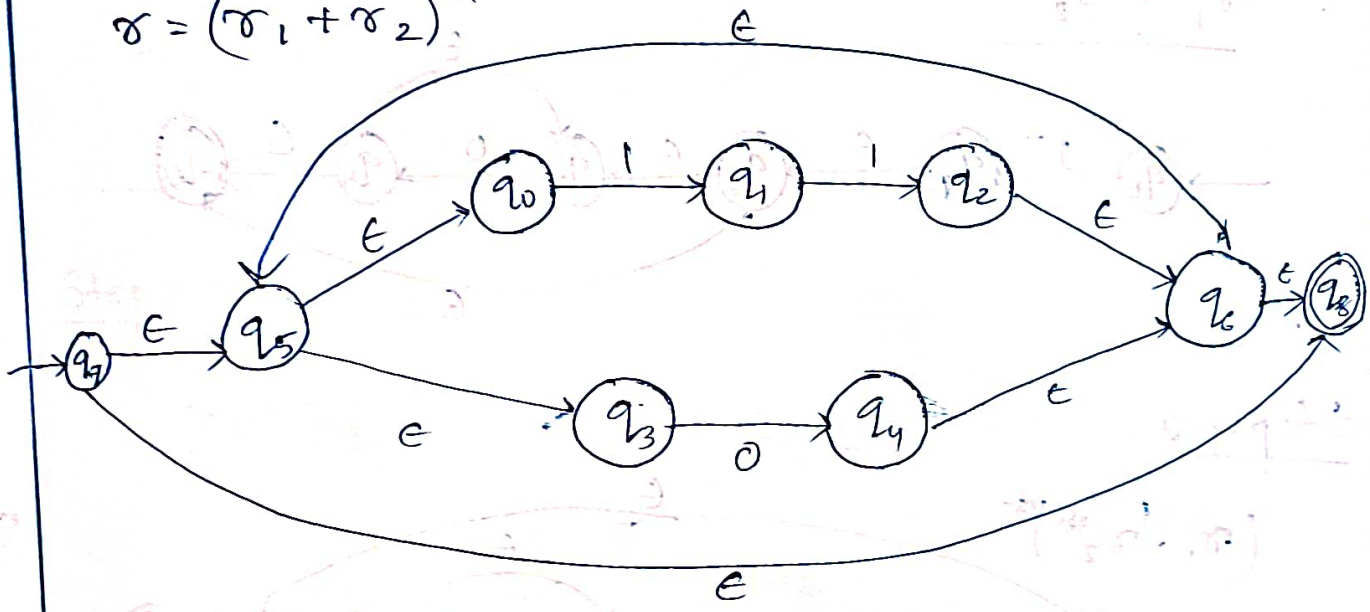
Step 3 :



$r_1 + r_2$

Step 4 :

$r = (r_1 + r_2)^*$



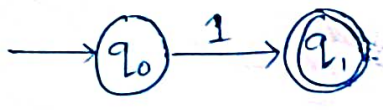
Q) Construct a ϵ NFA for the regular expression $r = (1.0^*)^*$

Step 1 : $r = (1 \cdot 0^*)^*$

$r = (r_1 \cdot r_2^*)^*$

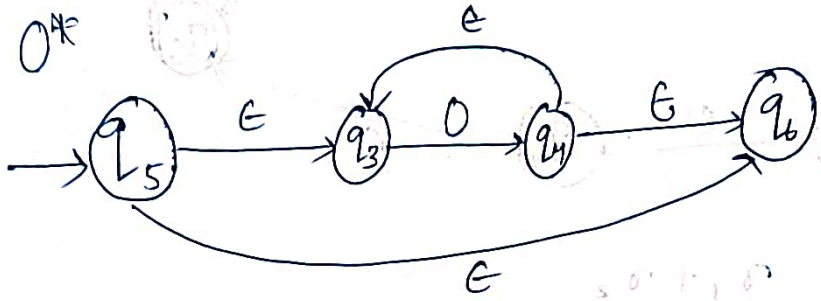
$r_1 = 1 \quad , \quad r_2 = 0^*$

Step 2 :

$r_1 = 1$ 

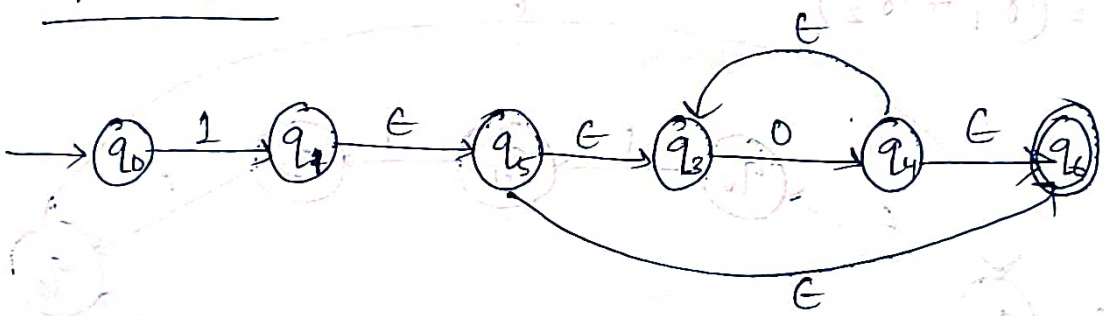
$r_2 = 0^*$

0^*



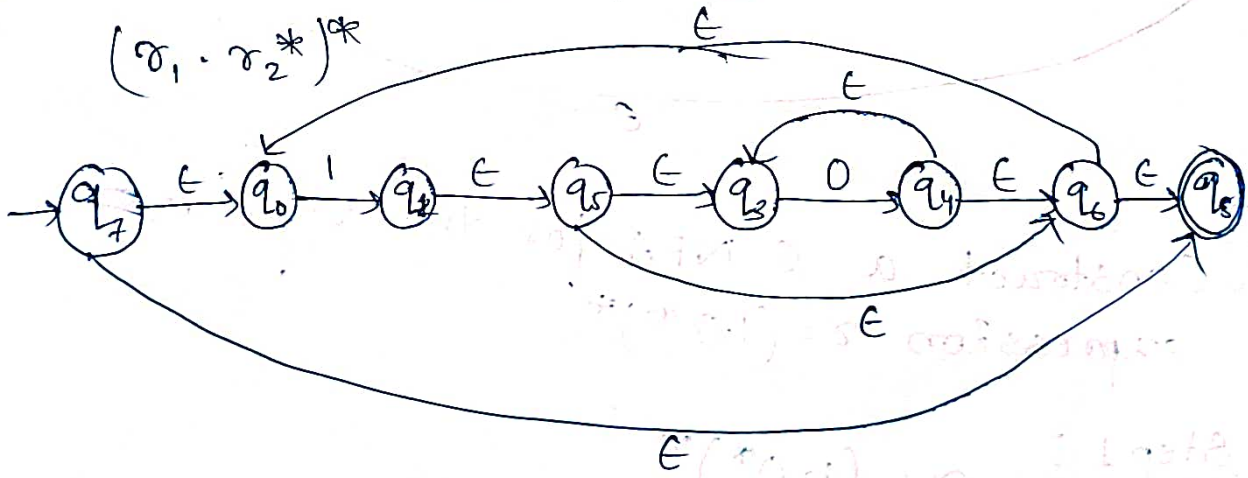
Step 3:

$r_1 \cdot r_2^*$



Step 4:

$(r_1 \cdot r_2^*)^*$



Q) Construct a ϵ NFA for regular expression

$(a+b)^* aba$

Step 1 :

$r = (a+b)^* aba$

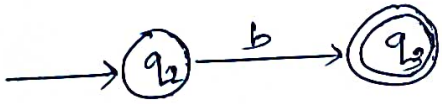
$r = (r_1 + r_2)^* . a.b.a$

$r_1 = a \quad , \quad r_2 = b$

Step 2 :

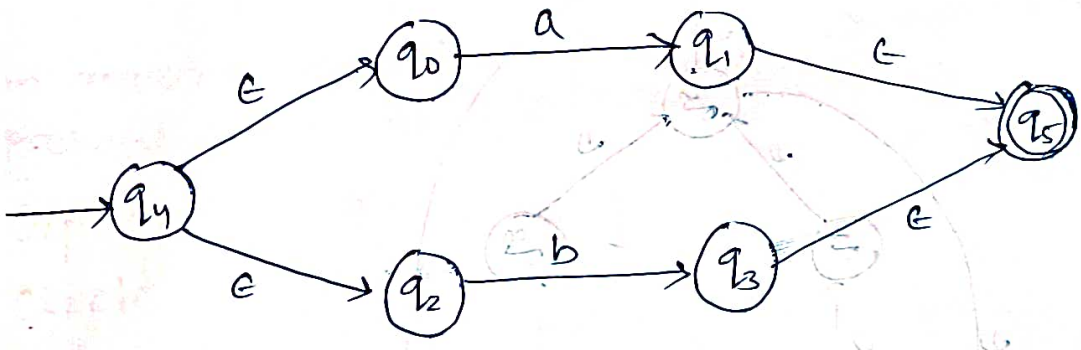
$r_1 = a$

$r_2 = b$



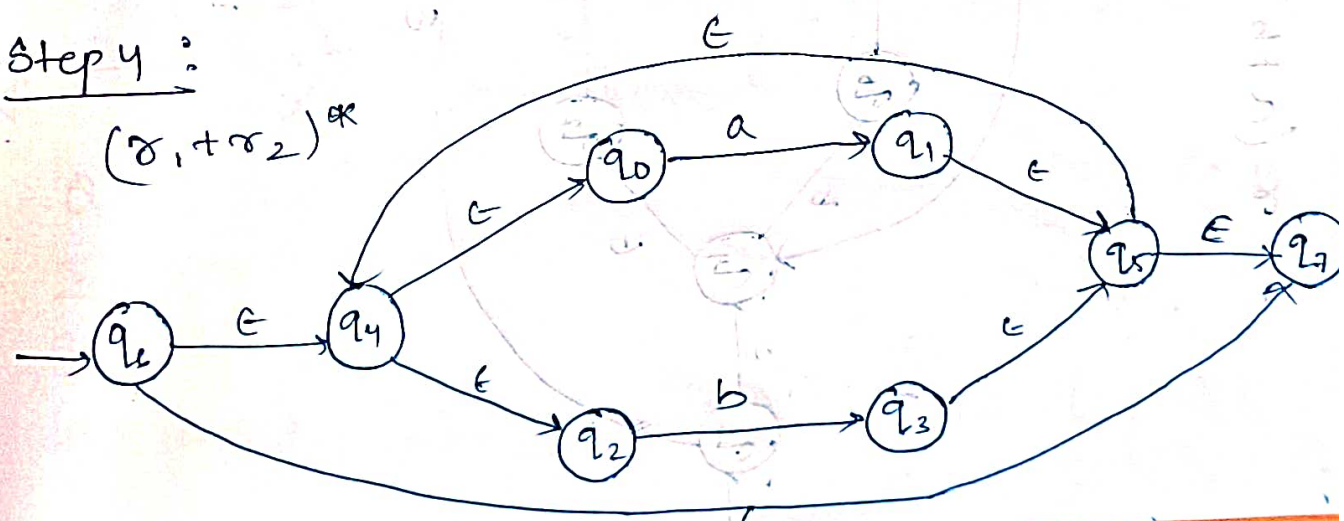
Step 3 :

$r_1 + r_2$



Step 4 :

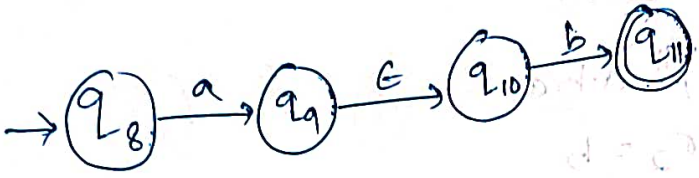
$(r_1 + r_2)^*$



Step 5 :

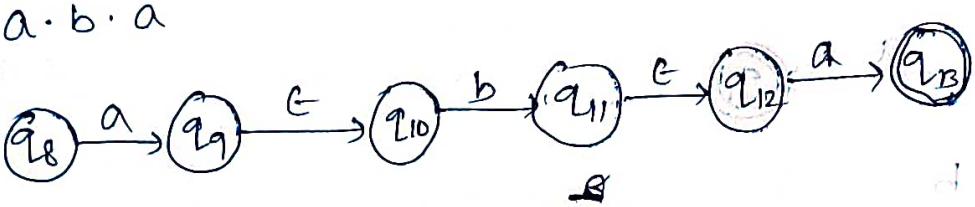
$r = (r_1 + r_2)^* \cdot abba$

$a \cdot b$

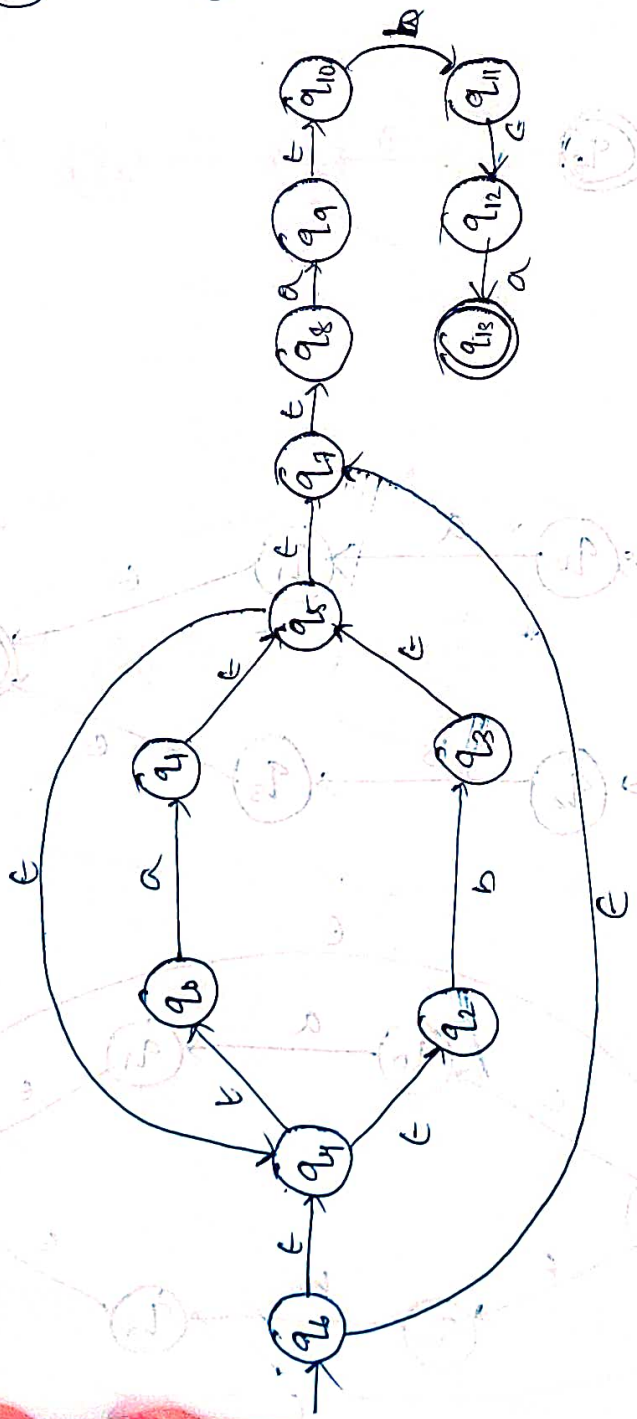


Step 6 :

$a \cdot b \cdot a$



Step 7 :



$r = (a+b)^* abba$

Moore machine & Mealy machine :

Moore machine and mealy machine are known as finite automata with output.

Moore Machine :

Moore machine is a finite automata which consist of 6 numbers of tuples.

$$M_0 = (Q, \Sigma, \Delta, \delta, q_0, \lambda)$$

Q = finite set of states

Σ = input alphabets / i/p symbols.

q_0 = initial state

δ = transition function which we find out by formula $(Q \times \Sigma = Q)$

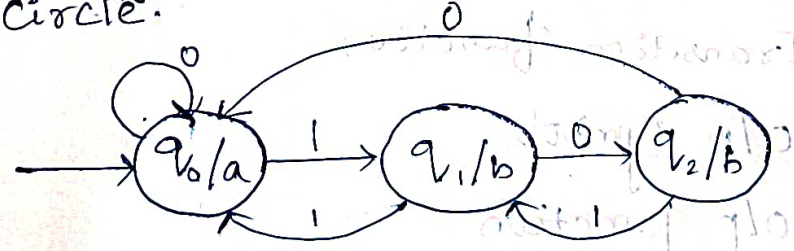
Δ = o/p symbols

λ = o/p functions.

→ In moore machine the o/p symbols are always present with the states.

→ o/p symbols are always written within the state circle.

eg:



$Q = (q_0, q_1, q_2)$

$\Sigma = (0, 1)$

$q_0 = q_0$

$\Delta = (a, b)$

$\lambda \cdot q_0 - a$

$q_1 - b$

$q_2 - b$

$\delta = q_0, q_1, q_2 \times (0, 1)$

$q_0, 0 = q_0$	$q_0, 1 = q_1$
$q_1, 0 = q_2$	$q_1, 1 = q_0$
$q_2, 0 = q_0$	$q_2, 1 = q_1$

→ In Moore machine if we pass the length of the i/p string is 'n' then the o/p is always 'n+1'.

eg: pass - 00110
o/p - a a a b a a

pass - ∈
o/p - a

pass - 01
o/p - a a b

pass - 010
o/p - a a b b

→ In moore machine if we pass a null string or empty string then o/p is always same with o/p of the ~~initial~~ initial state.

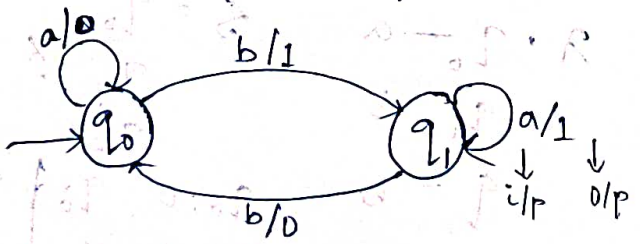
Mealy Machine:

→ In mealy machine we are using 6 no. of tuples.

$$M_e = (Q, \Sigma, \Delta, \delta, q_0, \lambda)$$

Q = finite set of state
Σ = i/p alphabets / i/p symbols
q_0 = initial state
δ = Transition function
Δ = o/p symbol
λ = o/p function.

eg:



→ left side is always i/p & right side is always o/p.

→ In mealy machine if we pass the length of string is 'n' then the length of o/p is always 'n'.

eg: i/p - a b b a
o/p - 0 1 0 0

i/p : ∈
o/p : nothing

δ-table for Mealy machine.

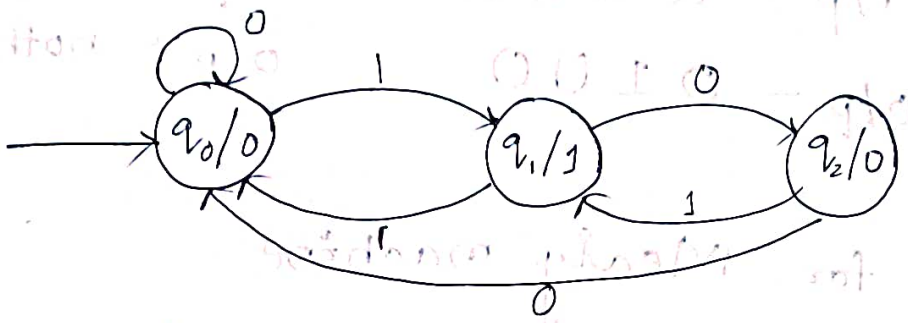
	a		b	
	state	o/p	state	o/p
q_0	q_0	0	q_1	1
q_1	q_1	1	q_0	0

δ-table for Moore Machine :

	i/p		o/p
	0	1	
q_0		q_1	a
q_1		q_0	b
q_2		q_1	b

Conversion Of Moore machine to Mealy mac

Q/ convert Moore machine to mealy meachine, for the figure given below.



$Q = (q_0, q_1, q_2)$

$\Sigma = (0, 1)$

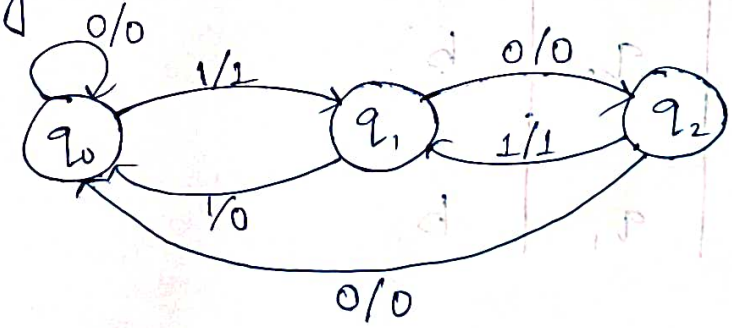
$q_0 = q_0$

$\Delta = (0, 1)$

$\delta =$

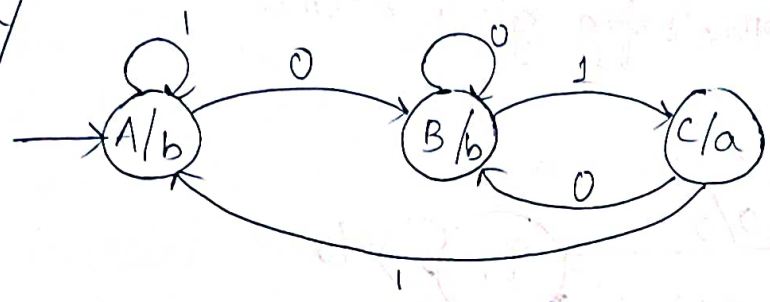
	0	1	O/P
			0
q_0	q_0	q_1	
q_1	q_2	q_0	1
q_2	q_0	q_1	0.

Mealy:

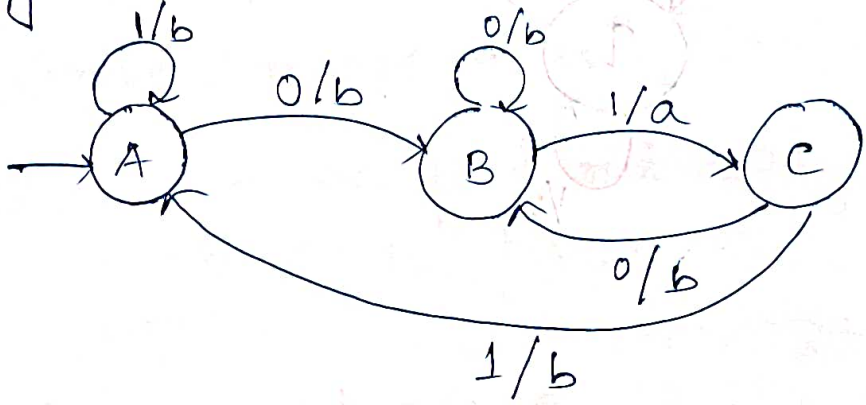


In conversion from moore to mealy the no. of states are always same.

Q/

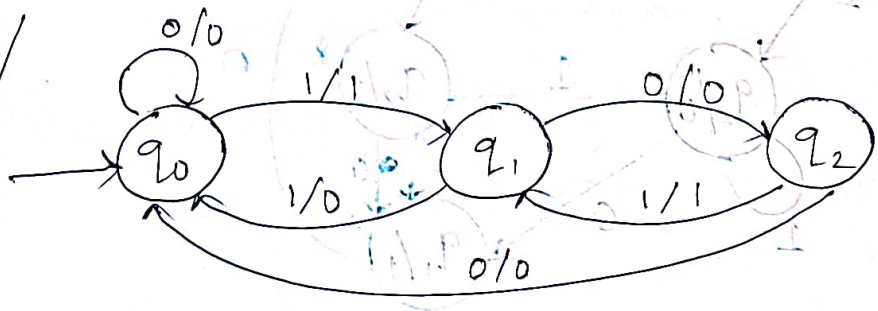


Mealy

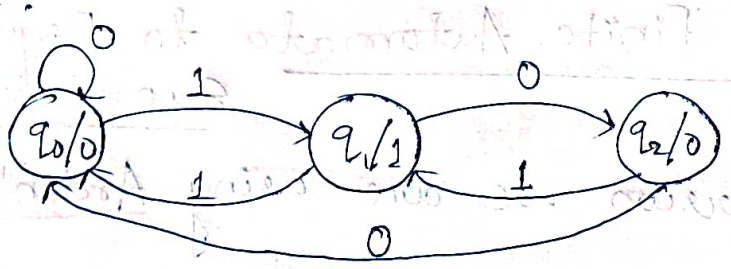


Conversion of Mealy Machine to Moore Machine.

Q/



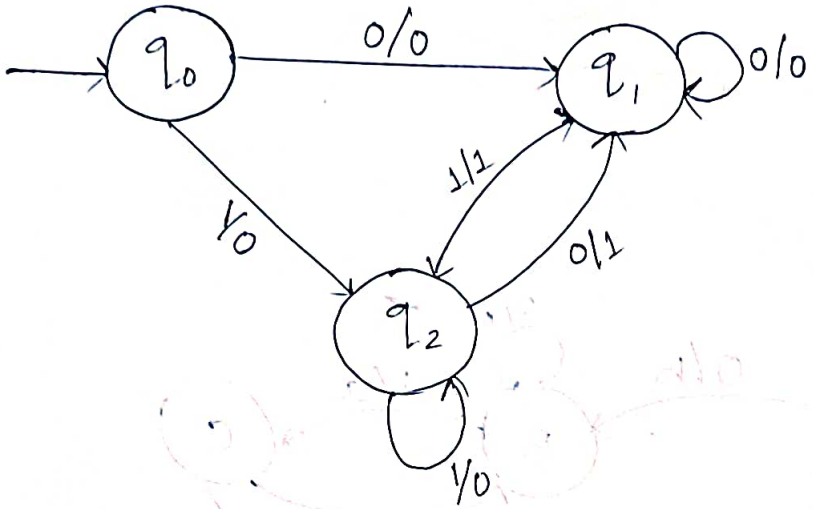
Moore :



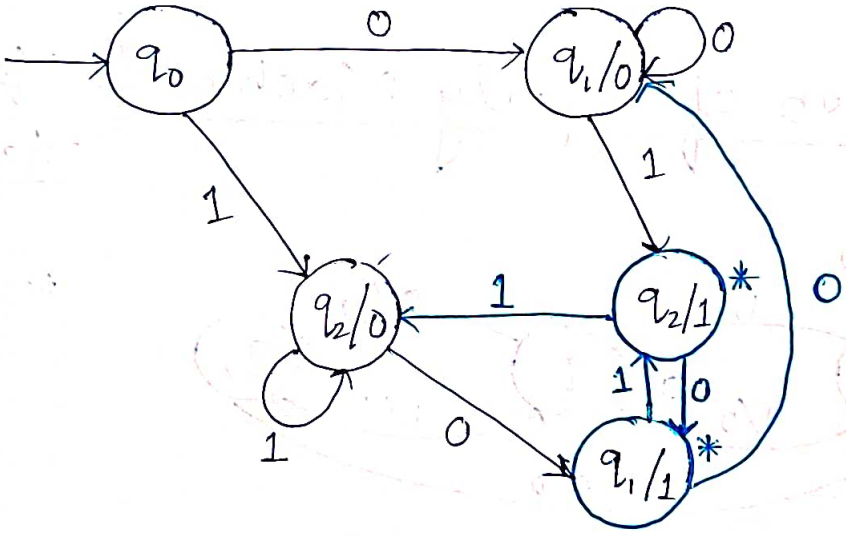
In conversion from mealy to moore the no. of states are not same.

Imp

Q. Convert this mealy machine to moore machine for the fig given below:



Moore:



Conversion of Finite Automata to Regular Expression:

For this conversion we are using Arden's theorem.

This theorem tells we find a unique solution i.e, $R = QP^*$ if the sol. format is $R = Q + RP$

For this conversion, mostly we follow 2 condition:

i) The finite automata should not contain any ϵ transition.

ii) The finite automata should contain one initial state.

If this two cond^{n}s are false then conversion is not possible.

→ For conversion we must follow some steps:

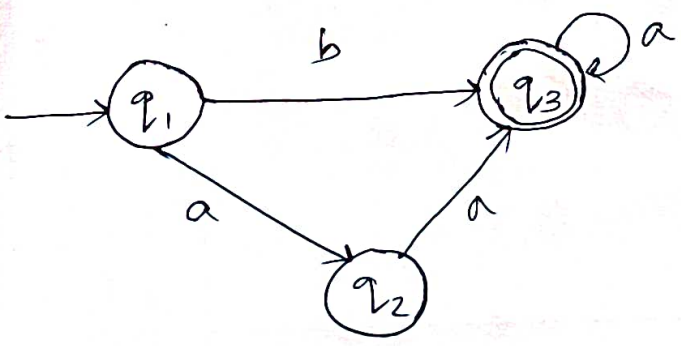
Step 1: First we find out the eq^n for each state based on incoming edges.

Step 2: We add one ϵ to the eq^n of initial state.

Step 3: Simplify this eq^n by apping Arden's theorem.

Step 4: We always find out the regular expression from the final state equation.

Q/



Eq^n for: $q_1 = \epsilon$ —— (1)

$q_2 = q_1 a$ —— (2)

$q_3 = q_1 b + q_2 a + q_3 a$ —— (3)

Apply eqn ① to eqn ②

$$q_2 = q_1 a$$

$$\Rightarrow q_2 = \epsilon a$$

$$\Rightarrow q_2 = a \quad \text{——— (4)}$$

$$q_3 = q_1 b + q_2 a + q_3 a$$

$$\Rightarrow q_3 = \epsilon b + aa + q_3 a$$

$$\Rightarrow q_3 = b + aa + q_3 a$$

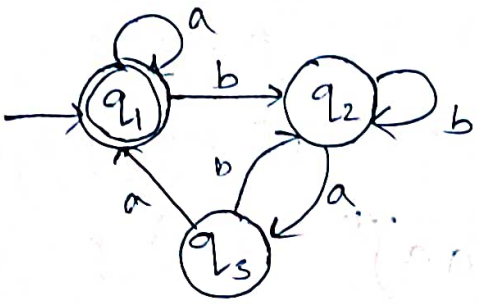
$$R = Q + RP$$

$$\Downarrow$$

$$R = QP^*$$

$$\Rightarrow R = (b + aa)a^*$$

Q/ Find out the regular expression



Eqⁿ for :

$q_1 = q_1 a + q_3 a + \epsilon$ ——— (1)

$q_2 = q_1 b + q_2 b + q_3 b$ ——— (2)

$q_3 = q_2 a$ ——— (3)

Apply eqⁿ (3) to eqⁿ (2)

$q_2 = q_2 b + q_2 ab + q_1 b$

$\Rightarrow q_2 = q_2 (b + ab) + q_1 b$

$\Rightarrow q_2 = q_1 b + q_2 (b + ab)$

$R = Q + RP$

$\Rightarrow q_2 = q_1 b (b + ab)^*$ ——— (4)

Apply eqⁿ (4) to eqⁿ (1)

$q_1 = q_1 a + q_3 a + \epsilon$

$\Rightarrow q_1 = q_1 a + q_2 aa + \epsilon$

$\Rightarrow q_1 = q_1 a + q_1 b (b + ab)^* aa + \epsilon$

$\Rightarrow$

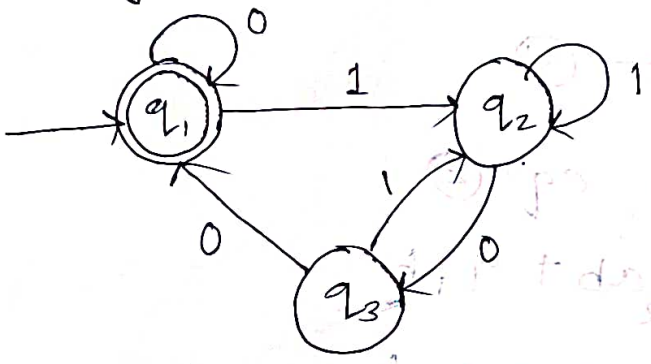
$\Rightarrow q_1 = q_{b01}\epsilon + q_1(a+b(b+ab)^*aa)$

$R = Q + RP$

$R = QP^*$

$R = (a+b(b+ab)^*aa)^*$

Q/ Construct a regular expression for the figure given below:



Eqn for

$q_1 = q_1 \cdot 0 + q_3 \cdot 0 + \epsilon$ —— ①

$q_2 = q_2 \cdot 1 + q_2 \cdot 1 + q_3 \cdot 1$ —— ②

$q_3 = q_2 \cdot 0$ —— ③

Apply eqn ③ to eqn ②

$q_2 = q_1 \cdot 1 + q_2 \cdot 1 + q_2 \cdot 0 \cdot 1$

$q_2 = q_1 \cdot 1 + q_2(1+0.1)$

$R = Q + RP$

$\Rightarrow q_2 = q_1 1 (1+0.1)^*$ —— ④

doubt/ Apply eqn ④ to eqn ①

$q_1 = q_1 \cdot 0 + q_3 \cdot 0 + \epsilon$

$\Rightarrow q_1 = q_1 \cdot 0 + q_2 \cdot 0 \cdot 0 + \epsilon$

$\Rightarrow q_1 = q_1 \cdot 0 + q_1 \cdot 1 (1 + 0 \cdot 1)^* \cdot 00 + \epsilon$

$\Rightarrow q_1 = q_1 (0 + 1(1 + 0 \cdot 1)^* \cdot 00) + \epsilon$

$\Rightarrow q_1 = \epsilon + q_1 (0 + 1(1 + 0 \cdot 1)^* 00)$

$R = Q + RP$

$R = QP^*$

$R = \epsilon (0 + 1(1 + 01)^* 00)^*$

Context free grammar & Context free Language

Module

In context free grammar, basically we have 4 types of tuples i.e,

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

where, V = finite set of variable
T = finite set of terminal
P = production rules
S = Start symbol

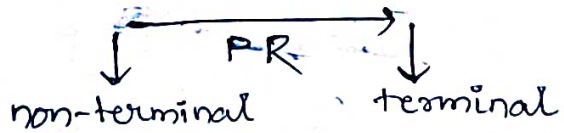
→ Non-terminal are always represent in the form of capital letters and terminal always represent in the form of small letters.

→ In production rule, mostly we are using the Substitution method.

→ Start symbol always present at the left side element of the first production rule.

→ The production rule always represented in an ($\longrightarrow$).

→ Left side element of the production rule is called as non-termial and right side is terminal.

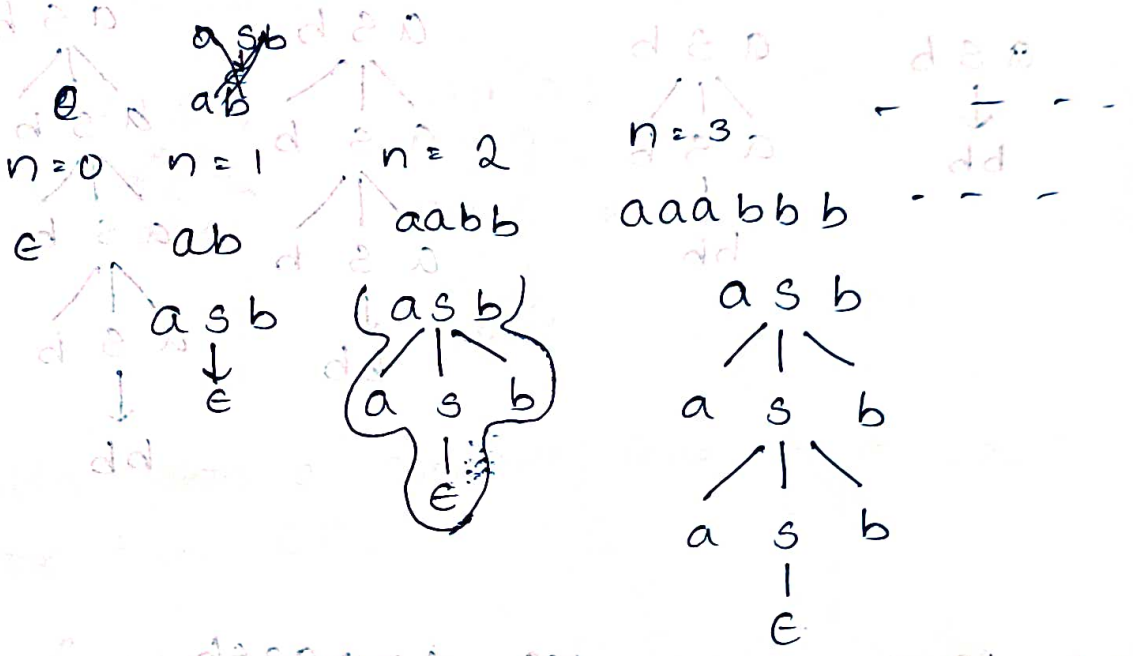


→ Right side combination of terminal and non-terminals are also possible.

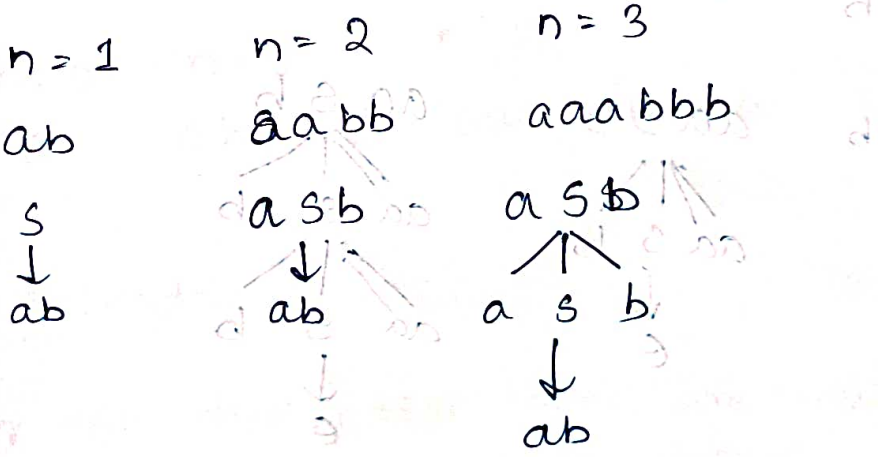
Q/ Construct a context free grammar (CFG) for the CFL $a^n b^n$, $n \geq 0$.

$a^n b^n$, $n \geq 0$ $\xrightarrow{CFG}$ $S \rightarrow asb$

$S \rightarrow \epsilon$



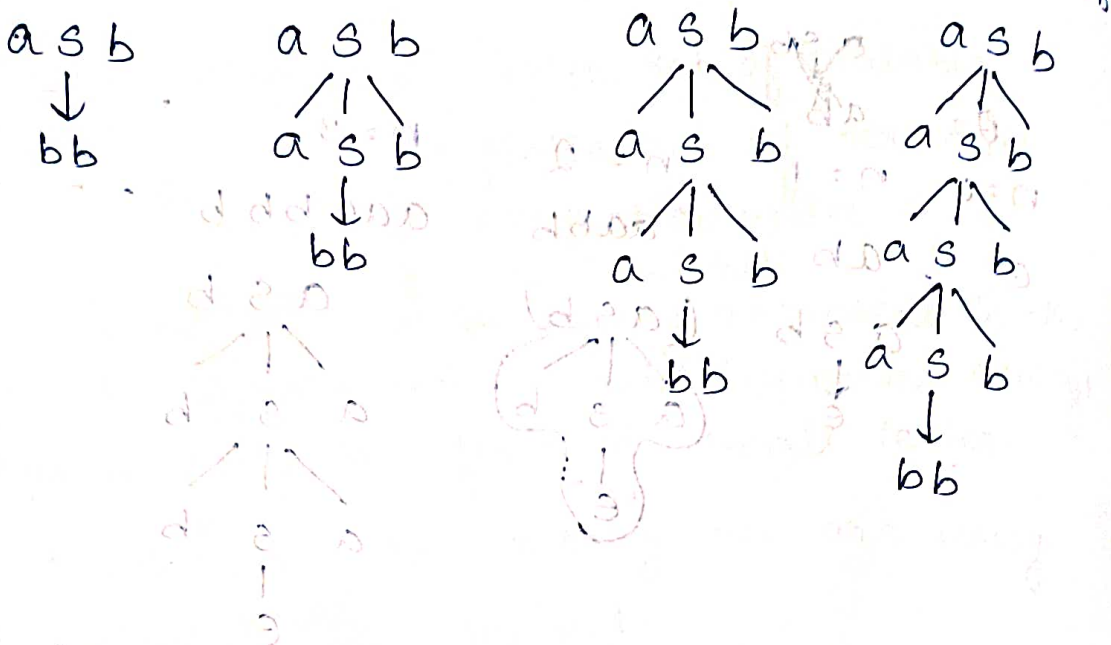
ii) $a^n b^n$, $n \geq 1$ $\xrightarrow{CFG}$ $S \rightarrow asb$, $S \rightarrow ab$



iii) $a^n b^{n+2}$, $n \geq 1$ $\xrightarrow{CFG}$ $S \rightarrow aSb$
$S \rightarrow bb$

$n = 1$ $n = 2$ $n = 3$ $n = 4$

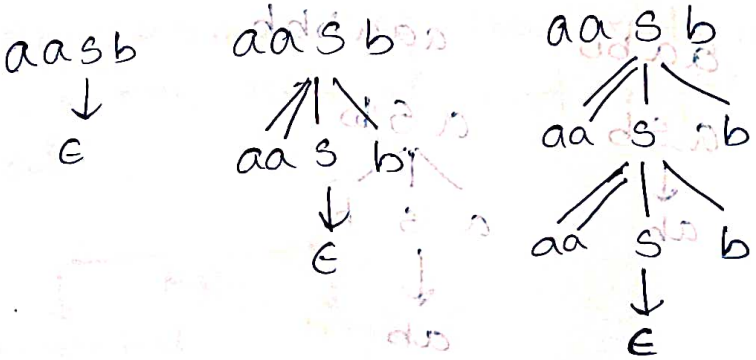
abbb aabbbb aaabbbbb aaaabbbbbb



iv) $a^{2n} b^n$, $n \geq 0$ $\xrightarrow{CFG}$ $S \rightarrow aasb$
$S \rightarrow \epsilon$

$n = 0$ $n = 1$ $n = 2$ $n = 3$

ϵ aab aaaabb aaaaaabbb



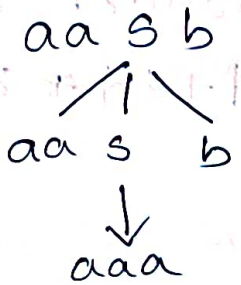
v) $a^{2n+3} b^n$, $n \geq 0$

CFG ⟹ $S \rightarrow aasb$
$S \rightarrow aaa$

$n = 0$ $n = 1$ $n = 2$

ϵ aaaaab aaaaaaabb
aaa

aa S b
aaa



aaa

Q/ Write down a content free grammar for the CFL

$$\{w \mid n_a(w) = n_b(w)\}$$

$S \rightarrow asb / bsa$
$S \rightarrow \epsilon$

$n = 0$ $n = 1$ $n = 2$ $n = 3$

ϵ ab/ba aabb/bbaa aaabbb/bbbaaa

Derivation Tree

For the derivation tree we must follow some steps.

1. In derivation tree the leaf node are alway represent by the terminals.
2. Root of the tree is always represent by the start symbol.
3. The interior nodes are always represent by

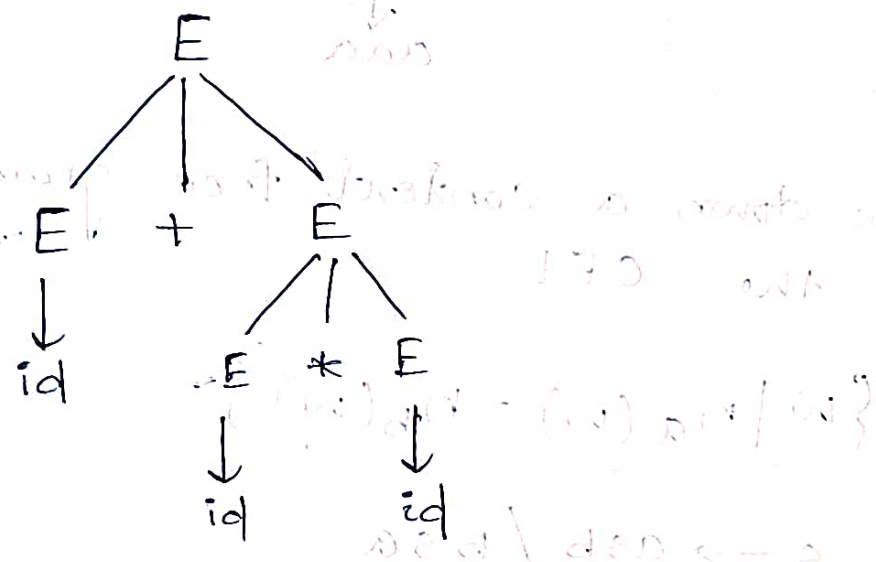
the non-terminals.

Q/ find out the derivation tree for the grammar

$$E \longrightarrow E + E$$
$$E \longrightarrow E * E$$
$$E \longrightarrow id$$

which will produce the string

$$w = id + id * id$$



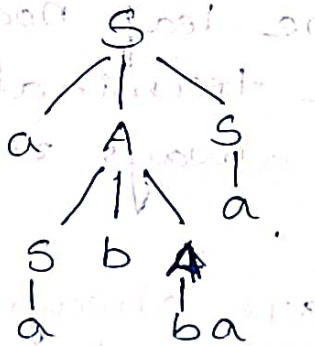
Q/ Construct a derivation tree for the grammar

$$S \longrightarrow aAS \mid a \quad \begin{matrix} S \to aAS \\ S \to a \end{matrix}$$

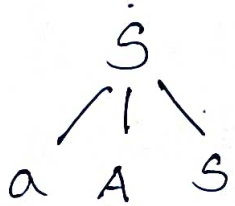
$$A \longrightarrow SbA \mid SS \mid ba \quad \begin{matrix} A \to SbA \\ A \to SS \\ A \to ba \end{matrix}$$

which will produce the string

$$w = aabbaa$$



$w = aabbaa$

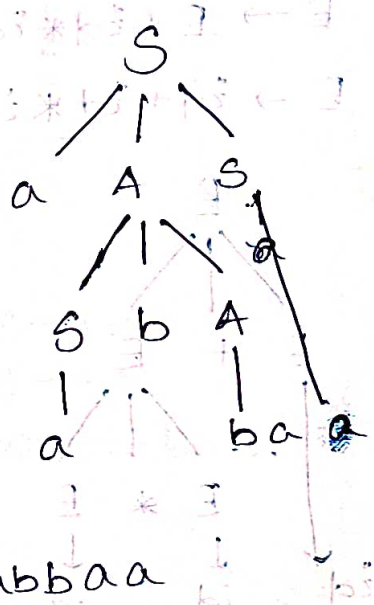


Q/ Write down a left most derivation tree and right most derivation tree for the string aabbaa for the grammar.

$S \rightarrow aAS \mid a$

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

Leftmost tree



$S \rightarrow aAS$

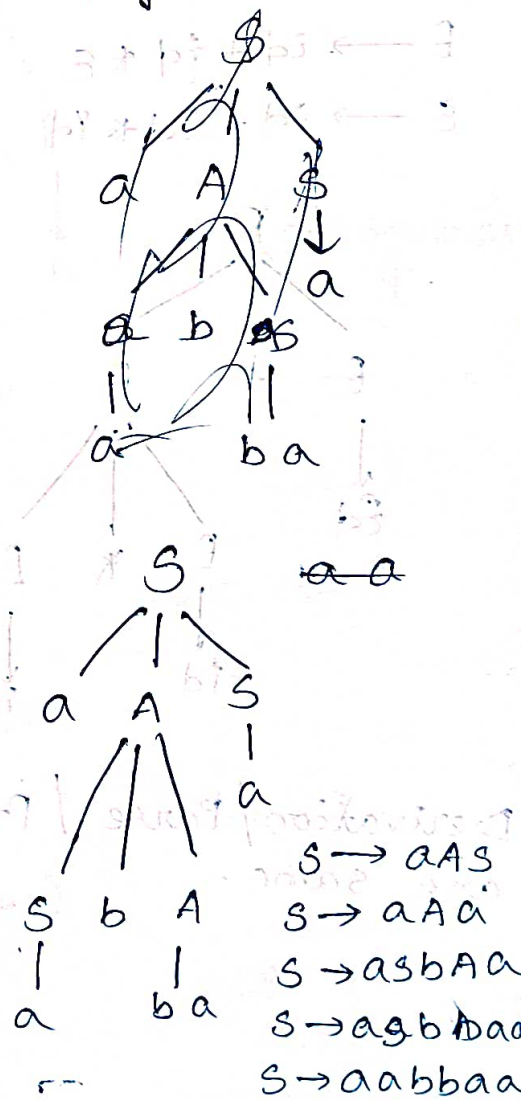
$S \rightarrow aSbAS$

$S \rightarrow aabAS$

$S \rightarrow aabbaS$

$S \rightarrow aabbaa$

Right most tree



$S \rightarrow aAS$

$S \rightarrow aAa$

$S \rightarrow asbAa$

$S \rightarrow aSbbaa$

$S \rightarrow aabbaa$

Q) Find out the left most and right most parse the for the given ~~parenthesis~~ grammer whose Production rules are

$E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow id$

$w = id + id * id$

In left most derivation tree we always put the value of terminal from left to right and In right most derivation tree we always put the value of terminal from right to left.

Left most

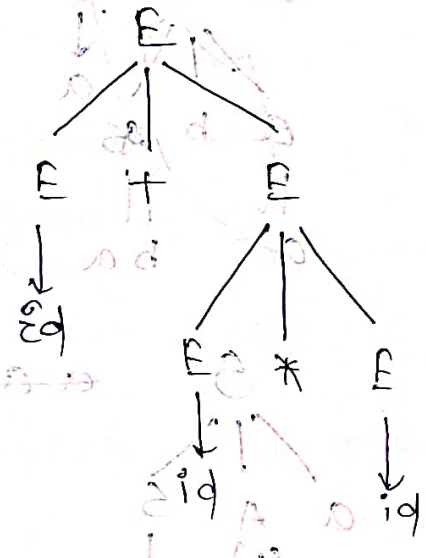
$E \rightarrow E+E$

$E \rightarrow id+E$

$E \rightarrow id+E*E$

$E \rightarrow id+id*E$

$E \rightarrow id+id*id$



Right most

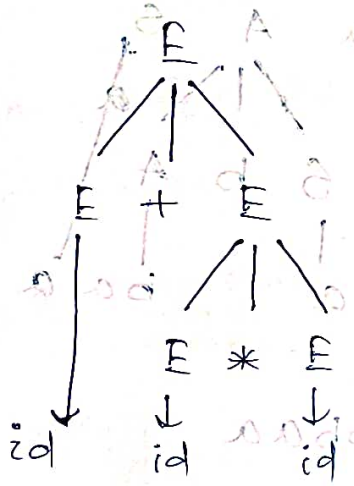
$E \rightarrow E+E$

$E \rightarrow E+E*E$

$E \rightarrow E+E*id$

$E \rightarrow E+id*id$

$E \rightarrow id+id*id$



Derivation / Parse / Production / Generation tree all are same.

Ambiguity in Grammar:

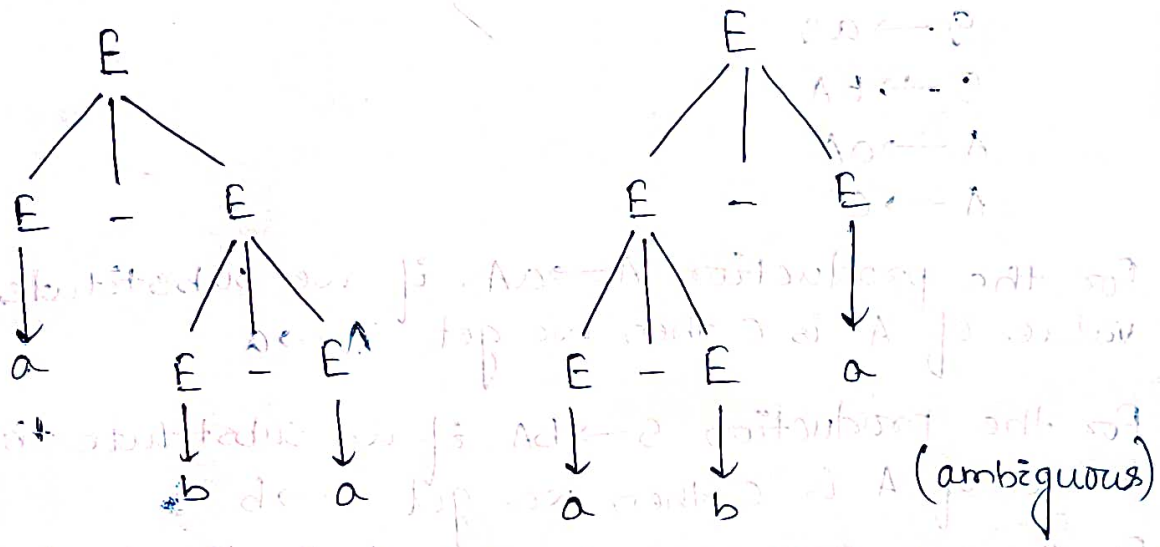
In ambiguity mainly we check our grammar is ambiguous or unambiguous.

A grammar is said to be ambiguous if more than one derivation tree is possible for the given string. If more than one derivation tree is not possible then we can say our grammar is unambiguous.

Q/ check the grammar is ambiguous or unambiguous for the string $\boxed{w = a-b-a}$ for the grammar whose production rules $E \rightarrow a/b$

$E \rightarrow E-E$

Ans:

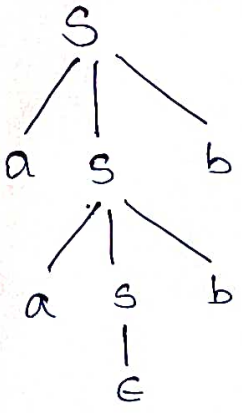


(ambiguous)

ii) $S \rightarrow a\,S\,b$

$S \rightarrow \epsilon$

$w = aabb$



(unambiguous)

Simplification of Context Free Grammar :

In simplification we mostly used in 3 steps :

Step 1 : Elimination of unit production

Step 2 : Elimination of ϵ production

Step 3 : Elimination of useless production

Elimination of ϵ production :

Q/ Reduce the following grammar such that there will be no ϵ production.

$S \rightarrow aS / bA$

$A \rightarrow aA / \epsilon$

$S \rightarrow aS$

$S \rightarrow bA$

$A \rightarrow aA$

$A \rightarrow \epsilon$

For the production $A \rightarrow aA$. if we substitute the value of A is ϵ then we get $A \rightarrow a$

For the production $S \rightarrow bA$ if we substitute the value of A is ϵ then we get $S \rightarrow b$

Finally my grammar consist of the production are $S \rightarrow aS$, $S \rightarrow b$, $A \rightarrow a$

Q2/. From the above grammar elimination of ϵ production.

$S \rightarrow AaB / aaB$

$A \rightarrow \epsilon$

$B \rightarrow bbA / \epsilon$

$S \rightarrow AaB$

$S \rightarrow aaB$

$A \rightarrow \epsilon$

$B \rightarrow bbA$

$B \rightarrow \epsilon$

For the production $B \rightarrow bbA$ if we substitute the value of A with ϵ then we get $B \rightarrow bb$

For the production $S \rightarrow aaB$ if we substitute the value of B with ϵ then we get $S \rightarrow aa$

For the production $S \rightarrow AaB$ if we substitute the value of A & B with ϵ then we get $S \rightarrow a$

After minimization my context free grammar consist of the production are

$S \rightarrow Aab$ $S \rightarrow aaB$ $S \rightarrow bbA$

$B \rightarrow bb$ $S \rightarrow aa$ $S \rightarrow a$

Elimination of useless Symbol : Doubt

A variable or non terminal is known as useless symbol in grammar if it is not connected with any terminal or it cannot be reached from the starting symbol.

G_1:

$S \rightarrow aAa$

$A \rightarrow b$

non-terminal = A

Connect to any terminal = Yes

Reachable from starting Symbol = Yes

G_2:

$S \rightarrow aAa / B$

$A \rightarrow b$

nonterminal = A B

G_3.

$S \rightarrow aAa$

$A \rightarrow bBB$

$B \rightarrow ab$

$C \rightarrow ab$

non terminal = A B C

G_4

$S \rightarrow a\$ / A / C$

$B \rightarrow aa$

$A \rightarrow a$

$C \rightarrow aCb$

non-terminal = A B C

Grammar : $S \rightarrow aAa \mid C$

$A \rightarrow B$

$C \rightarrow d$

	A	B	C
Non-terminal			
Connect to terminal	N	N	Y
Reachable from Starting symbol	Y ↓ Useless	Y ↓ Useless	Y ↓ Usefull

Eliminate the useless symbol from grammar

$S \longrightarrow a A a$

$A \longrightarrow b B B$

$B \longrightarrow a b$

$C \longrightarrow ab$

	A	B	C
Nonterminal			
Connected to terminal	yes	yes	yes
Reachable from Starting symbol	yes	yes	No ↓ useless

$A \rightarrow bBB$
$(babab)$
all are terminal so yes.

From this grammar we get the non-terminal 'C' is useless symbol so, after eliminating this useless symbol our new production rules are :

$S \longrightarrow aAa$

$A \longrightarrow bBB$

$B \longrightarrow ab$

ii) $S \rightarrow a\$ / A / C$

$B \rightarrow aa$

$A \rightarrow a$

$C \rightarrow aCb$

Non terminal	A	B	C
Connected to terminal	Y	Y	N
Reachable from starting symbol.	Y	N	Y

From the given grammar we get two useless symbol B, C.

After eliminating the useless grammar our new production rules are:

$S \rightarrow a\$ / A$

$A \rightarrow a$

Elimination of unit Production:

Q/ Eliminate the unit production for given grammar

$S \rightarrow A / bb$

$A \rightarrow B / b$

$B \rightarrow S / a$

From this grammar, the unit production are:

$S \rightarrow A$

$A \rightarrow B$

$B \rightarrow S$

After eliminating the unit production $S \rightarrow A$ we get $S \rightarrow b$

After eliminating the unit production $S \rightarrow A$ again we get $S \rightarrow a$ $(S \rightarrow A \rightarrow B \rightarrow a)$

After eliminating the unit production $A \rightarrow B$ we get $A \rightarrow a$

After eliminating the unit production $A \rightarrow B$ we again get $A \rightarrow bb$ $(A \rightarrow B \rightarrow S \rightarrow bb)$

After eliminating unit production $B \rightarrow S$, we get $B \rightarrow bb$

After eliminating unit production $B \rightarrow S$, we get $B \rightarrow b$

After eliminating the unit productions, our new production rules are:

$S \rightarrow bb / a / b$

$A \rightarrow b / a / bb$

$B \rightarrow a / b / bb$

Q/ Simplify the given grammar $S \rightarrow AaB / aaB$

$A \rightarrow \epsilon$

$B \rightarrow bbA / \epsilon$

Step 1:

For the production $B \rightarrow bbA$, we get $B \rightarrow bb$

$S \rightarrow AaB$, we get $S \rightarrow a$

$S \rightarrow AaB$, we get $S \rightarrow aA$

$S \rightarrow AaB$, we get $S \rightarrow aB$

$S \rightarrow aaB$, we get $S \rightarrow aa$

Now

$S \rightarrow AaB / aa / a / aA / aB$ ~~/ aaB~~

$S \rightarrow aaB$

$B \rightarrow bb / bbA$

Non terminal :	A	B
Connected terminals	N	Y
Reachable Start :	Y	Y

from the given grammar we get useless symbol A.

After eliminating the useless grammar our new production rules are:

$S \rightarrow aa \mid a \mid aB \mid aaB$

$B \rightarrow bb$

Normal Form Of CFG:

Basically we are using two normal form of CFG:

i) Chomsky's NF (CNF)

ii) Greibach NF (GNF)

Chomsky's NF:

for constructing a CNF mostly we are using 2 steps:

Step 1: Eliminate the null production and unit production.

Step 2: Convert the CFG in the form of CNF i.e, every production rule we convert in the form

$A \rightarrow BC$

$A \rightarrow a$

Q/ Construct a CNF for the given grammar

$S \rightarrow ASb$

$S \rightarrow BC_b$

$B \rightarrow AS$ or

$C_b \rightarrow b$

Q/ Construct a CNF for given grammar

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b$
$C \rightarrow FCa$

$C \rightarrow DA$
$D \rightarrow FC$

CNF :

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b$
$C \rightarrow HG$
$G \rightarrow a$
$H \rightarrow FC$

Q/ Construct a CNF for the given CFG

i) $S \rightarrow aAD$

CNF :

$S \rightarrow BC$
$B \rightarrow a$
$C \rightarrow AD$

ii) $S \rightarrow aAD$
$A \rightarrow aB$
$B \rightarrow b$
$D \rightarrow d$

CNF :

$S \rightarrow H_aG$
$H_a \rightarrow a$
$G \rightarrow AD$
$A \rightarrow H_aB$
$B \rightarrow b$
$D \rightarrow d$

Q/ Construct the CNF for the grammar given below:

$S \rightarrow aAD$

$A \rightarrow aB \mid bAB$

$B \rightarrow b$

$D \rightarrow d$

CNF:

$S \rightarrow C_aH$

$C_a \rightarrow a$

$H \rightarrow AD$

$A \rightarrow C_aB$

$A \rightarrow BE_bK$

$E_b \rightarrow b$

~~$E \rightarrow AB$~~ $K \rightarrow AB$

~~$D \rightarrow d$~~ $B \rightarrow b$

$D \rightarrow d$

Step 1: Eliminate the null production and unit production.

Step 2: Check whether our new production are in the form of CNF or not (after eliminating the null production & unit production).

Step 3: If the new productions are in the form of CNF change the name of non-terminal symbols into the form of A_i. in ascending order of i.

Step 4: If the non-terminals are in the ascending order then we check every production is in the form

$$A_i \longrightarrow A_j \text{ then } i < j$$

Step 5: Remove the recursion.

Q/ Find out the GNF for given grammar

$$S \longrightarrow CA$$
$$B \longrightarrow b$$
$$C \longrightarrow b$$
$$A \longrightarrow a$$

Step 1: Not necessary because there is no null production and unit production.

Step 2: CNF (our question is already in the form of CNF so there is no conversion)

Step 3:

I will Substitute:

S with A_1
C with A_2
A with A_3
B with A_4

] I rename the non terminals horizontally.

After appling this value my New production:

$$A_1 \longrightarrow A_2A_3$$
$$A_4 \longrightarrow b$$
$$A_2 \longrightarrow b$$
$$A_3 \longrightarrow a$$

Step 4 : If we consider the production $A_1 \rightarrow A_2A_3$ here the value of $i < j$ i.e, $i = 1$, $j = 2$

then no change.

But $A_1 \rightarrow A_2A_3$ production not in the form of GNF so we substitute the value of A_2 with b then finally our GNF form are $A_1 \rightarrow bA_3$

$A_4 \rightarrow b$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Q/ Find out the GNF for given grammar

$S \rightarrow CA \mid BB$

$B \rightarrow b$

$C \rightarrow b$

$A \rightarrow a$

Step 1 : Not necessary

Step 2 : CNF

Step 3 :

Substitute :

S with A_1

C with A_2

A with A_3

B with A_4

New production :

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow b$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Step 4 : If we consider the production $A_1 \rightarrow A_2 A_3$ here the value $i < j$ i.e, $i = 1$, $j = 2$ then no change. But $A_1 \rightarrow A_2 A_3$ production not in the form of GNF. So we substitute the value

$A_1 \rightarrow b A_3$

and Production $A_1 \rightarrow A_4 A_4$ also not in the form of GNF so we substitue the value $A_1 \rightarrow bA_4$

So finaly the GNF form are

$A_1 \rightarrow bA_3 / bA_4$

$A_4 \rightarrow b$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Q/ Find out the GNF for the grammar

$S \rightarrow CA / BB$

$B \rightarrow b$

$C \rightarrow \epsilon$

$A \rightarrow a$

Step 1 :

for the production $S \rightarrow CA$, if we substitute the value of C is ϵ then we get $S \rightarrow A$

So after eliminating we get, $S \rightarrow A / BB$

$B \rightarrow b$

$A \rightarrow a$

After eliminating unit production $S \rightarrow A$ we get

$S \rightarrow a$

New production :

$S \rightarrow a / BB$

$B \rightarrow b$

$A \rightarrow a$

Step 2 : In CNF

Step 3 :

Rename :

S with A_1

B with A_2

A with A_3

we get :

$A_1 \rightarrow a / A_2 A_2$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Step 4 :

$A_1 \rightarrow a / b A_2$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Push Down Automata

PDA is a mathematical model which consist of 7 no. of tuples

$$PDA = \{Q, \Sigma, q_0, F, \delta, \Gamma, Z_0\}$$

Q = finite set of state
Σ = input alphabet
q_0 = Initial state
F = final state
δ = transition
Γ = Stack or tape symbol
Z_0 = Stack start symbol (by default Z_0 present inside the stack).

→ In PDA basically we are using the concept of stack i.e, we are using push and pop operation.

→ PDA is of two types :

i) DPDA (Deterministic PDA)

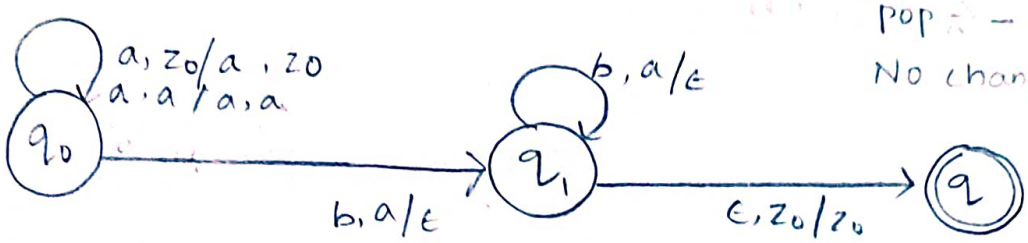
ii) NPDA (Nondeterministic PDA)

→ NPDA will accept set of all CFL but DPDA will not accept all CFL.

→ In PDA mostly we are using the concept of stack so representation of stack is

Q/ Design a PDA for the language $L = a^n b^n, n \geq 1$

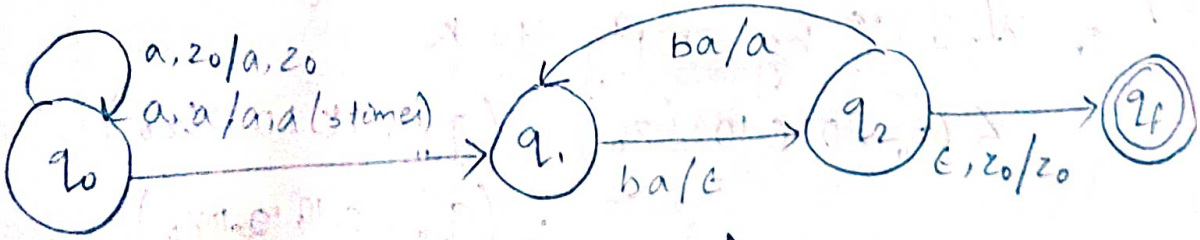
push ---/---
pop ---/ε
No change --/—



$\delta(q_0, a, z_0) = (q_0, a z_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, a, a) = (q_0, a, a)$
$\delta(q_0, a, a) = (q_0, a; a)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$

Q/ Construct a PDA for the lang. $L = a^n b^{2n}, n \geq 1$

$n = 2$ aabbbb
$n = 3$ aaabbbbbb
$n = 4$ aaaa bb b b b b b b



$\delta(q_0, a, z_0) = (q_0, a, z_0)$
$\delta(q_0, a, a) = (q_0, a, a)$
$\delta(q_0, a, a) = (q_0, a, a)$

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

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

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

$\delta(q_1, b$

Q/ Design a PDA which accept equal no. of a's and equal no. of b's if the string is abab.

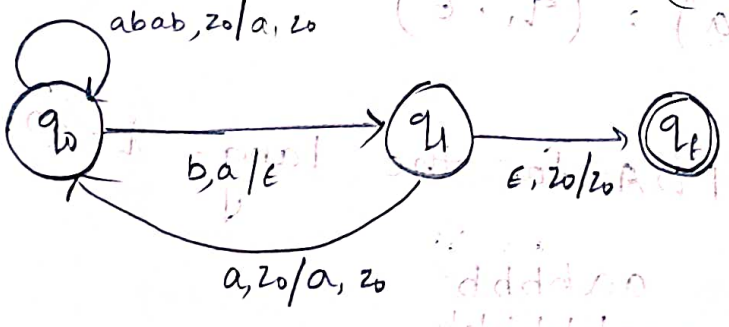
For the string abab the transitions are

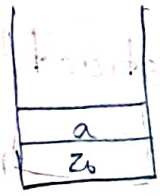
$\delta(q_0, abab, z_0) \Rightarrow (q_0, bab, az_0)$

$\delta($ $= (q_1, ab, z_0)$

$= (q_0, b, az_0)$

$= (q_1, \epsilon, z_0)$





Imp

Q/ Design a PDA that accept $\{wcw^R / w\}$ in $(0+1)^*$ by empty stack.

$\delta(q_0, 01c10, z_0) = (q_0, 1c10, 0z_0)$

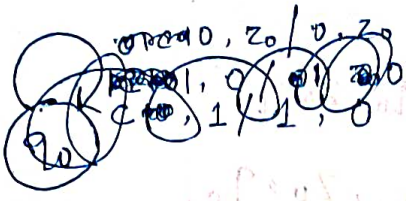
$= (q_0, c10, 10z_0)$

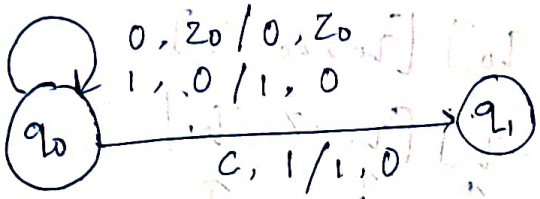
$= (q_0, 10, 10z_0)$

$= (q_1, 10, 0z_0)$

$= (q_1, \epsilon, z_0)$

$= (q_1, \epsilon, e)$





Contest Free Grammar to PDA :

Q/ construct a PDA for the grammar

1. $S \rightarrow aAA$
   $A \rightarrow aS$
   $A \rightarrow bS$
   $A \rightarrow a$

2. $S \rightarrow aA$
   $A \rightarrow aABD \mid a$

~~$B \rightarrow b$~~
~~$D \rightarrow d$~~

Step 2 :

GNF	PDA
$S \rightarrow aA$	$\delta(q_1, a, S) = (q_1, A)$
$A \rightarrow aABD$	$\delta(q_1, a, A) = (q_1, ABD)$
$A \rightarrow a$	$\delta(q_1, a, A) = (q_1, \epsilon)$

PDA to CFG

1. $\delta(q_0, a, z_0) = (q_0, az_0)$

$[q_0, z_0, q_0] = a\,[q_0, a, q_0]\,[q_0, z_0, q_0]$

$[q_0, z_0, q_0] = a\,[q_0, a, q_1]\,[q_1, z_0, q_0]$

$[q_0, z_0, q_1] = a\,[q_0, a, q_0]\,[q_0, z_0, q_1]$

$[q_0, z_0, q_1] = a\,[q_0, a, q_1]\,[q_1, z_0, q_1]$

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

$[q_0, a, q_0] = b\,[q_1, a, q_0]$

$[q_0, a, q_1] = b\,[q_1, a, q_1]$

3. $\delta(q_1, a, a) = (q_1, \epsilon)$

$[q_1, a, q_1] = a$

4. $\delta(q_1, \epsilon, a) = (q_1, \epsilon)$

$[q_1, a, q_1] = \epsilon$

5. $\delta(q_1, a, a) = (q_1, \epsilon)$

$[q_1, a, q_1] = a$

Ans:-

Q/ Construct PDA from the CFG if $M=\{\{p,q\},\{0,1\},\{q,Z\},\delta,\{x,z\}\}$ (Q, Σ, q_0, Z_0, trans)

where

1. $\delta(q,1,z)=(q,xz)$
2. $\delta(q,1,x)=(q,x)$
3. $\delta(p,x,x)=(p,\epsilon)$
4. $\delta(q,\epsilon,x)=(q,\epsilon)$

~~1. $\delta(q,1,z)=(q,xz)$~~

$[q,1,z]$

1. $\delta(q,1,z)=(q,xz)$

$[q,z,p]=1[q,x,p][p,z,p]$

$[q,z,p]=1[q,x,q][q,z,p]$

$[q,z,q]=1[q,x,p][p,z,q]$

$[q,z,q]=1\cdot[q,x,q][q,z,q]$

2. $\delta(q,1,x)=(q,x)$

$[q,x,q]=1[q,x,q]$

$[q,x,p]=1[q,x,p]$

3. $\delta(p,x,x)=(p,\epsilon)$

$[p,x,p]=x$

4. $\delta(q,\epsilon,x)=(q,\epsilon)$

$[q,x,q]=\epsilon$

Turing Machine:

Turing machine consist of 7 no. of Tuples

$$TM = (Q, \Sigma, \delta, q_0, F, \Gamma, B)$$

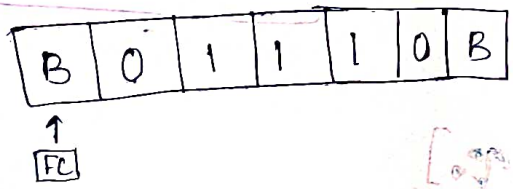
→ Blank symbol is present starting pos of string and ending pos of string.

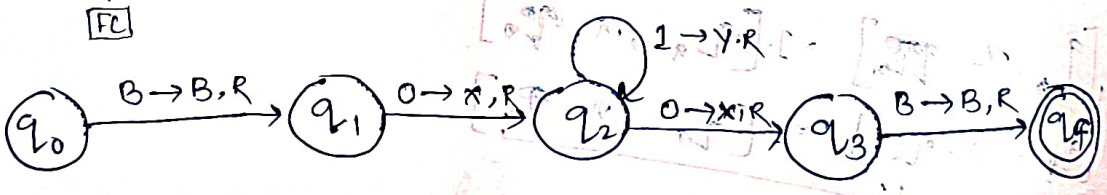
→ In turing machine we are using both read and write operation for this read and write operation mainly we are using left and right move.

Q/ Design a turing machine which accept the lang. $L = 01^*0$

R-right move

$01^*0 = 01110$



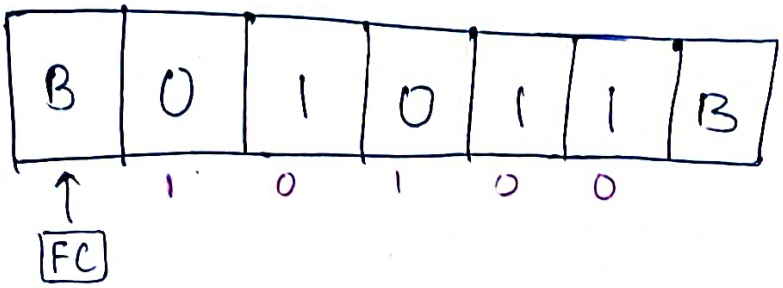


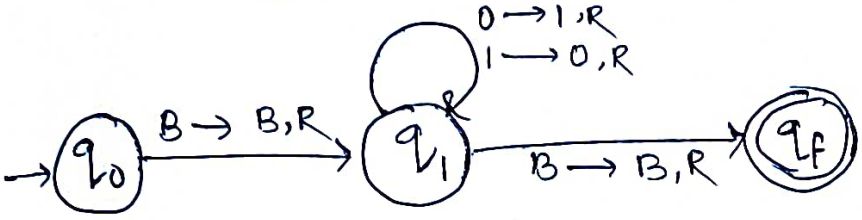
The turning machine is in the form of halt when it reaches the final state or when it gets the last blank space.

$$\delta(q_0, B) = (q_1, B, R)$$
$$\delta(q_1, 0) = (q_2, X, R)$$
$$\delta(q_2, 1) = (q_2, Y, R)$$
$$\delta(q_2, 0) = (q_3, X, R)$$
$$\delta(q_3, B) = (q_f, B, R)$$

Q/ Design a turing machine to compute 1's complement of a string.

L = 01011





$\delta(q_0, B) = (q_1, B, R)$

$\delta(q_1, 1) = (q_1, 0, R)$

$\delta(q_1, 1) = (q_1, 0, R)$

$\delta(q_1, 0) = (q_1, 1, R)$

$\delta(q, B) = (q_f, B, R)$

Q/ design a turing machine which compute 2's complement of a string

I/p

B	0	1	1	0	B

1's

B	1	0	0	1	B