Theory of Computation (TOC)

* TOC deals with whether and how efficiently problems can be solved on a model of computation using an algorithm.
* It deals understanding what problems can be solved by computers, how efficiently they can be solved, & what are the limits of computation. It explores the fundamental questions about what can & cannot be computed.

Kinds of TOC

(1) **Computability theory:-**

This theory is given by Godel, Church, Turing, Kleen & Post.
This theory used is to determine which problems are computable (solvable) & wih are not - computable (unsolvable).
It is also known as recursive theory.

(2) **Computationed Complexity theory:-**

This theory deal with determining the complexity of the problem during computation of a problem is efficiently solvable, then it is "computationally easy" and if it can not be solved efficiently then it is computationally hard.

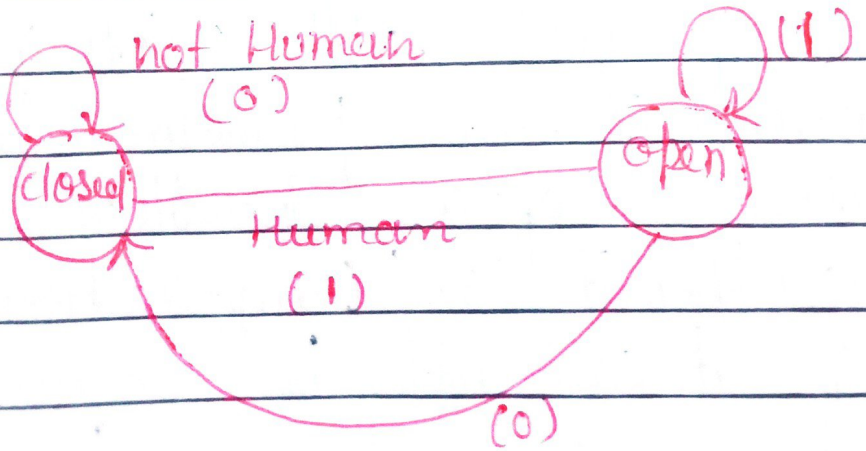
(3) Automata Theory:- It deals with designing a self-propelled computing devices (Automation) that follows a predetermined sequence of operations automatically.

Automation:- An automation is a mathematical model of any computing device / Machine that performs computations of an input by moving through a series of states.

* At each state of computation, a transition function determines the next state based on present state & input.

* Once the computation reaches an accepting state, it accepts the input else rejects it.

Eg- Automata Machine.



Kinds of Automation:-

(1) Finite Automation (FA).
(2) Pushdown Automation (PDA).
(3) Linear Bounded Automation (LBA)
(4) Turing Machine (TM).

Kleene Closure

(i) Kleene star closure (Σ^*)

The set of all possible strings over an alphabet Σ including null string (ε or λ)

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \text{----}$$

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

Σ^0 → Strings of Length 0 → $\{\lambda\}$

Σ^1 → $\{0, 1\}$

$\{00, 11, 10, 01\}$ - - -

(ii) Kleene Positive closure (Σ^+)

It is similar to Kleene star closure excluding null string (λ)

$$\Sigma^+ = \Sigma^* - \{\lambda\}$$

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

$$\Sigma^+ = \{0, 1, 00, 01, 10, 11 \ldots\}$$

Language [L]:- A language is a subset of strings over an alphabet Σ following specific set of rules. It is represented by L.

* If Σ is an alphabet, then L is a subset of Σ^* following certain rules(s)

Eg:- Let $\Sigma = \{0, 1\}$ then

$L = \{\omega \in \{0, 1\}^* \mid \omega$ has equal no. of 0's & 1's$\}$

$L = \{\Lambda, 01, 10, 0011, 1100, 0101, 0110, 1010, \ldots\}$

Eg:- Let $\Sigma = \{a, b, c\}$ then ω has strings starting with 'a' ending with 'c'.

$L = \{ac, abc, aabc, aac, abcc, abbc \ldots$
$\ldots abcac, aabbcc \ldots\}$

Application of finite Automata.

1. designing the lexical analysis of compiler (FA)
2. design & verification of hardware digital circuits (FA)
3. used in text editors (FA)
4. for the implementation of spell character
5. Pattern recognition (FA)
6. designing the syntax analysis of compiler (PDA)
7. for implementation of genetic programming (LBA)
8. for implementation of neural networks (TM)
9. for implementation of robotics applications (TM)
10. for implementation of artificial intelligence (TM)

* Additional application of finite Automata.

=> Automatic photo printing machine.
=> Card punching machine
=> Automatic traffic lights.
=> Aircraft control.
=> space craft
=> Vending Machine
=> Elevators.

After going through this chapter, you should be able to understand :

- Alphabets, Strings and Languages
- Mathematical Induction
- Finite Automata
- Equivalence of NFA and DFA
- NFA with ϵ - moves

1.1 ALPHABETS, STRINGS & LANGUAGES

Alphabet

An alphabet, denoted by Σ, is a finite and nonempty set of symbols.

Example:

1. If Σ is an alphabet containing all the 26 characters used in English language, then Σ is finite and nonempty set, and $\Sigma = \{a, b, c, \ldots, z\}$.
2. $X = \{0,1\}$ is an alphabet.
3. $Y = \{1,2,3,\ldots\}$ is not an alphabet because it is infinite.
4. $Z = \{\}$ is not an alphabet because it is empty.

String

A string is a finite sequence of symbols from some alphabet.

Example :

"*xyz*" is a string over an alphabet $\Sigma = \{a, b, c, \ldots, z\}$. The empty string or null string is denoted by ϵ.

Length of a string

The length of a string is the number of symbols in that string. If w is a string then its length is denoted by $|w|$.

Example :

1. $w=abcd$, then length of w is $|w|= 4$
2. $n = 010$ is a string, then $|n|= 3$
3. ϵ is the empty string and has length zero.

The set of strings of length K (K ≥ 1)

Let Σ be an alphabet and $\Sigma = \{a, b\}$, then all strings of length K ($K \geq 1$) is denoted by Σ^K.

$\Sigma^K = \{w : w \text{ is a string of length } K, K \geq 1\}$

Example:

1. $\Sigma = \{a,b\}$, then

   $\Sigma^1 = \{a,b\}$,

   $\Sigma^2 = \{aa, ab, ba, bb\}$,

   $\Sigma^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

   $|\Sigma^1|= 2 = 2^1$ (Number of strings of length one),

   $|\Sigma^2|= 4 = 2^2$ (Number of strings of length two), and

   $|\Sigma^3|= 8 = 2^3$ (Number of strings of length three)

2. $S = \{0,1,2\}$, then $S^2 = \{00,01,02,11, 10,12,22,20,21\}$, and $|S^2|= 9 = 3^2$

Concatenation of strings

If w_1 and w_2 are two strings then concatenation of w_2 with w_1 is a string and it is denoted by w_1w_2. In other words, we can say that w_1 is followed by w_2 and $|w_1w_2| = |w_1| + |w_2|$.

Prefix of a string

A string obtained by removing zero or more trailing symbols is called prefix. For example, if a string $w = abc$, then a, ab, abc are prefixes of w.

Suffix of a string

A string obtained by removing zero or more leading symbols is called suffix. For example, if a string $w = abc$, then c, bc, abc are suffixes of w.
A string a is a proper prefix or suffix of a string w if and only if $a \neq w$.

Substrings of a string

A string obtained by removing a prefix and a suffix from string w is called substring of w. For example, if a string $w = abc$, then b is a substring of w. Every prefix and suffix of string w is a substring of w, but not every substring of w is a prefix or suffix of w. For every string w, both w and $\in$ are prefixes, suffixes, and substrings of w.
Substring of w = w − (one prefix) − (one suffix).

Language

A Language L over Σ, *is a subset of* Σ^*, *i. e., it is a collection of strings over the alphabet* Σ. ϕ, and $\{\in\}$ are languages. The language ϕ is undefined as similar to infinity and $\{\in\}$ is similar to an empty box i.e. a language without any string.

Example:

1. $L_1 = \{01, 0011, 000111\}$ is a language over alphabet $\{0,1\}$
2. $L_2 = \{\in, 0, 00, 000, ...\}$ is a language over alphabet $\{0\}$
3. $L_3 = \{0^n 1^n 2^n : n \geq 1\}$ is a language.

Kleene Closure of a Language

Let L be a language over some alphabet Σ. Then Kleene closure of L is denoted by L^* and it is also known as reflexive transitive closure, and defined as follows :

L^* = {*Set of all words over* Σ}

= {word of length zero, words of length one, words of length two, ...}

$= \bigcup_{K=0}^{\infty} (\Sigma^K) = L^0 \cup L^1 \cup L^2 \cup \ldots$

Example:

1. $\Sigma = \{a, b\}$ and a language L over Σ. Then

   $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$

   $L^0 = \{\epsilon\}$

   $L^1 = \{a, b\}$,

   $L^2 = \{aa, ab, ba, bb\}$ and so on.

   So, $L^* = \{\epsilon, a, b, aa, ab, ba, bb \ldots\}$
2. $S = \{0\}$, then $S^* = \{\epsilon, 0, 00, 000, 0000, 00000, \ldots\}$

Positive Closure

If Σ is an alphabet then positive closure of Σ is denoted by Σ^+ and defined as follows:

$\Sigma^+ = \Sigma^* - \{\epsilon\}$ = {*Set of all words over* Σ *excluding empty string* ϵ}

Example :

if $\Sigma = \{0\}$, then $\Sigma^+ = \{0, 00, 000, 0000, 00000, \ldots\}$

1. 2 MATHEMATICAL INDUCTION

Based on general observations specific truths can be identified by reasoning. This principle is called mathematical induction. The proof by mathematical induction involves four steps.

Basis : This is the starting point for an induction. Here, prove that the result is true for some n = 0 or 1.

Induction Hypothesis : Here, assume that the result is true for n = k.

Induction step : Prove that the result is true for some n = k + 1.

1.3 FINITE AUTOMATA (FA)

A finite automata consists of a finite memory called input tape, a finite - nonempty set of states, an input alphabet, a read - only head , a transition function which defines the change of configuration, an initial state, and a finite - non empty set of final states.

A model of finite automata is shown in figure 1.1.

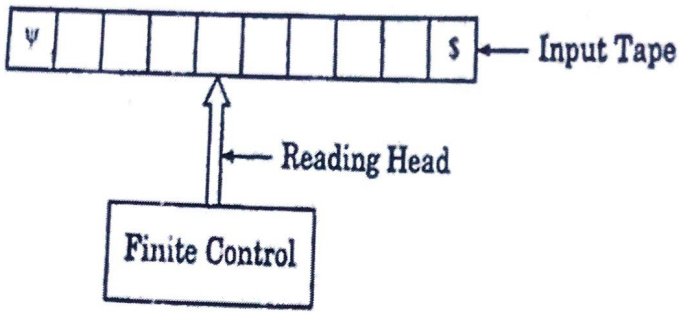


FIGURE 1.1 : Model of Finite Automata

The input tape is divided into cells and each cell contains one symbol from the input alphabet. The symbol 'ψ' is used at the leftmost cell and the symbol '$' is used at the rightmost cell to indicate the beginning and end of the input tape. The head reads one symbol on the input tape and finite control controls the next configuration. The head can read either from left - to - right or right - to -left one cell at a time. The head can't write and can't move backward. So , FA can't remember its previous read symbols. This is the major limitation of FA.

Deterministic Finite Automata (DFA)

A deterministic finite automata M can be described by 5 - tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is finite, nonempty set of states,
2. Σ is an input alphabet,
3. δ is transition function which maps $Q \times \Sigma \rightarrow Q$ i. e. the head reads a symbol in its present state and moves into next state.
4. $q_0 \in Q$, known as initial state
5. $F \subseteq Q$, known as set of final states.

Non - deterministic Finite Automata (NFA)

A non - deterministic finite automata M can be described by 5 - tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is finite, nonempty set of states,
2. Σ is an input alphabet,
3. δ is transition function which maps $Q \times \Sigma \rightarrow 2^Q$ i.e., the head reads a symbol in its present state and moves into the set of next state (s). 2^Q is power set of Q,
4. $q_0 \in Q$, known as initial state, and
5. $F \subseteq Q$, known as set of final states.

The difference between a DFA and a NFA is only in transition function. In DFA, transition function maps on at most one state and in NFA transition function maps on at least one state for a valid input symbol.

States of the FA

FA has following states :

1. **Initial state :** Initial state is an unique state ; from this state the processing starts.
2. **Final states :** These are special states in which if execution of input string is ended then execution is known as successful otherwise unsuccessful.
3. **Non - final states :** All states except final states are known as non - final states.
4. **Hang - states :** These are the states, which are not included into Q, and after reaching these states FA sits in idle situation. These have no outgoing edge. These states are generally denoted by ϕ. For example, consider a FA shown in figure1.2.

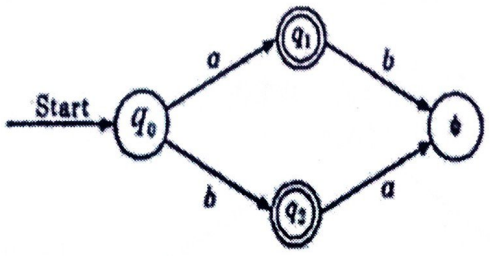


FIGURE 1.2 : Finite Automata

q_0 is the initial state, q_1, q_2 are final states, and ϕ is the hang state.

Notations used for representing FA

We represent a FA by describing all the five - terms $(Q, \Sigma, \delta, q_0, F)$. By using diagram to represent FA make things much clearer and readable. We use following notations for representing the FA:

1. The initial state is represented by a state within a circle and an arrow entering into circle as shown below:

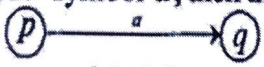
(Initial state q_0)

2. Final state is represented by final state within double circles:

(Final state q_f)

3. The hang state is represented by the symbol 'ϕ' within a circle as follows:

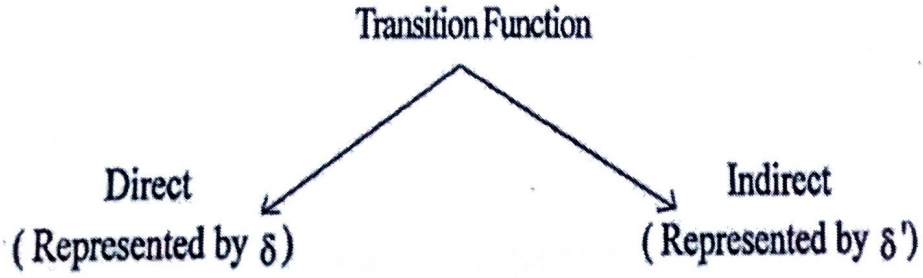
4. Other states are represented by the state name within a circle.
5. A directed edge with label shows the transition (or move). Suppose p is the present state and q is the next state on input - symbol 'a', then this is represented by



6. A directed edge with more than one label shows the transitions (or moves). Suppose p is the present state and q is the next state on input - symbols 'a_1' or 'a_2' or ... or 'a_n' then this is represented by

Transition Functions

We have two types of transition functions depending on the number of arguments.



Direct transition Function (δ)

When the input is a symbol, transition function is known as direct transition function.

Example : $\delta(p, a) = q$ (Where p is present state and q is the next state).
It is also known as one step transition.

Indirect transition function (δ')

When the input is a string, then transition function is known as indirect transition function.

Example : $\delta'(p, w) = q$, where p is the present state and q is the next state after | w | transitions. It is also known as one step or more than one step transition.

Properties of Transition Functions

1. If $\delta(p, a) = q$, then δ (p, ax) = δ(q, x) and if δ' (p, x) = q, then δ' (p, xa) = δ'(q, a)
2. For two strings x and y ; $\delta(p, xy) = \delta(\delta(p, x), y)$, and $\delta'(p, xy) = \delta'(\delta'(p, x), y)$

Example :1. A DFA $M = (\{q_0, q_1, q_2, q_f\}, \{0,1\}, \delta, q_0, \{q_f\})$ is shown in figure1.3 .

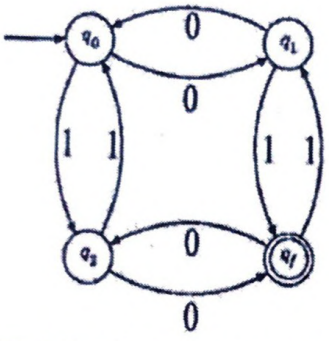


FIGURE 1.3 : Deterministic finite automata

Where δ is defined as follows :

		0	1
→	q_0	q_1	q_2
	q_1	q_0	q_f
	q_2	q_f	q_0
	q_f	q_2	q_1

2. A NFA $M_1 = (\{q_0, q_1, q_2, q_f\}, \{0,1\}, \delta, q_0, \{q_f\})$ is shown in figure1.4.

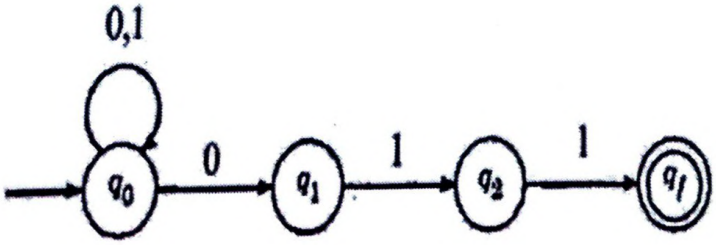
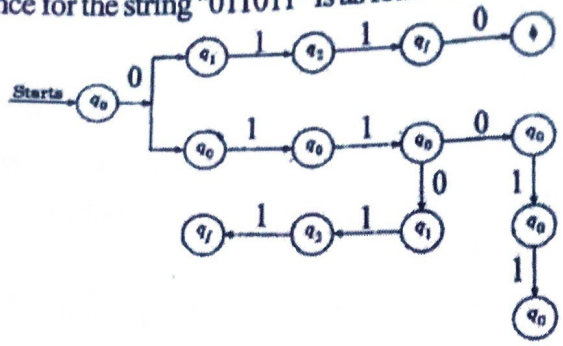


FIGURE 1.4 : Non - deterministic finite automata

3. Transition sequence for the string "011011" is as follows :



One execution ends in hang state ϕ, second ends in non - final state q_0, and third ends in final state q_f hence string "011011" is accepted by third execution.

Difference between DFA and NFA

Strictly speaking the difference between DFA and NFA lies only in the definition of δ. Using this difference some more points can be derived and can be written as shown :

DFA	NFA
1. The DFA is 5 - tuple or quintuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is set of finite states Σ is set of input alphabets $\delta : Q \times \Sigma$ to Q q_0 is the initial state $F \subseteq Q$ is set of final states	The NFA is same as DFA except in the definition of δ. Here, δ is defined as follows : $\delta : Q \times (\Sigma \cup \in)$ to subset of 2^Q
2. There can be zero or one transition from a state on an input symbol	There can be zero, one or more transitions from a state on an input symbol
3. No $\in$ – transitions exist i.e., there should not be any transition or a transition if exist it should be on an input symbol	$\in$ – transitions can exist i. e., without any input there can be transition from one state to another state.
4. Difficult to construct	Easy to construct

ation.

Languages

yes

Que |S|=n & all symbols are different how many substring is possible?

$\Sigma = \{a, b, c, d, e, \}$

S = abc

Length 0 → ∈ — (1)

1 → a, b, c = 3.

2 → ab, bc, ac = 3.

3 → abc = 1.

= (7)

Length 0 → 1 ✓

" " 1 → n ✓

" " 2 → n-1 = 3-1 = 2

" " 3 → n-2 = 3-2 = 1.

" " 4 → n-3

(n-1) → n-2

" " n → n-(n-1)

$\boxed{\frac{n(n+1)}{2} + 1}$ including null.

$\boxed{\frac{n(n+1)}{2}}$ except null

Que |S|=n & all symbols are different how many substrings.

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

S = aaa

L = 0 ⇒ 1

1 = 1

2 = 1

$\boxed{n+1}$

(aab)
aabaabcaab

Qu How many sub strings of length 'n' can be made 'K' different symbols.

$|\Sigma| = K$

$\Sigma = \{a, b, c\}$ $K = 3$

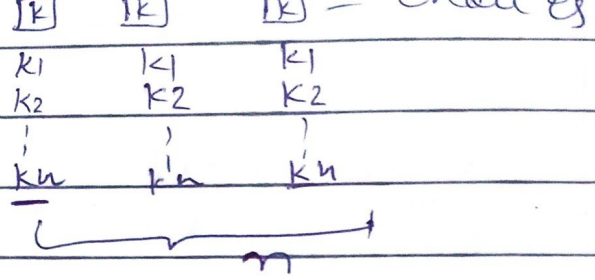
Let $n = 2$

$3 \times 3 = 9$

↓ choices Here

↓ choices Here

3 choices



K^n choices

* OPERATIONS ON STRINGS :-

1. Power of string
2. Concatination
3. Reversal
4. Prefix
5. Sufix

Power of string $w^n \mid n \geq 0$

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

$w = ab$ $(ab)^0 = \epsilon$

$w^1 = (ab)^1 = 1 = (a, b)$

$w^2 = (ab)^2 = abab$ . . . .

* Why $w^0 = \epsilon$ $w = ab$

bcoz $w^0 \cdot w^2 = w^{2+0} = w^2 = abab$

↓

$\epsilon \cdot abab = \boxed{abab}$

$\Sigma^+ = \Sigma^* - (\epsilon)$ $\Sigma = \{a, b\}$ $\{a, b, c\}$

~~Concatination~~: * Power of Σ

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

$\Sigma^0 = \{a, b\}^0 = \{\epsilon\}$

$\Sigma^1 = \{a, b\}$

$\Sigma^2 = \{a, b\}^2 = \{aa, ab, ba, bb\}$

$\Sigma^3 = \{a, b\}^3 = \{ab\} * \{ab\} * \{ab\} = \{aaa, aab, \ldots bbb\}$

Σ^* , $\Sigma^+ \rightarrow \{\Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\} = \{a, b, aa, bb, ab, ba, \ldots\}$

$\{\Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \ldots\} = \{\epsilon, a, b, aa, bb, ab, ba \ldots\}$

Universal Language

(1) * Σ^* is a Language which contains all possible symbol over Σ.

(2) * Σ^* is the Largest possible Language over Σ & hence we call it a Universal Lang.

(3) Any Language over Σ will be subset of Σ^*

$L \subseteq \Sigma^*$

Smallest Language = ϕ

Largest " " = Σ^*

2^{Σ^*} uncountably infinite set

(4) Σ^* & Σ^+ both are infinite Languages.

* string always finite Length

* no of string infinite

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

Σ^* always contain NULL

$\Sigma^* \Rightarrow \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \ldots$

Σ^+ never contain NULL

Σ^* → Kleen Closure OR Star Closure
Σ^+ → positive closure

DATE

* **Concatination:-**

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

$u = ab$

$v = ba$

$u \cdot v = abba$

$v \cdot u = baab$

$\{u \cdot v \neq v \cdot u\}$ it is not commutative

$(u \cdot v)^R = v^R \cdot u^R$

Prob consider a string $u \cdot v$ string $u \in \Sigma$ & $v \in \Sigma^*$, $|u \cdot v| = ??$

(a) $1 + |v|$ (c) Both

(b) $|u| + |v|$ (d) none

\# Reverse.

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

$u = 110$ $u^R = 011$

$\{u \neq u^R\}$

Abstract computing device means, the computing model, that allows for a detailed & precise analysis of how computer system function. It is similar to mathematical function, in that receive

Palindrom

$u\ u^R \mid u \in \Sigma^*$

odd Length	even Length
0	ϵ
101	00
000	11
111	1001
010	0110

$\{u \times u^R\}$

यहाँ छोटा even Length palindrom

$|\Sigma|^{\frac{n+1}{2}}$ for odd

$|\Sigma|^{\frac{n}{2}}$ for even

inputs & produces outputs based on predefined rules

* It consists of a finite memory.
* It is most restricted Mathematical Model of computing device.

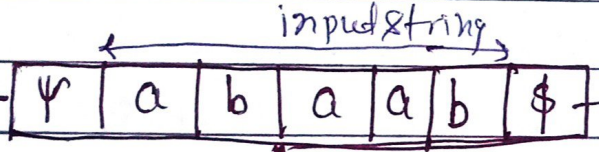
Finite Automata (FA) → Type 3 - G

* FA is an abstract computing device
* It is a Mathematical model of a system with discrete inputs, outputs States & set of transition from state to state, that occurs on input symbols from alphabet Σ.
* It is most restricted Automaton

Its representations:

* Graphical { Transition Diagram }
* Tabular { " " Table }
* Mathematical { " " function or Mapping fun.

* used in String Matching
* used in Lexical analysis phase of compiler
* pattern matching,
* text editors & verification of H/W digital circuit.



Input tape :- tape is divide into squares, each square consists of single symbol from the input alphabet.

Reading head :- is used to read symbol from the i/p Tape
- read one symbol at a time.

⇒ read left to right.

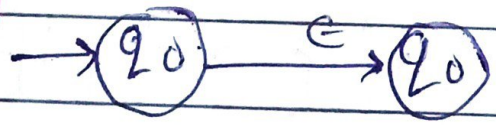
Empty transition:- NFA can also have ϵ-transitions or empty transitions. These transitions allow the NFA to move from one state to another without consuming any input.

* String accepted by DFA

1. Starting from initial state
2. after reading entire i/p string
3. DFA is anyone of the final state.
4. Then we say that string is accepted.

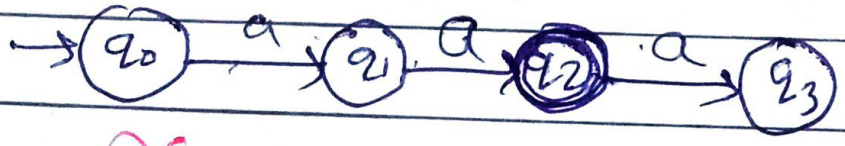
$\Sigma = \{a\}$

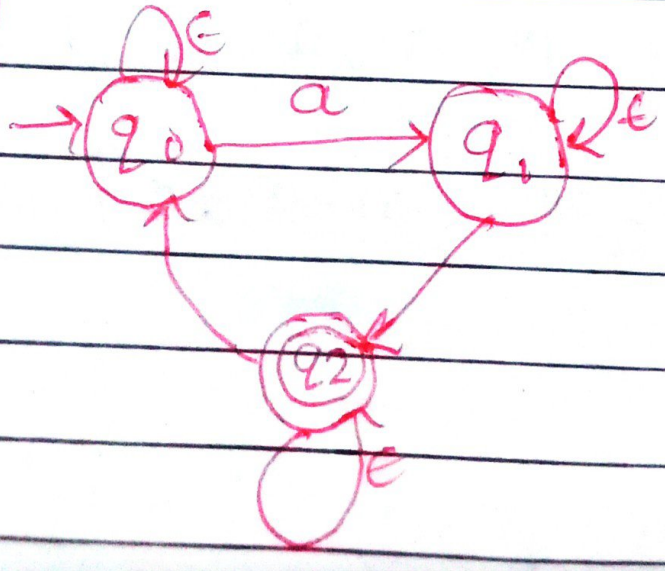
$\Sigma^* = \{\epsilon, a, aa, aaa, aaaa, ----\}$



(ii)

(iii)





All Digital computer are deterministic
[all real world machine]

DFA :- {Deterministic Finite Automata}

* 'Deterministic' specify that transition of DFA from one state to another state on a given input is unique [i.e there is only one next state]
* null string (λ) is not allowed.
* It is necessary to define DFA for
* each input in each state

DFA has 5 tuples $(Q, \Sigma, q_0, F, \delta)$
where

$Q \rightarrow$ Total no of finite, non empty set of states.
$\Sigma \rightarrow$ Finite, non empty set of input alphabet
$q_0 \rightarrow$ initial state / start state $\{q_0 \in Q\}$
$F \rightarrow$ Set of final states $\{F \subseteq Q\}$
$\delta \rightarrow$ transition function

$Q \times \Sigma \rightarrow Q$ → next states
↓ ↓
present input
State alphabet

$\delta(q_0, a) \rightarrow q_1$; $\delta(q_0, b) \rightarrow q_0$
$\delta(q_1, a) \rightarrow q_1$; $\delta(q_1, b) \rightarrow q_0$

* DFA is difficult to design
* DFA cannot be converted to NDFA
* Dead configuration is NOT allowed

NDFA → Not for real computers

DATE

NDFA { Non-deterministic Finite Automata }

* "Non-deterministic" specify that transition of NDFA from one state to another state on a given input is not unique (i.e there may be more than 1 next state possible!)
* null string (Λ) is allowed
* It is not necessary to defined NDFA for each input in each state

NDFA also have 5 tuples

$(Q, \Sigma, q_0, F, \delta)$; where

$Q \rightarrow$ Total finite, non empty set of states

$\Sigma \rightarrow$ Finite, non empty set of input alphabet

$q_0 \rightarrow$ Initial state / Start state $\{q_0 \in Q\}$

$F \rightarrow$ set of final states $\{F \subseteq Q\}$

$\delta \rightarrow$ transition function

$$Q \times (\Sigma \cup \Lambda) \rightarrow 2^Q$$

- Q: present state
- $(\Sigma \cup \Lambda)$: i/p alphabet
- 2^Q: set of max possible next states

$\delta(q_0, a) \rightarrow q_1$ $\quad$ $\delta(q_1, a) \rightarrow q_0$

$\delta(q_1, b) \rightarrow q_1$

* NDFA is easy to design
* NDFA can be converted to DFA
* Dead configuration is allowed.

* Representation of Transition fun^{n} in FA

(1) Transition Table (2) Transition diagram

Transition Table:- It is the tabular representation of the transition function.

eg: $\delta(q_0, a) \rightarrow q_0$ $\quad \delta(q_1, 0) \rightarrow q_1$

$\delta(q_0, 1) \rightarrow q_1$ $\quad \delta(q_1, 1) \rightarrow q_0$

Initial State $= q_0$

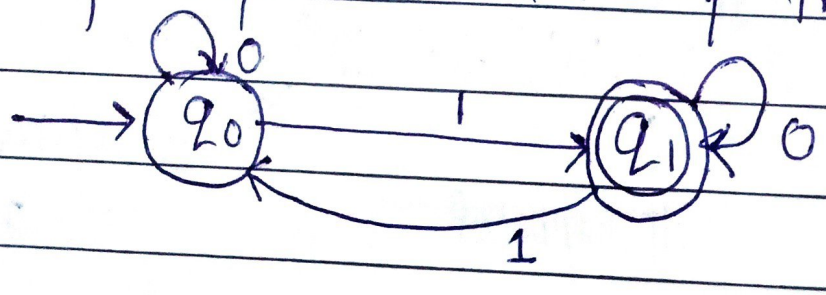
Final state $= q_1$

where q = Total no of states

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

q \ Σ	0	1
$\rightarrow q_0$	q_0	q_1
(q_1)	q_1	q_0

(2) **Transition diagram?-** It is the diagrammatically representation of transition functio



Acceptance of a String by DFA

Eg: Show whether the given string "10110" is accepted by given DFA or not.

$\delta(q_0, 10110) \Rightarrow \delta(q_1, 0110)$ $\{(q_0,1) \to q_1$
$\Rightarrow \delta(q_1, 110)$ $(q_1,1) \to q_1$
$\Rightarrow \delta(q_0, 10)$ $(q_1,0) \to q_0$
$\Rightarrow \delta(q_1, 0)$ $(q_0,1) \to q_1$
$\Rightarrow \delta(q_1, 1)$ $(q_1,0) \to q_1$
$\Rightarrow q_0$

It is non-final state

So this string is not accepted i.e This string does not belong to given DFA.

* Representation of NFA:-

(i) using transition function:-

$Q = \{q_0, q_1, q_2\}$ $\Sigma = \{a, b\}$ initial state $= q_0$

final state $= q_2$

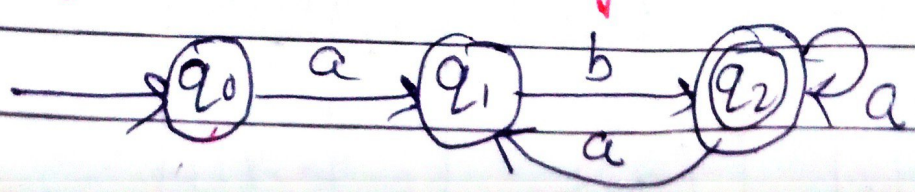
transition function

$\delta(q_0, a) \to q_1$

$\delta(q_1, b) \to q_2$

$\delta(q_2, a) \to q_1, q_2$

(ii) using transition diagram.



(iii) using transition table:

Q \ Σ	a	b
q_0	q_1	
q_1		q_2
(q_2)	q_1, q_2	

Equivalance of NDFA & DFA
(Conversion from NDFA to DFA)

Eg:- Convert the given non-deterministic finite automata (NDFA) into DFA
$M = (\{q_0, q_1, q_2\}, (a, b), \delta, q_0, \{q_2\})$
where δ is given by

Q \ Σ	a	b
→q_0	q_0, q_1	q_2
q_1	q_0	q_1
(q_2)		q_0, q_1

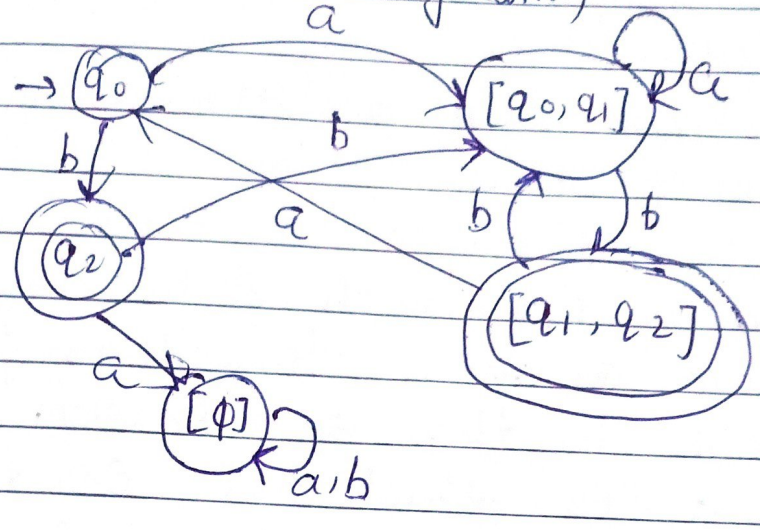
Soln

DFA transition table

Q \ Σ	a	b
→ $[q_0]$	$[q_0, q_1]$	$[q_2]$
$[q_2]$	$[\phi]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_1, q_2]$
$[\phi]$	$[\phi]$	$[\phi]$
$[q_1, q_2]$	$[q_0]$	$[q_0, q_1]$

If no of states in NFA is = n
Then in DFA it will be 2^n

DFA (transition diagram)



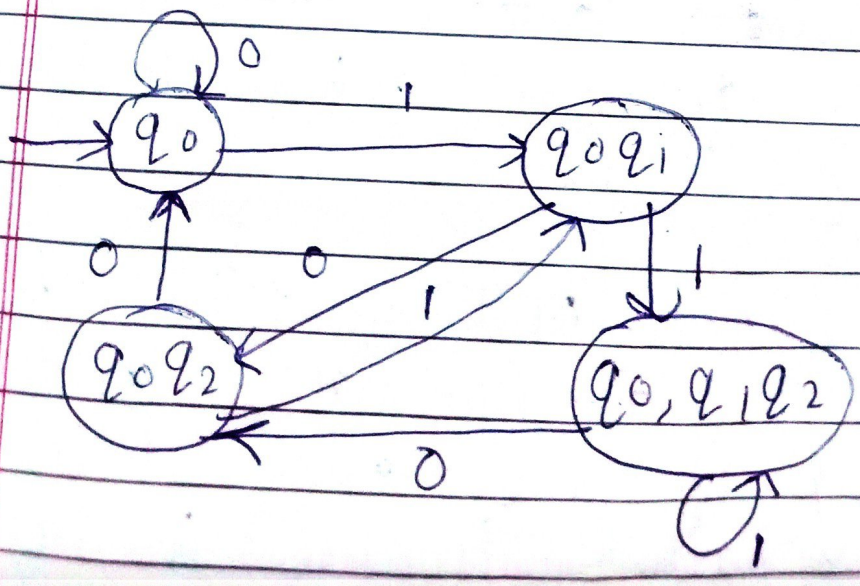
Que convert NFA into DFA

NFA

State	0	1
→q_0	q_0	q_0q_1
q_1	q_2	q_2
(*)q_2	–	–

⇒

States	0	1
→$[q_0]$	$[q_0]$	$[q_0q_1]$
$[q_0q_1]$	$[q_0q_2]$	$[q_0q_1q_2]$
$[q_0q_2]$	$[q_0]$	$[q_0q_1]$
$[q_0q_1q_2]$	$[q_0q_2]$	$[q_0q_1q_2]$



NFA

States:- Like DFA, an NFA has a finite number of states.

Transition:- From each state, there can be zero or more transitions for each symbols in the of alphabet. Unlike a DFA, which has exactly one transition for each symbol & state combination, an NFA can have multiple transitions from a state for the same symbol.

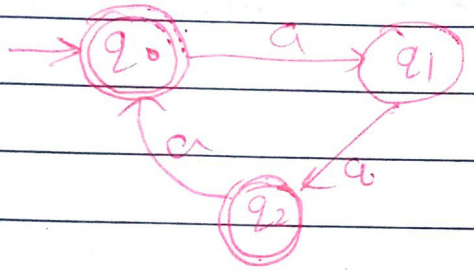
Non-determinism:- The term "non-deterministic" refers to the fact that when there are multiple transition posible from a state for a given input symbol, the NFA can choose any one of them to proceed. It doesn't have to make a unique choice

Acceptance:- NFA can have multiple accepting states. It accepts a string if there exists at least one path through the automata that leads to an accepting state

Set of all strings accepted by DFA is called Language accepted by DFA.

smallest string
$= L\{a^2, a^5, a^8, a^{11}, a^{14} \ldots\}$
$L = \{a^{3n+2} \mid n \geq 0\}$

* This DFA will accepted all the strings over input alphabet $\Sigma\{a\}$, whose lengths when divide by 3 gives remainder 2.



* 'ϵ' is accepted by iff & only iff initial state is final state

$\{\epsilon, a^2, a^3, a^5, a^6, a^8, a^9, \ldots\}$

$L = \{a^{3n+2} \cup a^{3m} \mid n, m \geq 0\}$

divide by 3 gives remainder 2 OR 0

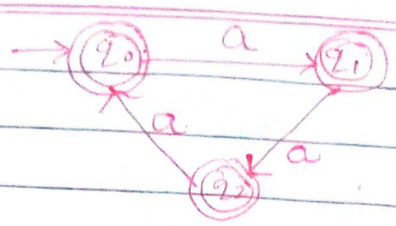
Imp Language 'L' is accepted by DFA iff and only iff strings are presented in Language & all Language which is not in 'L' are rejected.

FA is
DFA.

}

he strings
whose
gives

*

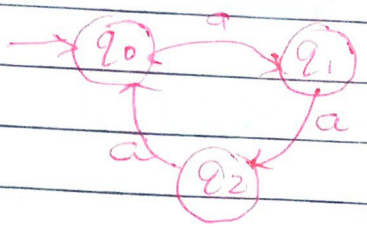


$\{\epsilon, a, a^2, a^3, a^4, a^5 \ldots\}$

$L = \{a^n \mid n \geq 0\}$

When divide by 3 gives remainder 0 1 or 2.

*

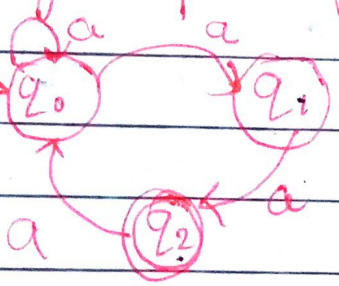


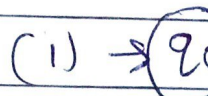
Language accepted by this DFA is ϕ
accepte- Language
empty Language.

⇒ If DFA & NFA has no final state, then Language accepted will be ϕ. {empty Language}.

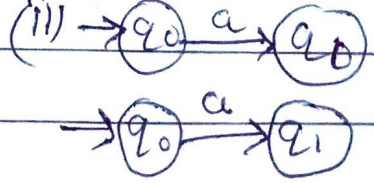
* String accepted by NFA

⇒

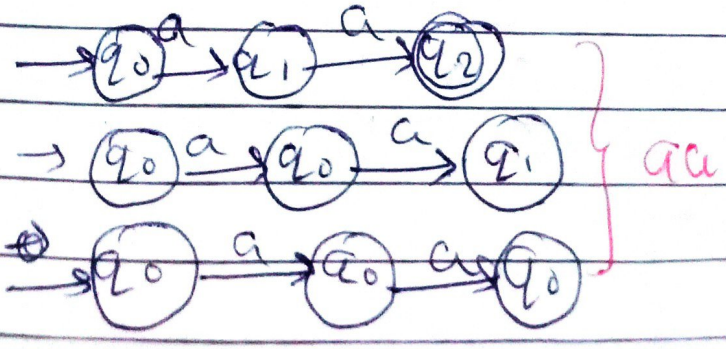


(i) 

ϵ

(ii) 

a

(iii) 

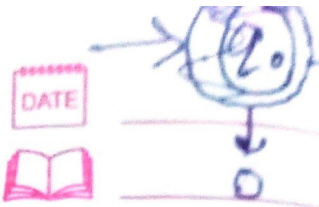
aa

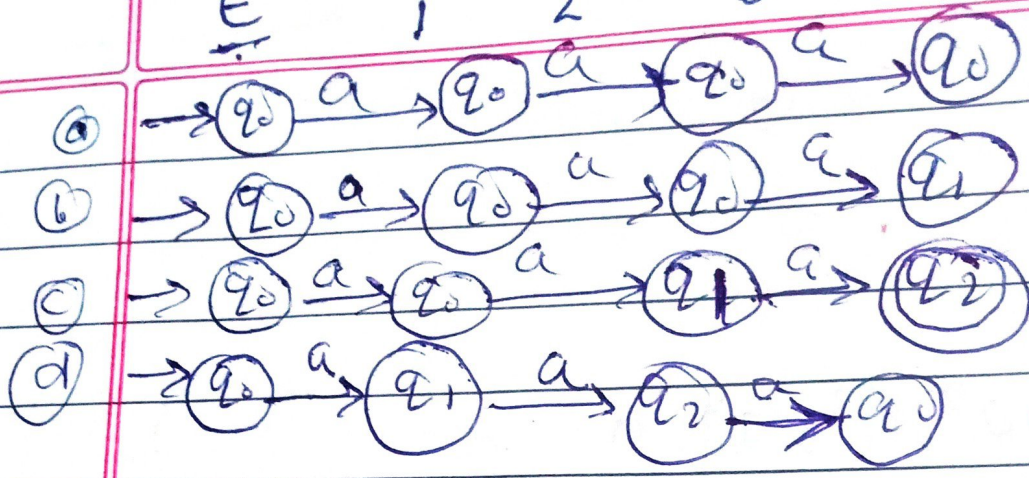
1. 0^0, 0^1, 0^2, 0^3

0, 00, 000

$\downarrow$ $\downarrow$ $\downarrow$ $\downarrow$

ϵ 1 2 3





There exist at least one path ~~for NF~~ after accessing it, NFA is at a one of ~~final state~~.

Language accepted by NFA

$L = \{a^2, a^3, a^4, a^5, a^6 \ldots\ldots\ldots\}$

$L = \{ a^n \mid n \geq 2 \}$

Length at least two

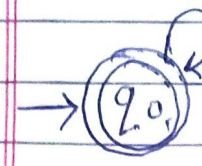
Σ^+

DATE 1/03

111 0 1101

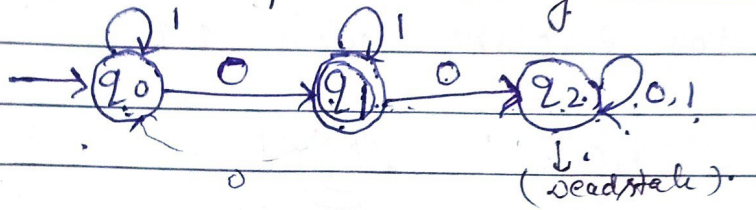
Design DFA for the following Language over $\Sigma = \{0, 1\}$

① That accept all strings of 0's & 1's.



$L =$

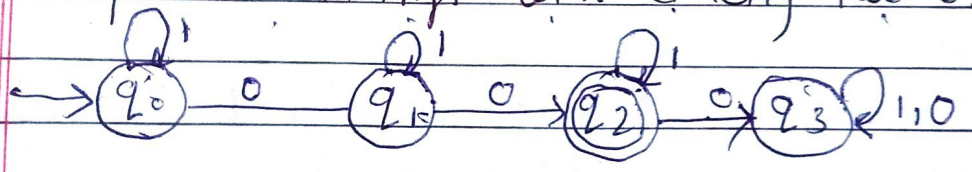
② that accept all strings with exactly one 0's.



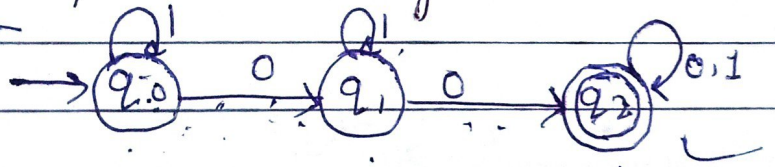
$|w| = 0$

③ accept all strings with exactly two 0's.

$|w| = 2$

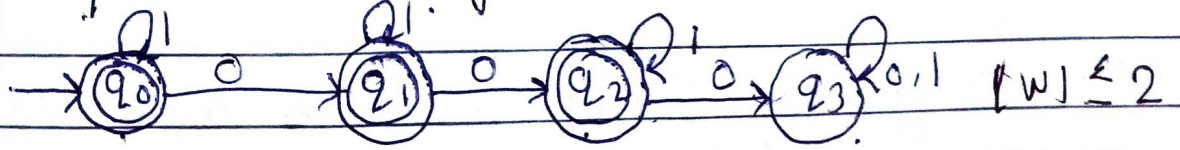


④ accept all strings with at least two 0's.



$|w| \geq 2$

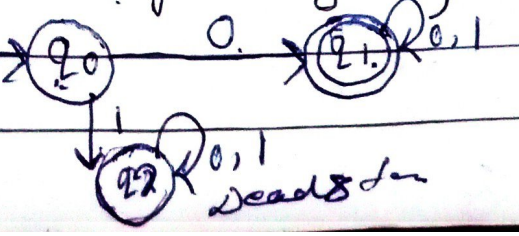
⑤ accept all strings with at most two 0's



$|w| \leq 2$

$L = \{\epsilon, 0, 00, 00\}$

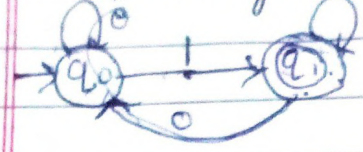
⑥ all strings begining with 0



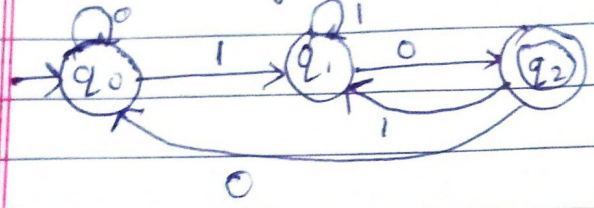
$L = \{0 \ldots\}$

for NFA at any

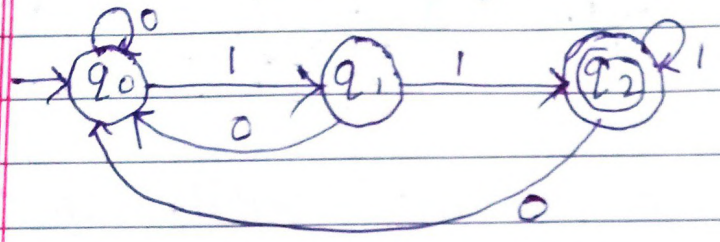
(7) all strings ending with 1



(8) all strings ending with 10

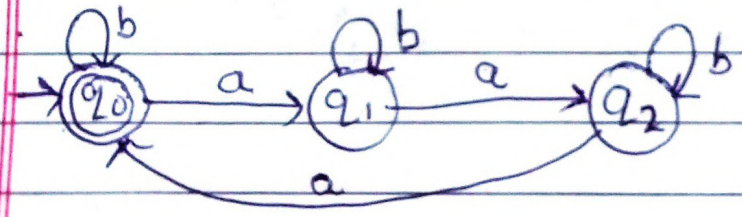


(9) all string ending with 11

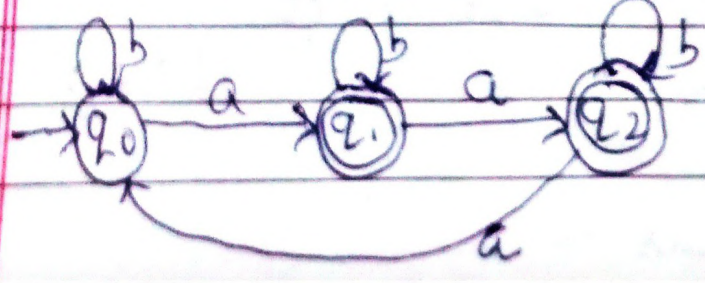


(10) accepts all strings in which no of a's are multiple of 3.

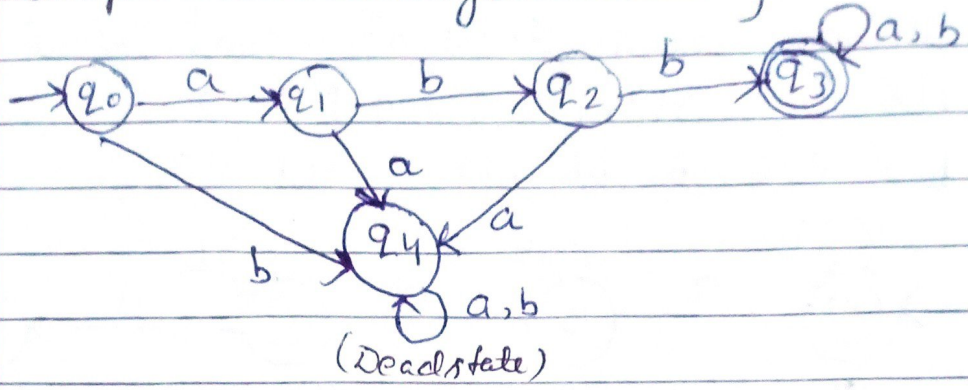
$\Sigma = \{a, b\}$



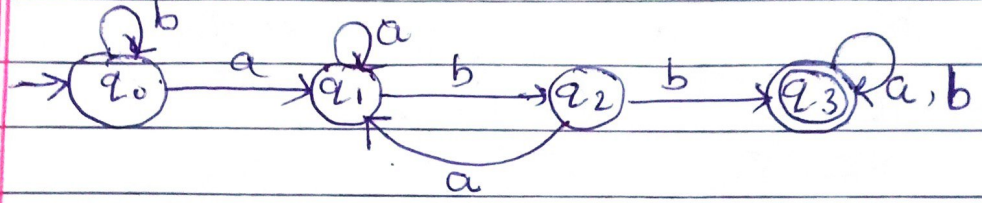
(11) accepts all strings in which no of a's are not multiple of 3.



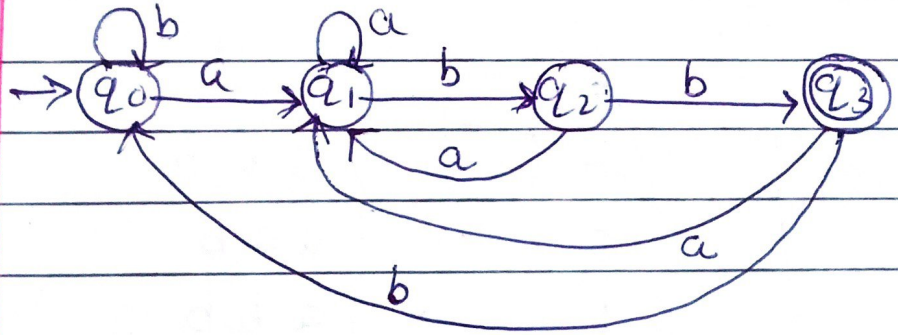
(12) accepts all strings starting with abb.



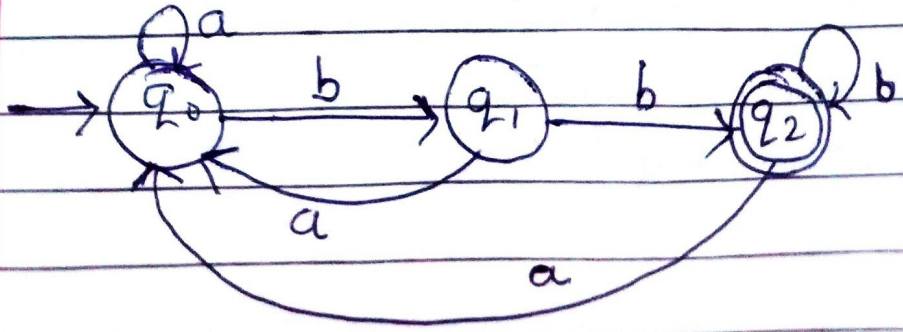
(13) accepts all strings with abb as substrings.



(14) accepts all strings ending with abb.



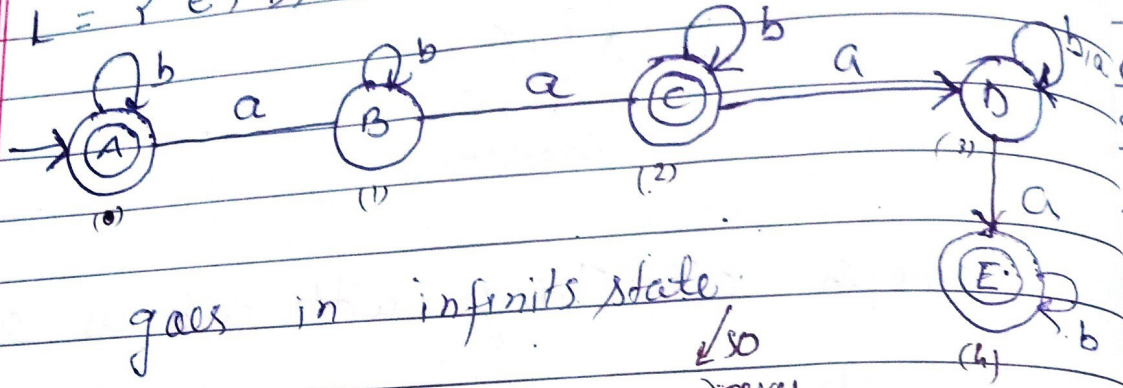
(15) accepts all strings ending with bb



(16) Construct DFA which accepts set of all strings over $\Sigma = \{a, b\}$ such that

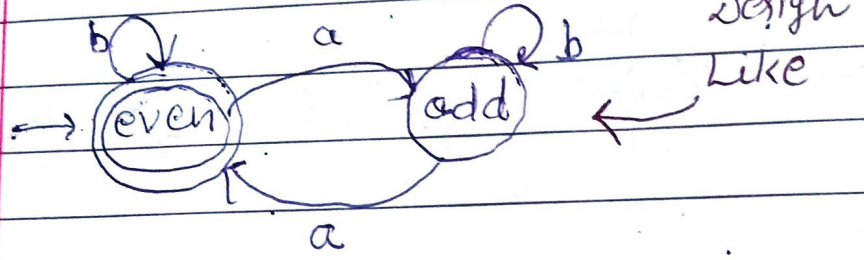
(i) $n(a)(w) \bmod 2 = 0$

$L = \{\epsilon, b, bb, bbb \ldots aa, aaaa, \ldots\}$



goes in infinite state

So design like

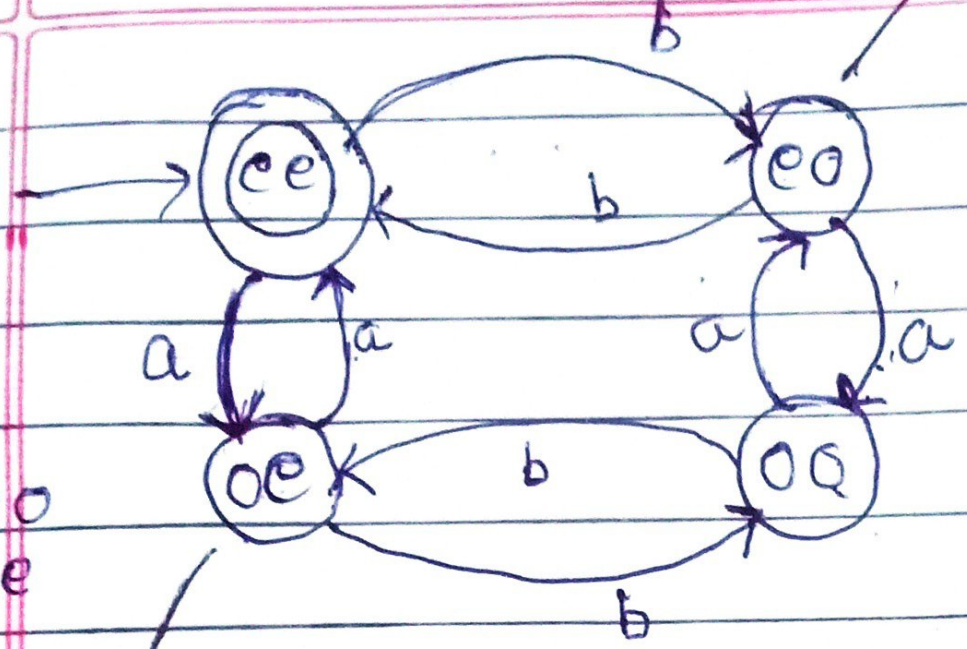


(ii) $n_{\{a\}}(w) \bmod 2 = 0$ & & $n_{\{b\}}(w) \bmod 2 = 0$

a	b
0	0
1	1

a	b	
0	0	aabb
0	1	aab
1	0	abb
1	1	ab

a	b	
0	0	even even (ee)
0	1	even odd (eo)
1	0	odd even (oe)
1	1	odd odd (oo)



$\frac{a}{0}\ \frac{b}{0}$

$\frac{a}{0}\ \frac{b}{1}$

ab - oo
aa - ee

ba - oo
bb - ee

baa - eo
bab - oe

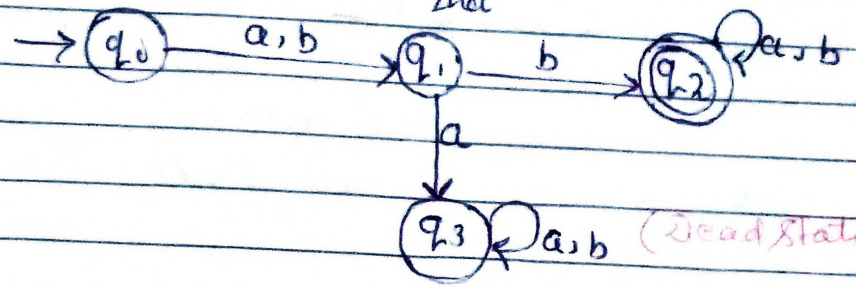
When we will see a's no odd & b's no even
like $\frac{a}{1} - \frac{b}{0}$ b's value 0 which is even

Page No.
Date

Q Design a minimal DFA that accepts all strings over the alphabet $\Sigma = \{a, b\}$ such that for every accepted string 2nd from left end is always b.

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

Language = _ b _ _ _ _
2nd

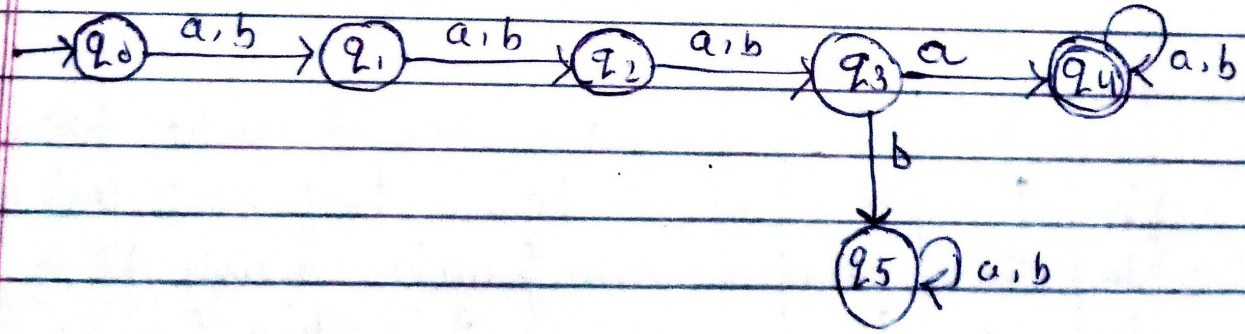


② Such that for every accepted string 4th from left end is always a

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

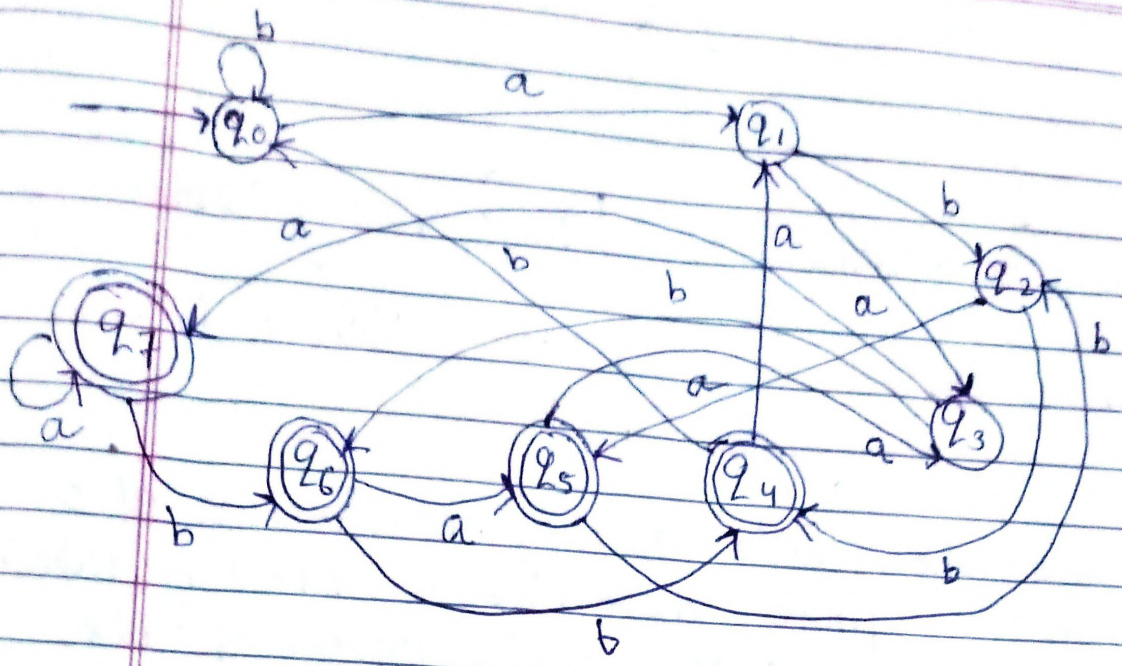
L = _ _ _ a _ _ _
4th

[these symbol can be anything either a or b]
but 4th will be a]



- When no is 2nd from Left, no of states 4
  2+2 = 4
- When no is 4th from left, no of states 6
  4+2 = 6

If position is 'n' from Left side, then No of states will be n+2



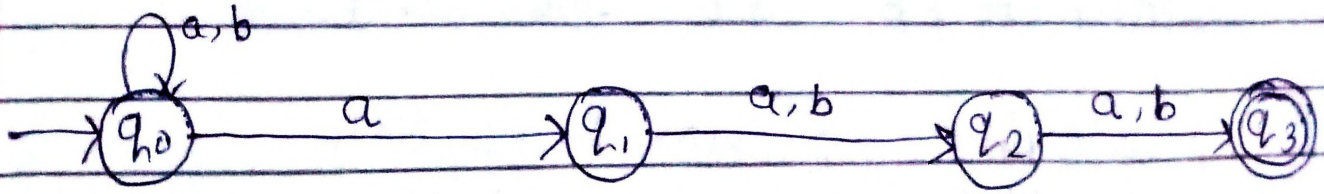
Que: Construct a Non Deterministic Finite Automation (NFA) for the Language L which accepts all the strings in which the third symbol from right end is always 'a' over $\Sigma = \{a, b\}$

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

L = a/b a/b a a/b a/b

3rd 2nd 1st

↓ from right side will be a



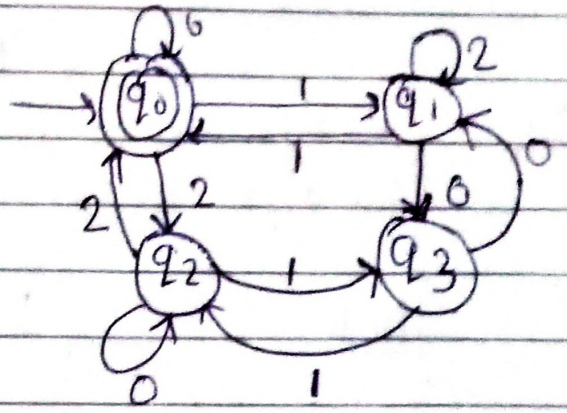
This is NFA for the given.

Important question on Construction of DFA & NFA

Que Construct a DFA for ternary number divisible by 4

ternary $\{0, 1, 2\}$ will be input alphabet
& divisible by 4 means
remainders $[0, 1, 2, 3]$ will be states

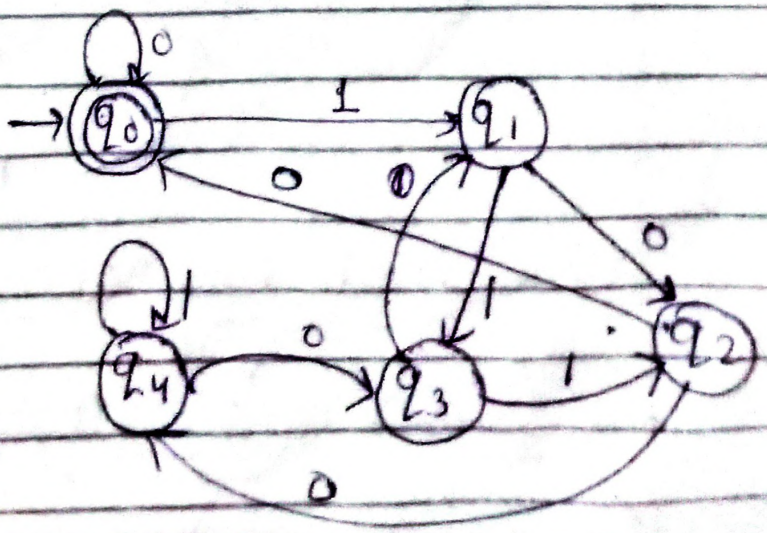
	0	1	2
→(q0)	q0	q1	q2
q1	q3	q0	q1
q2	q2	q3	q0
q3	q1	q2	q3



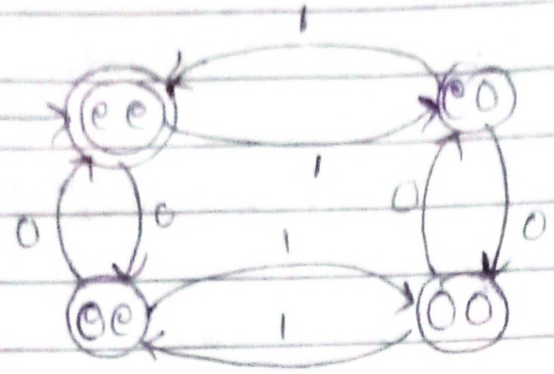
Que Define DFA & design a DFA that accepts the binary number whose equivalent is divisible by 5

Binary = $\{0, 1\}$ remainder = $\{0, 1, 2, 3, 4\}$

	0	1
→(q0)	q0	q1
q1	q2	q3
q2	q4	q0
q3	q1	q2
q4	q3	q4



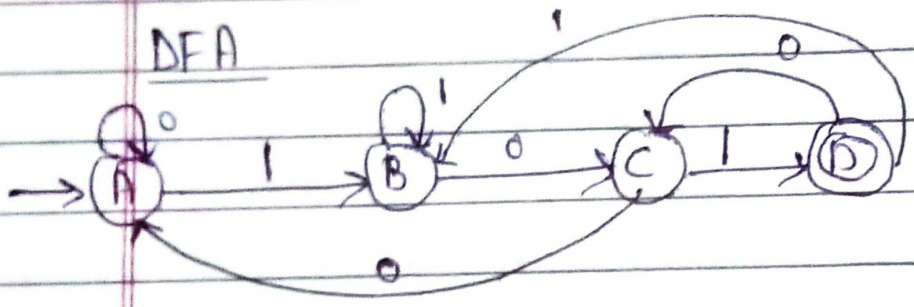
Q.11 Design the DFA that accepts an even number of 0's & even no. of 1's



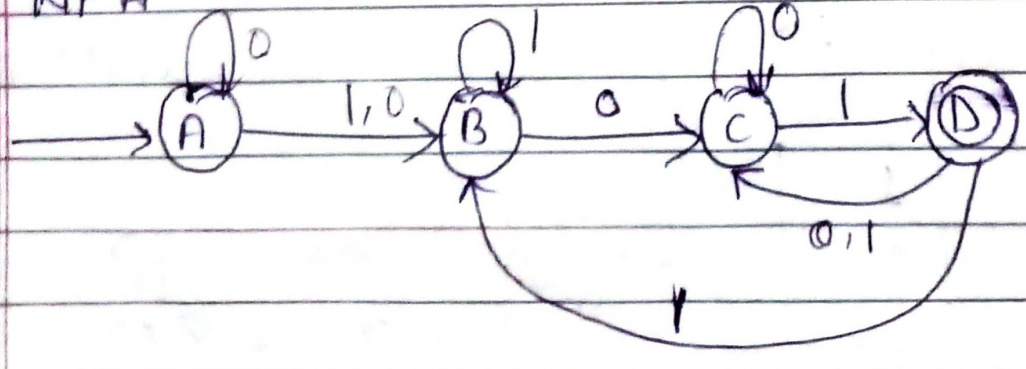
∴ NFA

Q.12 Design a DFA to accept the string that always ends with 101.

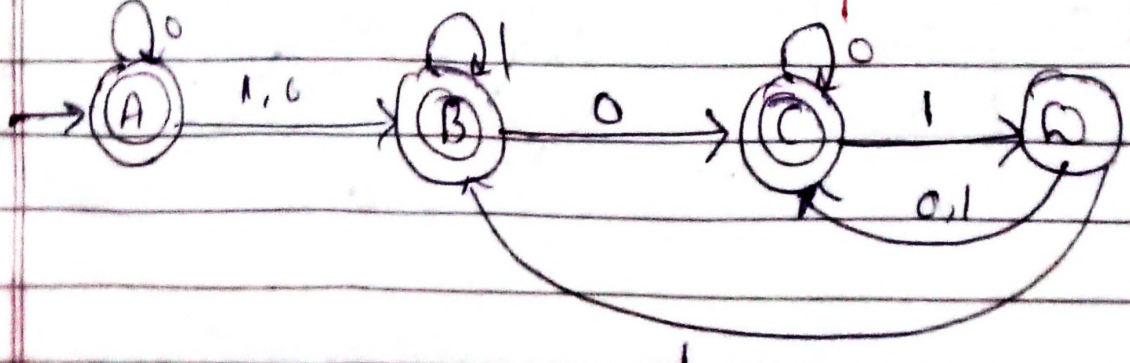
DFA



NFA



NFA that does not accept string ending 101

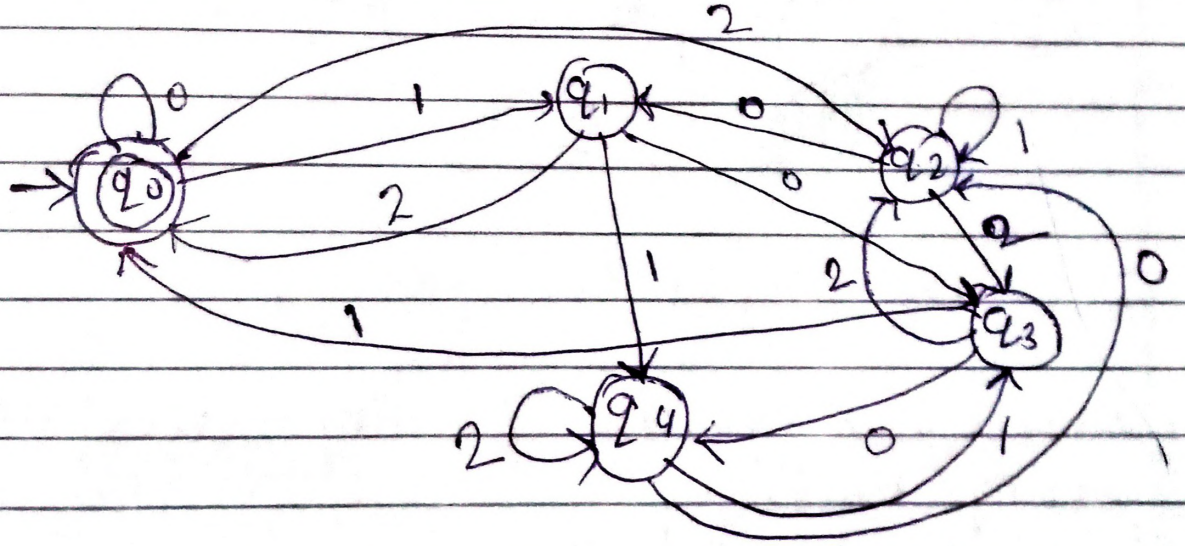


Q Design a FA for ternary number divisible by 5.

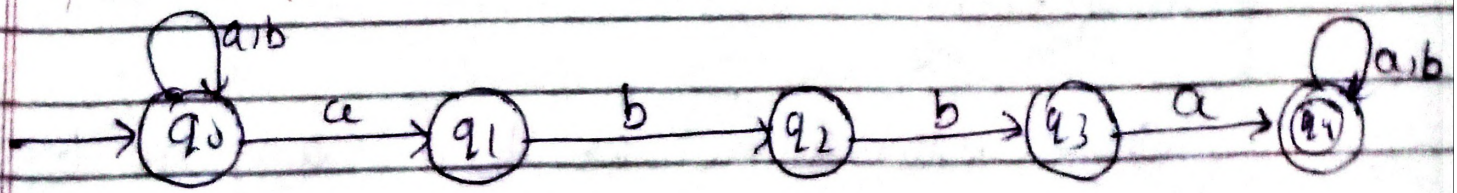
Input alphbet ternary = {0, 1, 2}

States {0, 1, 2, 3, 4}

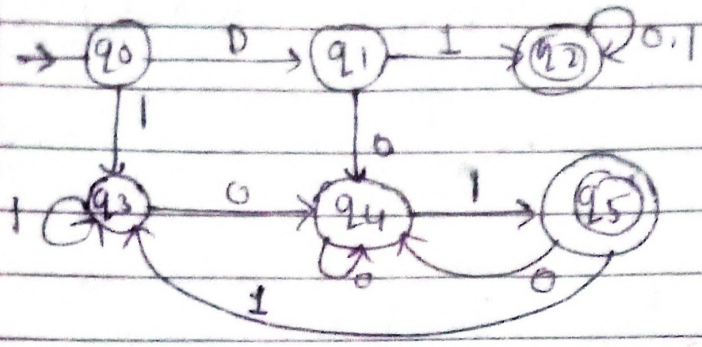
	0	1	2
→ (q0)	q0	q1	q2
q1	q3	q4	q0
q2	q1	q2	q3
q3	q4	q0	q1
q4	q2	q3	q4



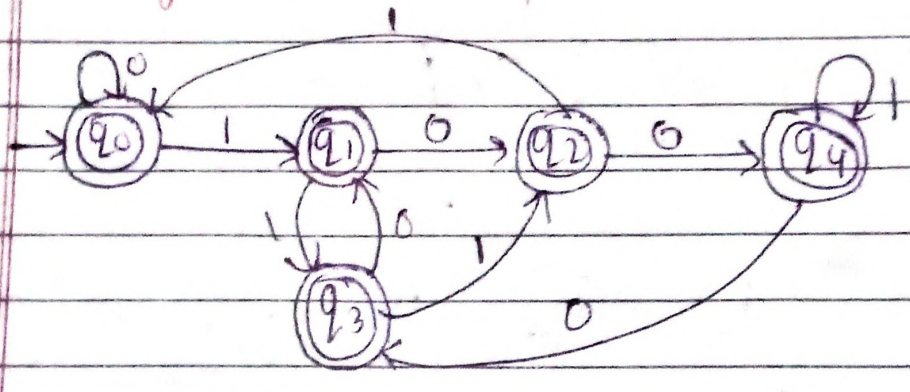
Q Design NFA that accepts all the strings for input alphabet containing substring abba



Ques Design a DFA that accept a substring starting with 01 or 10 and ending with 01 for $\Sigma = \{0, 1\}$



Ques Draw the DFA for $L = \{|w| \bmod 5 \mid w \in (0,1)^*\}$



Ques Draw a DFA for the following accepting strings on $\{0, 1\}$

(i) mod 3 = 0 | w

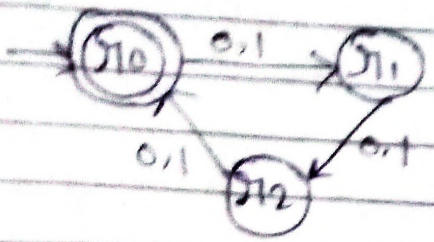
(ii) mod 3 > 0 | w

(iii) mod 3 > 1 | w

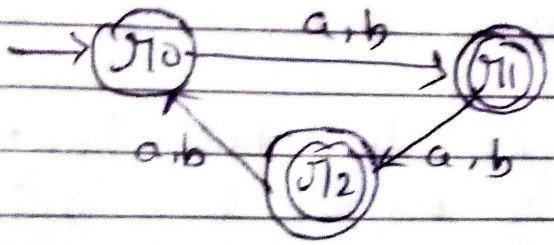
w, where w represents the length of the string.

Remember when we divide any no by 3 {0, 1, 2}

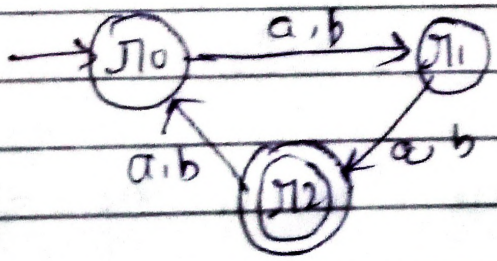
(i) accept only when remainder 0 mod 3 = 0



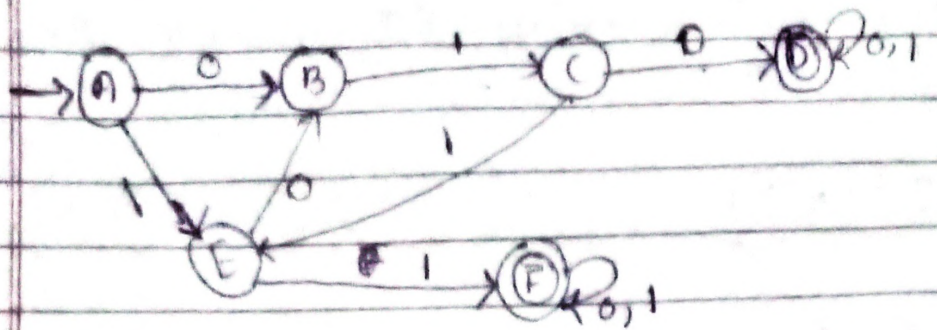
(ii) accept only when remainder greater 0 mod 3 > 0 means q1 & q2 will be final state



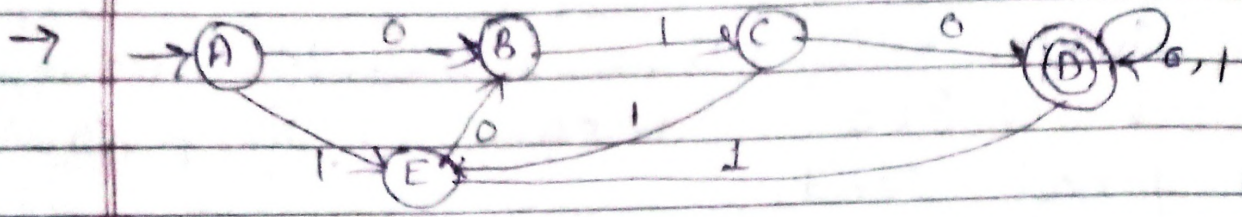
(iii) accept only when remainder greater than.



Qu Draw the finite automata which accepts all the strings containing both 11 & 010 as substrings.



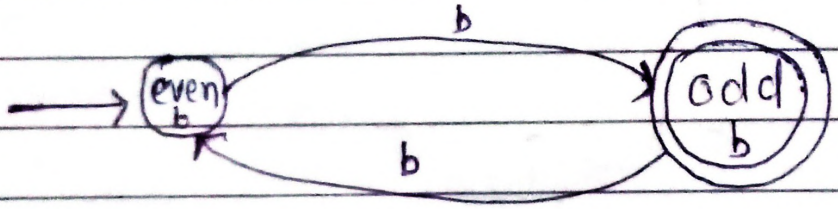
Final DFA



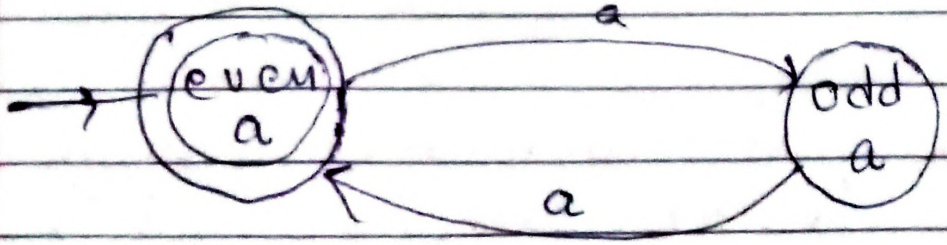
Eg: Construct an automata with even number of a's & odd number of b's.

Let L_1 & L_2 are the two languages of odd number of b's & even number's of a's respectively

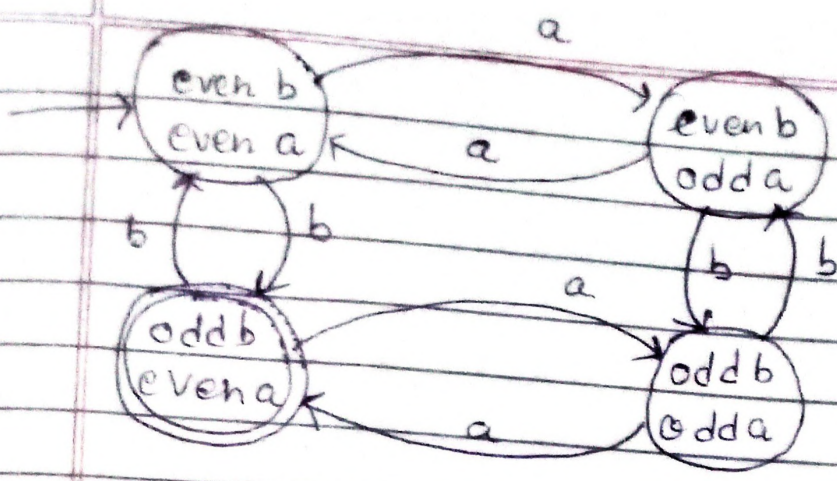
* DFA (L_1) when odd no of b's



* DFA (L_2) when even no of a's

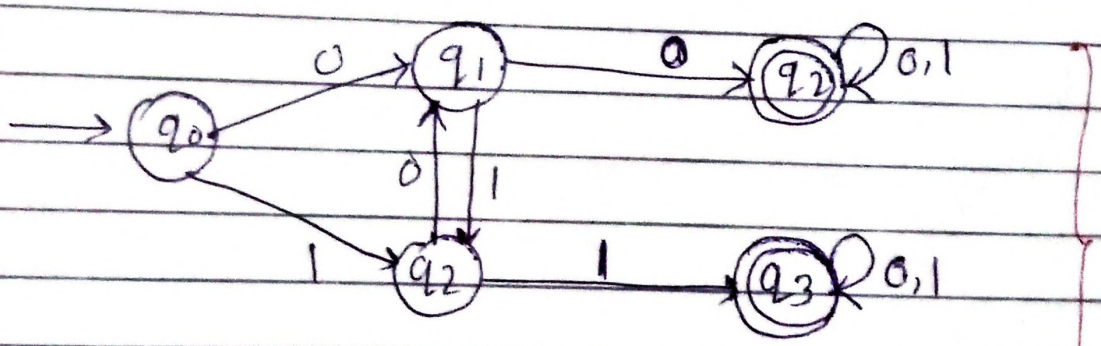


We need to determine $L = (L_1 \cup L_2)^*$ the states of DFA (L_1) & DFA (L_2) will be the cross product.

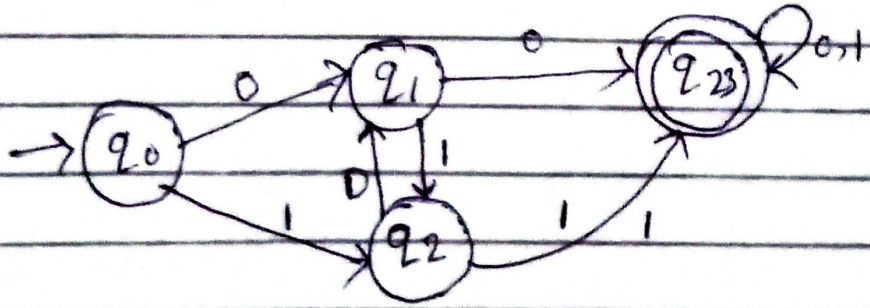


Qw:- Design FA for the following languages consisting binary strings
(i) Having both or either 11 as substring
(ii) Number's of 0's is odd and number of 1's is multiple of 3.

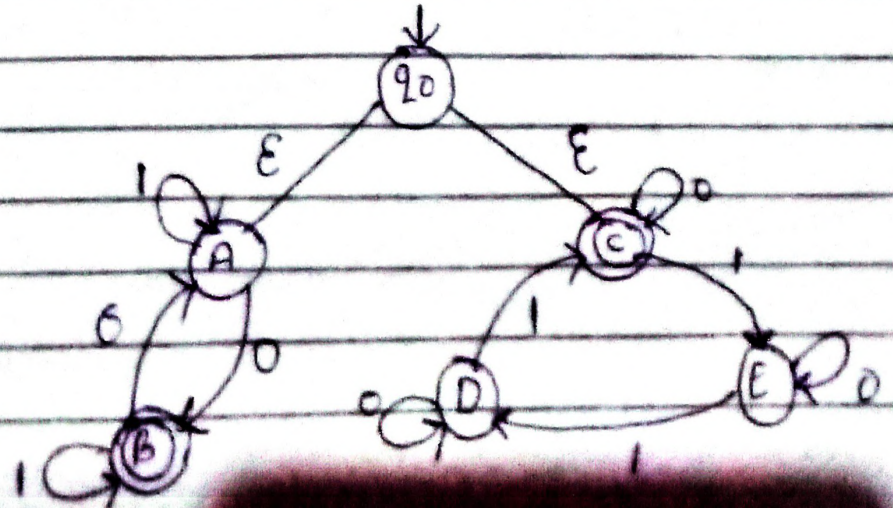
(i)



after merging q_2 & q_3



(ii)



Minimization of DFA

- Equivalance Method OR Partition Method
- Table Filling Method OR Myhill-Nerode Method

* DFA minimization helps simplify the structure of a deterministic finite automaton (DFA) by reducing the number of states while preserving its functionality.
* This simplification makes the DFA easier to understand, analyze & implement in various applications.

⇒ DFA minimization using Equivalence or Partition Method

Two states 'A' & 'B' are said to be equivalent if :-

$\delta(A, a) \rightarrow$ Final state
$\delta(B, a) \rightarrow$ Final state

then A & B are equivalence.

$\delta(A, b) \rightarrow$ non Final state
$\delta(B, b) \rightarrow$ non Final state

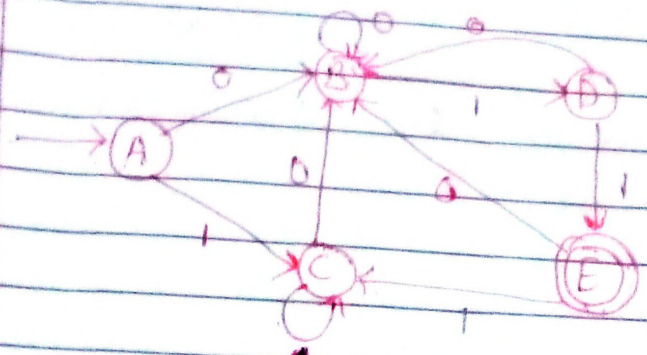
then

(A & B) are equivalence.

$X = \{a, b\}$ input symbol

if $|x| = 0$, then A ≠ B are said to be 0 equivalent
if $|x| = 1$, then A ≠ B are said to be 1 equivalent
if $|x| = 2$, then A ≠ B are said to be 2 equivalent
⋮
if $|x| = n$, then A ≠ B are said to be 'n' equivalent

Eg:-



Step 1:- Design transition table

	0	1
→A	B	C
B	B	D
C	B	C
D	B	E
(E)	B	C

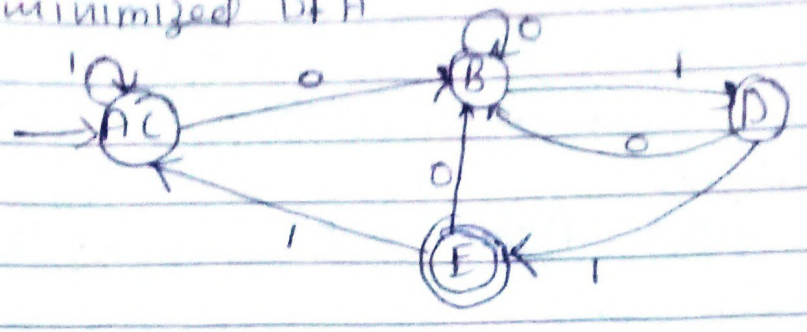
Step 2: E quivalence G1 G2

0 Equivalence {A, B, C, D} {E}

at 0 Equivalence make two group all non final state in G1 Group & Final in G2 Group

0 Equivalence {A, B, C, D} {E}
1 Equivalence {A, B, C} {D} {E}
2 Equivalence {A, C} {B} {D} {E}
3 Equivalence {A, C} {B} {D} {E}

minimized DFA



Eg:- Construct a minimum DFA equivalent to the DFA described by

	0	1
→q_0	q_1	q_5
q_1	q_6	q_2
(q_2)	q_0	q_2
q_3	q_7	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

Soln

0 Equivalence

$\{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}$ $\{q_2\}$

1- Equivalence

$\{q_0, q_4, q_6\}$

$\{q_1, q_7\}$

$\{q_3, q_5\}$ $\{q_2\}$

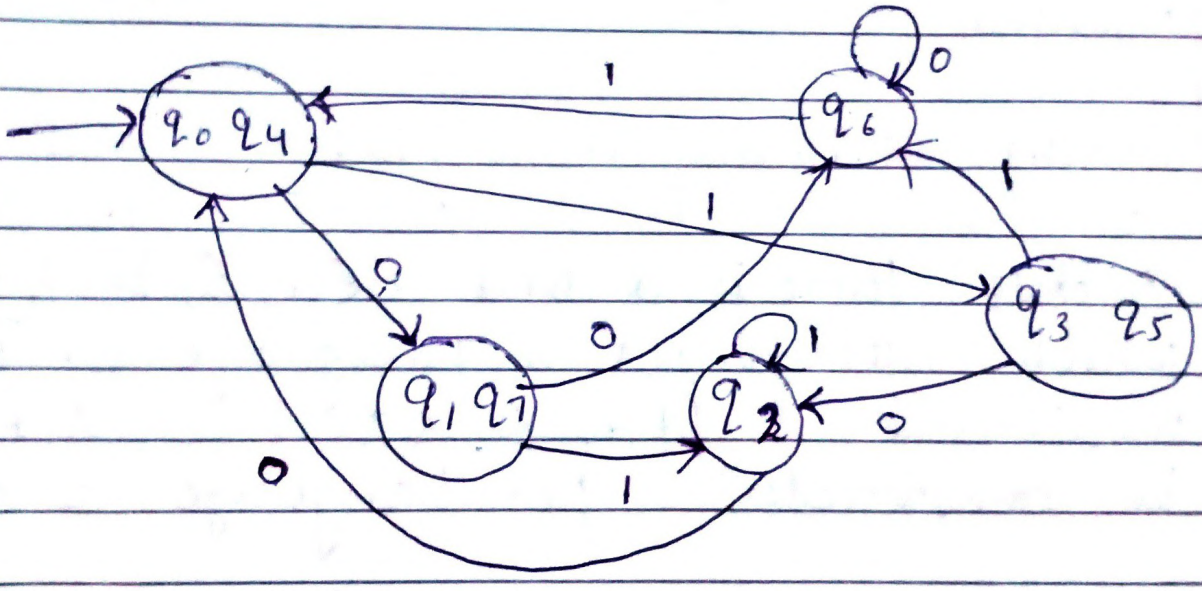
2- Equivalence

$\{q_0, q_4\}$ $\{q_6\}$ $\{q_1, q_7\}$ $\{q_3, q_5\}$ $\{q_2\}$

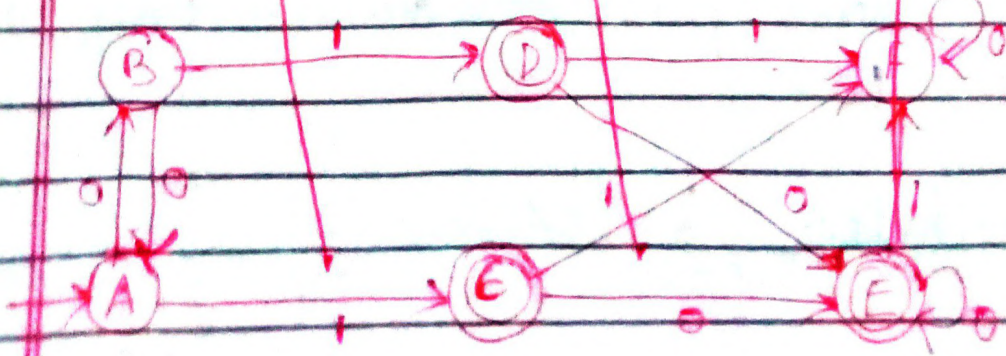
3- Equivalence

$\{q_0, q_4\}$ $\{q_6\}$ $\{q_1, q_7\}$ $\{q_3, q_5\}$ $\{q_2\}$

	0	1
→ $\{q_0\ q_4\}$	$\{q_1, q_7\}$	$\{q_3, q_5\}$
$\{q_6\}$	$\{q_6\}$	$\{q_0, q_4\}$
$\{q_1, q_7\}$	$\{q_6\}$	$\{q_2\}$
$\{q_3, q_5\}$	$\{q_2\}$	$\{q_6\}$
$\{q_2\}$	$\{q_0, q_4\}$	$\{q_2\}$



Eg:- When there are more than one final st



0 - Equivalence - {A, B, F} {C, D, E}
1 - Equivalence - {A, B,} {F}

Minimization of DFA:-

* DFA minimization stands for converting a given DFA to its equivalent DFA with minimum number of states.
* DFA minimization is also called as optimization of DFA & uses partitioning algorithm

* Algorithm of Minimization of DFA:-

Suppose there is a DFA $D\langle Q, \Sigma, q_0, \delta, F\rangle$ which recognizes a Language L. Then the minimized DFA $D \langle Q, \Sigma, q_0, \Delta, F\rangle$ can be constructed for Language L as:- Eg:-

Step 1:- We will divide Q {set of states} into two sets. One set will contain all final states and other set will contain non final states. This partition is called P_0.

Step 2:- Initialize $k = 1$

Step 3:- Find P_k by partitioning the different sets of P_{k-1}. In each set of P_{k-1}

will take all possible pair of states. If two states of a set are distinguishable, we will split the sets into different sets in P_k.

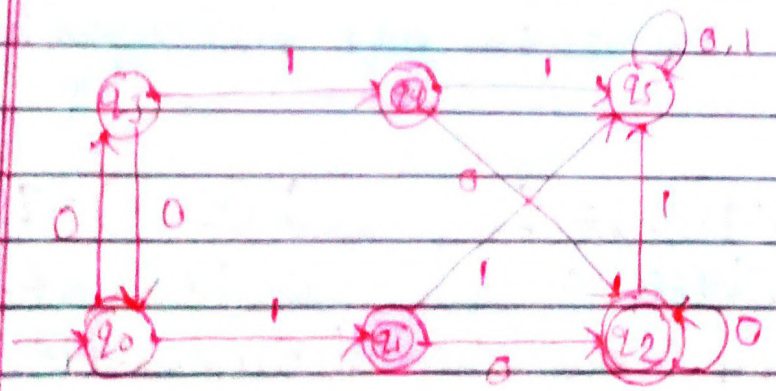
Step 4: Stop when $P_k = P_{k-1}$ (No change in partitions)

Step 5: All states of one set are merge of into one. No of states in minimized DFA will be equal to no of sets in P_k

How do find whether two states in partition P_k are distinguishable?

Two states (q_i, q_j) are distinguishable in partition P_k if for any input symbol a, $\delta(q_i, a)$ & $\delta(q_j, a)$ are in different sets in partition P_{k-1}.

eg:-



	0	1
→ q_0	q_3	q_1
(q1)	q_2	q_5
(q2)	q_2	q_5
q_3	q_0	q_4
(q4)	q_2	q_5
q_5	q_5	q_5

Step1 0-Equivalence will have two sets of states one set contain q_1, q_2, q_4 which are final states of DFA & another set will contain remaining states.
So

0-Equivalence
$\{q_1, q_2, q_4\}$ $\{q_0, q_3, q_5\}$

Step2 1-Equivalence
To calculate 1-Equivalence, we will check whether sets of partition 0-Equi. can be partitioned or not.

i) for set $\{q_1, q_2, q_4\}$
$\delta(q_1, 0) = q_2$ } $\delta(q_1, 1) = q_5$ }
$\delta(q_2, 0) = q_2$ } $\delta(q_2, 1) = q_5$ }
So q_1 & q_2 are not distinguishable

Similarly
$\delta(q_1, 0) = q_2$ } $\delta(q_1, 1) = q_5$ }
$\delta(q_4, 0) = q_2$ } $\delta(q_4, 1) = q_5$ }
So q_1 & q_4 are not distinguishable

Since, q_1 & q_2 are not distinguishable & q_1 & q_4 are also not distinguishable, so q_2 & q_4 are not distinguishable
So
$\{q_1, q_2, q_4\}$ set will be not be split in 1-Equivalence.

(ii) For set $\{q_0, q_3, q_5\}$

$\delta(q_0, 0) = q_3$ $\quad$ $\delta(q_0, 1) = q_1$

$\delta(q_3, 0) = q_0$ $\quad$ $\delta(q_3, 1) = q_4$

Moves of q_0 & q_3 on input symbol 0 are q_3 & q_0 respectively which are in same set in partition 0- Equivalence.

similarly Moves of q_0 & q_3 on input symbol 1 are q_1 & q_4 which are in same set in partition 0- Equivalence.

So $\{q_0, q_3\}$ are not distinguishable

$\left.\begin{array}{l}\delta(q_0,0) = q_3 \\ \delta(q_5,0) = q_5\end{array}\right\}$ $\quad$ $\left.\begin{array}{l}\delta(q_0,1) = q_1 \\ \delta(q_5,1) = q_5\end{array}\right\}$

Moves of q_0 & q_5 on input symbol 1 are q_1 & q_5 respectively which are in different set in partition 0-Equivalence. so q_0 & q_5 are distinguishble. so set $\{q_0, q_3, q_5\}$ will be partitioned into $\{q_0, q_3\}$ & $\{q_5\}$ so

1 - Equivalence

$\{q_0, q_3\}$ $\{q_5\}$, $\{q_1, q_2, q_4\}$

2- Equivalence

(iii) For set $\{q_0, q_3\}$

$\left.\begin{array}{l}\delta(q_0,0) = q_3 \\ \delta(q_3,0) = q_3\end{array}\right\}$ $\quad$ $\left.\begin{array}{l}\delta(q_0,1) = q_1 \\ \delta(q_3,1) = q_4\end{array}\right\}$

not distinguishable

(iv) for set $\{q_1, q_2, q_4\}$

$\delta(q_1, 0) = q_2$ $\quad \delta(q_1, 1) = q_5$

$\delta(q_2, 0) = q_2$ $\quad \delta(q_2, 1) = q_5$

so not distinguishable

$\delta(q_1, 0) = q_2$ $\quad \delta(q_1, 1) = q_5$

$\delta(q_4, 0) = q_2$ $\quad \delta(q_4, 1) = q_5$

so not distinguishable

(v) for set $\{q_5\}$

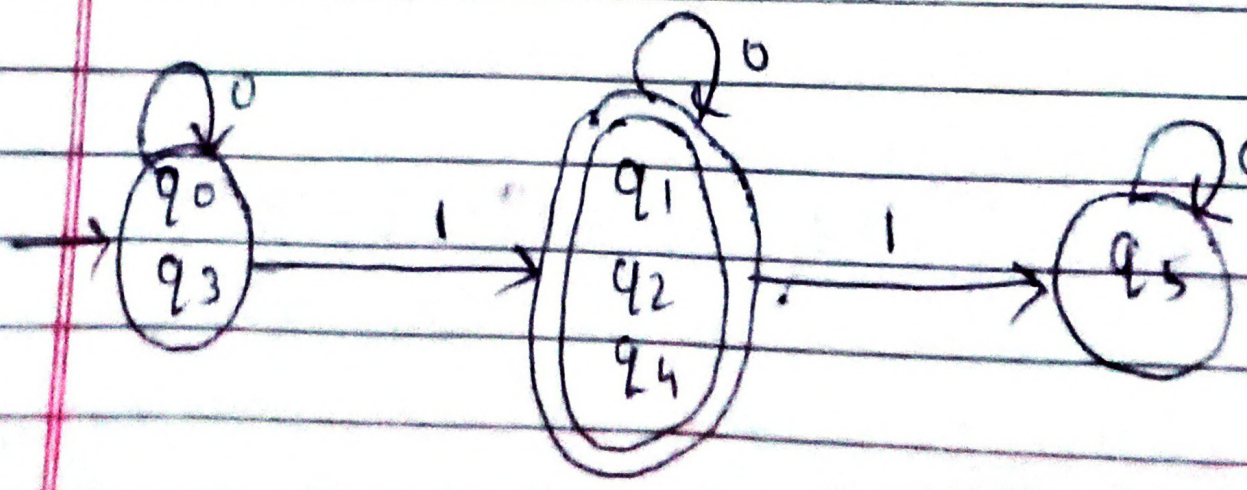
since we have only one state in the set, it can't be further partitioned.

so

2- Equivalence

$\{q_1, q_2, q_4\}, \{q_0, q_3\} \{q_5\}$

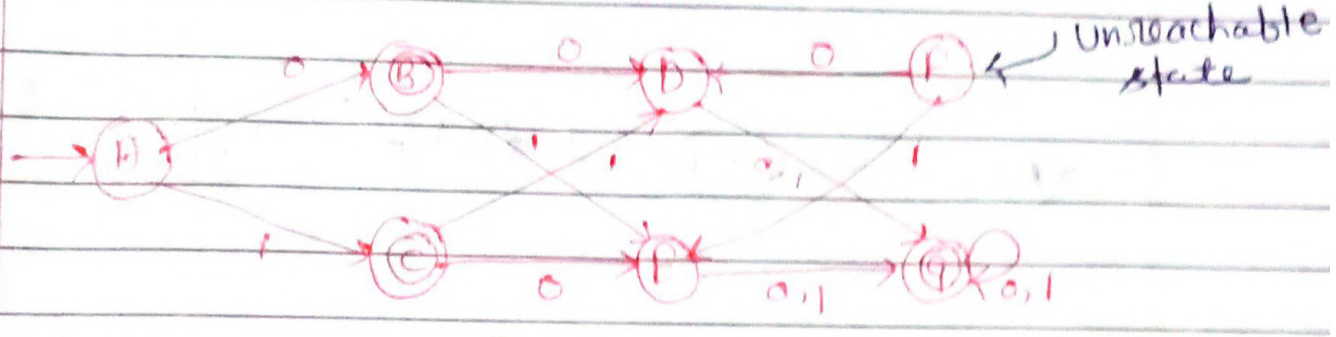
	0	1
→ $\{q_0\ q_3\}$	$\{q_0, q_3\}$	$\{q_1\ q_2\ q_4\}$
$\{q_5\}$	$\{q_5\}$	$\{q_5\}$
$\{q_1\ q_2\ q_4\}$	$\{q_1\ q_2\ q_4\}$	$\{q_5\}$



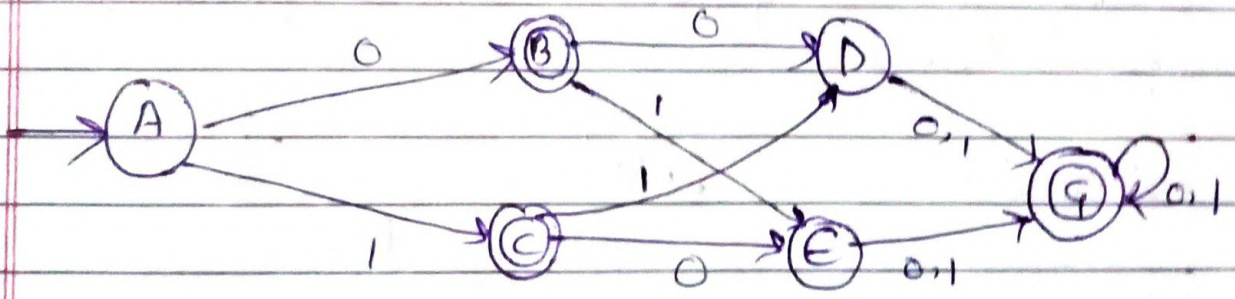
* Minimize the DFA when there are Unreachable states involved.

Unreachable States

A state is said to be Unreachable if there is no way it can be reached from the initial state.



So remove F state from the state transition diagram.



	0	1
→A	B	C
(B)	D	E
(C)	E	D
D	G	G
E	G	G
(G)	G	G

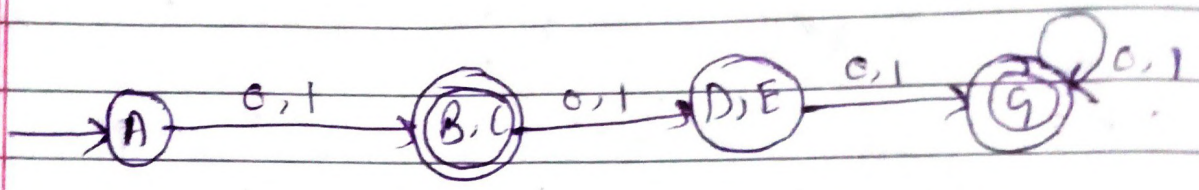
0- Equivalence
{A, D, E}, {B, C, G}
G1 G2

1- Equivalence
{A, D, E} {B, C} {G}

2- Equivalence
{A} {D, E} {B, C} {G}

3- Equivalence
{A}, {D, E} {B, C} {G}

	0	1
→ {A}	{B, C}	{B, C}
{D, E}	{G}	{G}
(B, C)	{D, E}	{D, E}
(G)	{G}	{G}



Qu Construct a minimum state automata equivalent to a DFA whose transition table is as follows where Q3 & Q4 are final state.

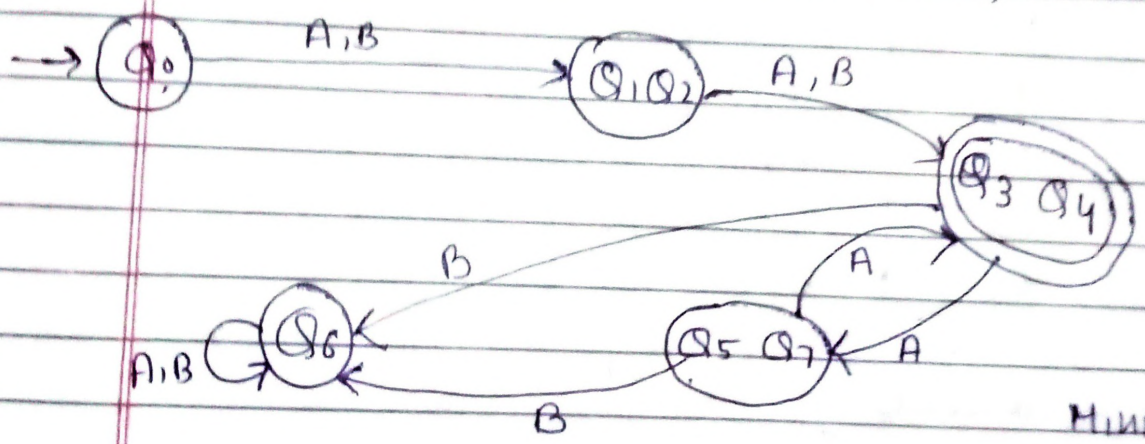
State/Σ	Input	
	A	B
→ Q0	Q1	Q2
Q1	Q4	Q3
Q2	Q4	Q3
(Q3)	Q5	Q6
(Q4)	Q7	Q6
Q5	Q3	Q6
Q6	Q6	Q6
Q7	Q4	Q6

0- Equivalence
$\{Q_0, Q_1, Q_2, Q_5, Q_6, Q_7\}$ $\{Q_3, Q_4\}$

1- Equivalence
$\{Q_0, Q_6\}$ $\{Q_1, Q_2\}$ $\{Q_5, Q_7\}$ $\{Q_3, Q_4\}$

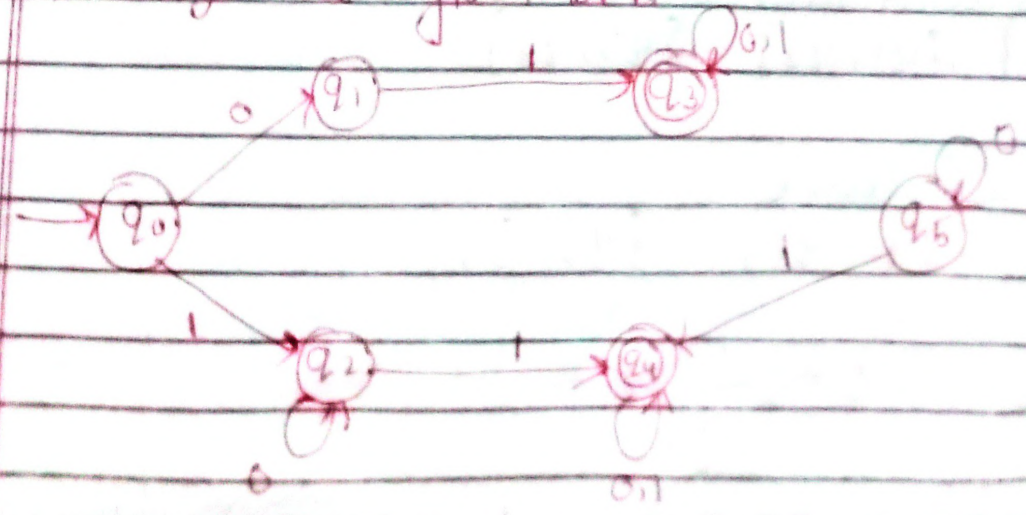
2- Equivalence
$\{Q_0\}$ $\{Q_6\}$ $\{Q_1, Q_2\}$ $\{Q_5, Q_7\}$ $\{Q_3, Q_4\}$

3- Equivalence
$\{Q_0\}$ $\{Q_6\}$ $\{Q_1, Q_2\}$ $\{Q_5, Q_7\}$ $\{Q_3\ Q_4\}$
2- Equ & 3 Equiv both are same, so we stop.



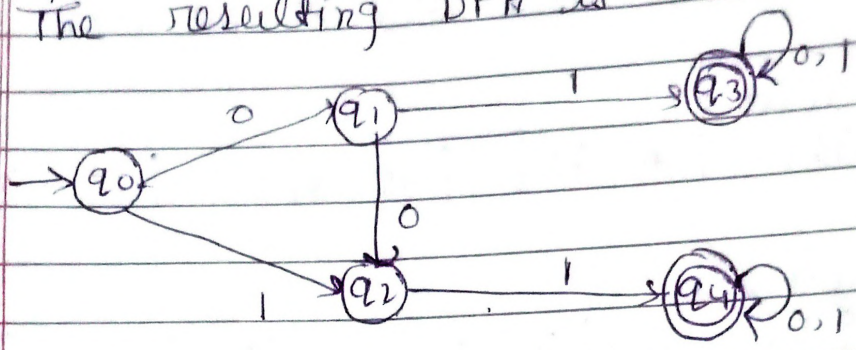
Minimized DFA

Q:- Minimize the given DFA



- State q_5 is unreachable state from the initial state
- So, we eliminate it & its associated edges from the DFA

The resulting DFA is -



	0	1
→q0	q1	q2
q1	q2	q3
q2	q2	q4
(q3)	q3	q3
(q4)	q4	q5

0- Equivalence

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

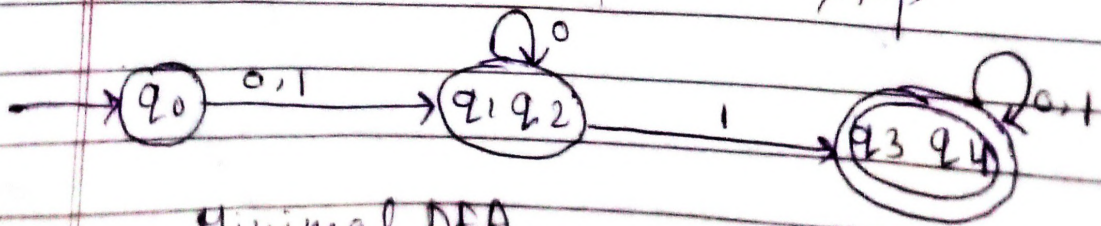
1- Equivalence

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

2- Equivalence

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

1- Equi = 2 Equi , so, we stop



Minimal DFA

8. Minimize the given DFA

State	0	1
→ q0	q1	q5
q1	q6	q2
(q2)	q0	q2
q3	q2	q6
q4	q7	q5
q5	q2	q6
q6	q6	q4
q7	q6	q2

nip

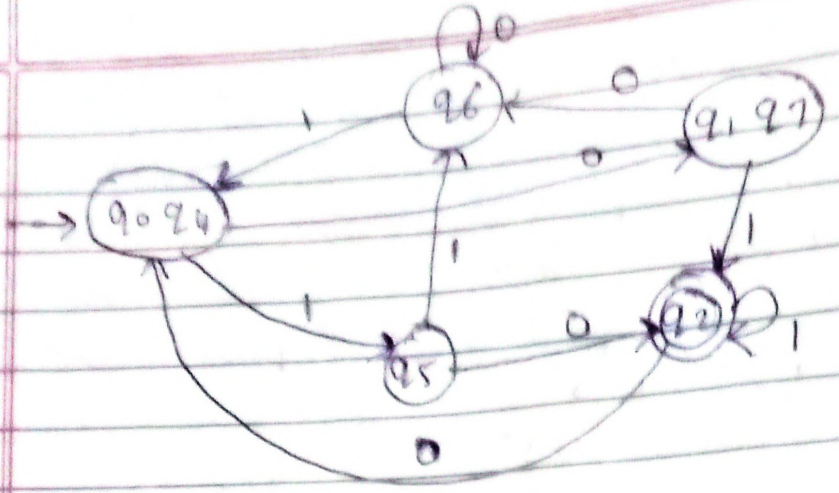
remove from table bcoz q3 is unreachable state

0- Equivalence
{ q0, q1, q4, q5, q6, q7 } {q2}

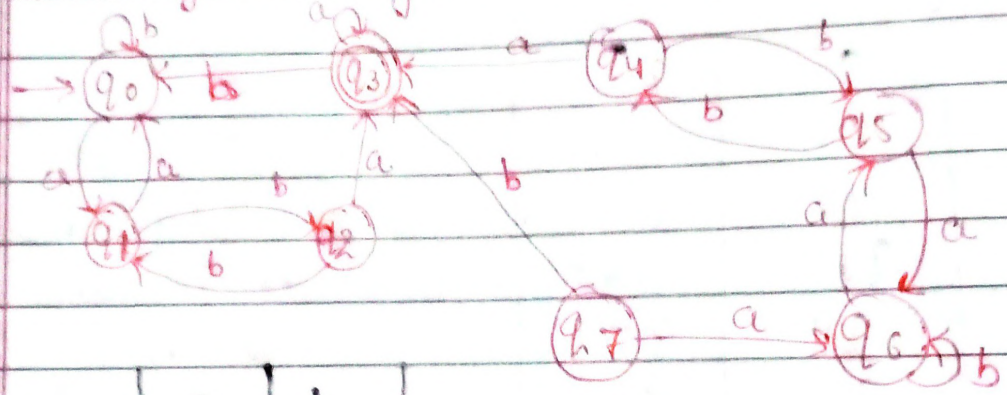
1 Equivalence
{q0, q4, q6} {q1, q7} {q5} {q2}

2. Equivalence
{q0, q4} {q6} {q1, q7} {q5} {q2}

3- Equivalence
{q0, q4} {q6} {q1, q7} {q5} {q2}



Q: Minimize the given DFA



	a	b
→q0	q1	q0
q1	q0	q2
q2	q3	q1
(q3)	q3	q0
q4	q3	q5
q5	q6	q4
q6	q5	q6
q7	q6	q3

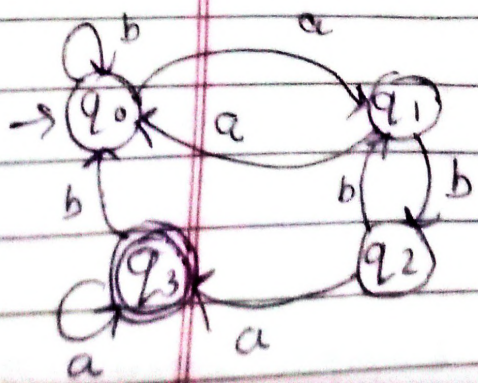
not reachable state

0- Equivalence
{q0, q1, q2} {q3}

1- Equivalence
{q0, q1} {q2} {q3}

2- Equivalence
{q0} {q1} {q2} {q3}

3- Equivalence
{q0} {q1} {q2} {q3}



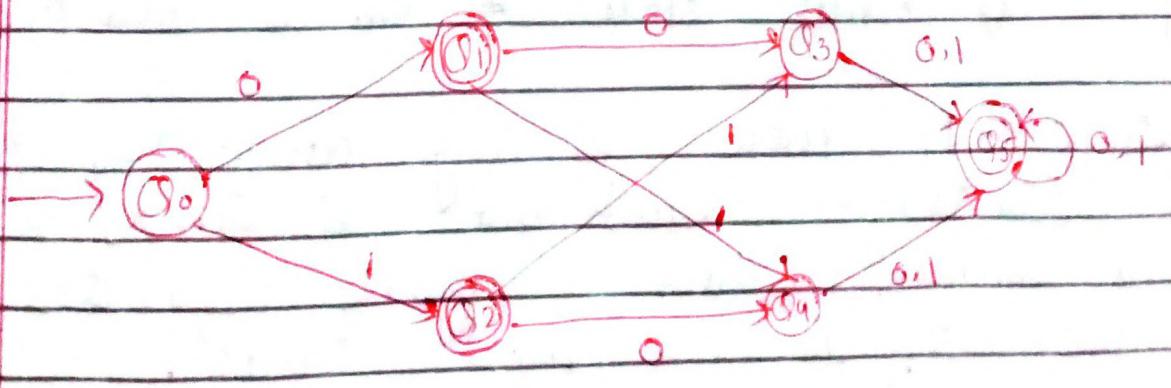
* Minimization of DFA using Myhill-Nerode Theorem

Myhill-nerode theorem can be used to convert a DFA to its equivalent DFA with minimum no of states. This method of minimization is also called table filling Method.

Steps for the Minimize DFA using Myhill Nerode Theorem:

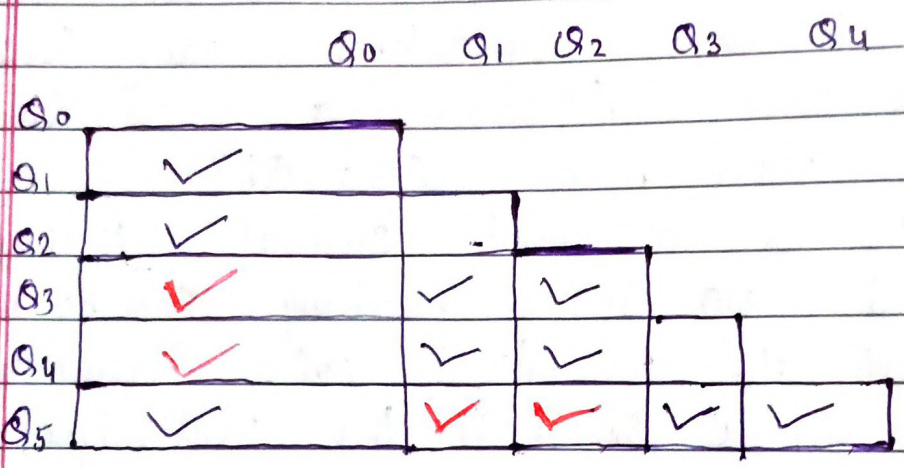
1. Create the pairs of all the states involved in the given DFA
2. Mark all the pairs (Q_a, Q_b) such that Q_a is final state & Q_b is Non-final state.
3. If there is any unmarked pair (Q_a, Q_b) such a that $\delta(Q_a, x)$ and $\delta(Q_b, x)$ is marked, then mark (Q_a, Q_b).
   Here 'x' is a input symbol. Repeat this step untill no more marking can be made.
4. Combine all the unmarked pairs and make them a single state in the minimized DFA.

Example:-



States	Inputs	
	0	1
→ Q_0	Q_1	Q_2
(Q_1)	Q_3	Q_4
(Q_2)	Q_4	Q_3
Q_3	Q_5	Q_5
Q_4	Q_5	Q_5
(Q_5)	Q_5	Q_5

(1) * Step 1:- Create the pairs of all the states involved in DFA:



(2) * Step 2:- Mark all the pairs (Q_a, Q_b) such that Q_a is final state & Q_b is non final state

(3) * Step 3:- If there is any unmarked pair (Q_a, Q_b) such that $\delta(Q_a, x)$ & $\delta(Q_b, x)$ is marked, then mark (Q_a, Q_b). Here x is a input symbol. Repeat this step until no more marking can be made.

⇒ Check for the unmarked pair, Q_2, Q_1

- Check when $x=0$

  $\delta(Q_2,0) = Q_4$ & $\delta(Q_1,0) = Q_3$ now check if the pair Q_4, Q_3 is marked & no it is not marked.

- Check when $x=1$

  $\delta(Q_2,1) = Q_3$ & $\delta(Q_1,1) = Q_4$, now Check if the pair Q_4, Q_3 is marked & no it is not marked.

- Hence we cannot mark the pair Q_2, Q_1

⇒ Check Q_3, Q_0

$x=0$

$\delta(Q_3,0) = Q_5$ & $\delta(Q_0,0) = Q_1$

pair Q_5, Q_1 is unmarked

$x=1$

$\delta(Q_3,1) = Q_5$ & $\delta(Q_0,1) = Q_2$

pair Q_5, Q_2 is unmarked, so we can't mark Q_3, Q_0

⇒ Check Q_4, Q_3

$x=0$

$\delta(Q_4,0) = Q_5$ & $\delta(Q_3,0) = Q_5$

$x=$ Such pair of states Q_5, Q_5 don't exists

$x=1$

$\delta(Q_4,1) = Q_5$ & $\delta(Q_3,1) = Q_5$

pair of states Q_5, Q_5 don't exists

Hence we cannot mark the pair Q_4, Q_3

⇒ Check unmarked pair Q_4, Q_0

$x = 0$

$\delta(Q_4, 0) = Q_5$ & $\delta(Q_0, 0) = Q_1$

pair Q_5 Q_1 unmarked

$x = 1$

$\delta(Q_4, 1) = Q_5$ & $\delta(Q_0, 1) = Q_2$

pair Q_5, Q_2 unmarked

Hence we cannot mark the pair Q_4, Q_0

⇒ Check for the unmarked pair Q_5, Q_1

$x = 0$

$\delta(Q_5, 0) = Q_5$ & $\delta(Q_1, 0) = Q_3$

Q_5, Q_3 is marked,

Hence we can mark the pair Q_5 Q_1

$x = 1$

$\delta(Q_5, 1) = Q_5$ & $\delta(Q_1, 1) = Q_4$

Q_5, Q_4 is marked

Hence we can mark the pair Q_5, Q_1

⇒ Check for the unmarked pair Q_5, Q_2

$x = 0$

$\delta(Q_5, 0) = Q_5$ & $\delta(Q_2, 0) = Q_4$

pair Q_5, Q_4 is marked

Hence we can mark the pair Q_5, Q_2

$x = 1$

$\delta(Q_5, 1) = Q_5$ & $\delta(Q_2, 1) = Q_3$

Q_5, Q_3 is marked.

* We have checked for all the unmarked pairs but dont need to stop here we need to continue the process untill no more markings can be made.

=> unmarked pair Q_2, Q_1

$\delta(Q_2, 0) = Q_4$ & $\delta(Q_1, 0) = Q_3$

Q_4, Q_3 is unmarked pair

$\delta(Q_2, 1) = Q_3$ & $\delta(Q_1, 1) = Q_4$

pair Q_3 Q_4 is unmarked pair

cannot mark the pair Q_2, Q_1.

=> check unmarked pair Q_3, Q_0

$\delta(Q_3, 0) = Q_5$ $\delta(Q_0, 0) = Q_1$

Q_5, Q_1 is marked

Hence we can mark the pair Q_3, Q_0

$\delta(Q_3, 1) = Q_5$ $\delta(Q_0, 1) = Q_2$

Q_5, Q_2 is marked.

pair Q_3, Q_0 is marked

=>

Check Q_4, Q_0

$\delta(Q_4, 0) = Q_5$ $\delta(Q_0, 0) = Q_1$

Q_5, Q_1 is marked

Hence we can marked Q_4, Q_0.

$\delta(Q_4, 1) = Q_5$ $\delta(Q_0, 1) = Q_2$

Q_5, Q_2 marked

⇒ Check Q4 Q3
$\delta(Q_4, 0) = Q_5$ & $\delta(Q_3, 0) = Q_5$
Q_5, Q_5 pair don't exists
Q_4, Q_3 don't mark

$\delta(Q_4, 1) = Q_5$ & $\delta(Q_3, 1) = Q_5$
States Q_5, Q_5 don't exists.

Hence we can not mark the pair Q_4, Q_3

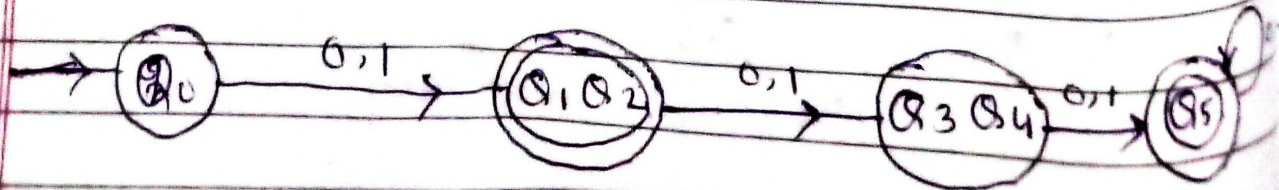
★ ~~Hence~~ Now even though we repeat the procedure we cannot mark the pairs Q_2, Q_1 (since Q_4, Q_3 is not marked) & Q_4, Q_3 (since Q_5, Q_5 such pair of states does not exists).

Hence we stop here.

④ ★ Step 4:- Combine all the unmarked pairs & make them as a single state in the minimized DFA.

The unmarked Pairs Q_2, Q_1 & Q_4, Q_3 hence we combine them.

Following is the Minimized DFA with $Q_1 Q_2$ & $Q_3 Q_4$ as the combined states.

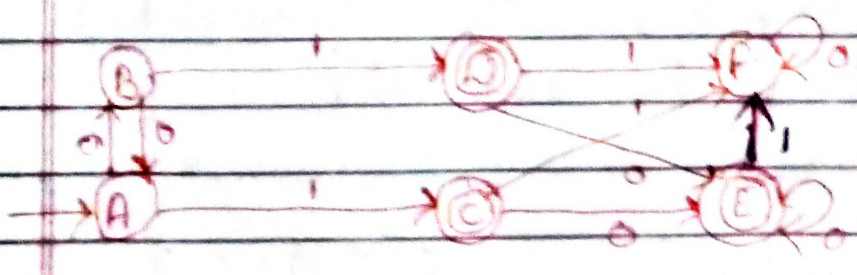


- Q_0 remains as our starting state.
- Q_1 & Q_2 were our final states so even we combine them they will remain as the combined final state.
- Q_5 is the another final state we have.

Transition table for minimized DFA

States	Inputs	
	0	1
→ Q_0	$Q_1 Q_2$	$Q_1 Q_2$
($Q_1 Q_2$)	$Q_3 Q_4$	$Q_3 Q_4$
$Q_3 Q_4$	Q_5	Q_5
(Q_5)	Q_5	Q_5

Sol Minimized the given DFA using Myhill Nerode theorem



A B C D E F

A
B
C ✓ ✓
D ✓ ✓
E ✓ ✓
F ✓ ✓ ✓ ✓ ✓

Page No
Date

Checks unmarked pair

• (B,A) → $\delta(B,0) = A$, $\delta(B,1) = D$
$\delta(A,0) = B$, $\delta(A,1) = C$
(A,B) & (D,C) are unmarked.

• (D,C) → $\delta(D,0) = E$, $\delta(D,1) = F$
$\delta(C,0) = E$, $\delta(C,1) = F$
(E,E) & (F,F) not exists

• (E,C) →
$\delta(E,0) = E$, $\delta(E,1) = F$
$\delta(C,0) = E$, $\delta(C,1) = F$
(E,E) & (F,F) pair not exists.

• (E,D) →
$\delta(E,0) = E$, $\delta(E,1) = F$
$\delta(D,0) = E$, $\delta(D,1) = F$
(E,E) & (F,F) not exists

• (F,A) →
$\delta(F,0) = F$, $\delta(F,1) = F$
$\delta(A,0) = B$, $\delta(A,1) = C$
(F,B) not marked but (F,C) is marked
that's why
we can mark (F,A)

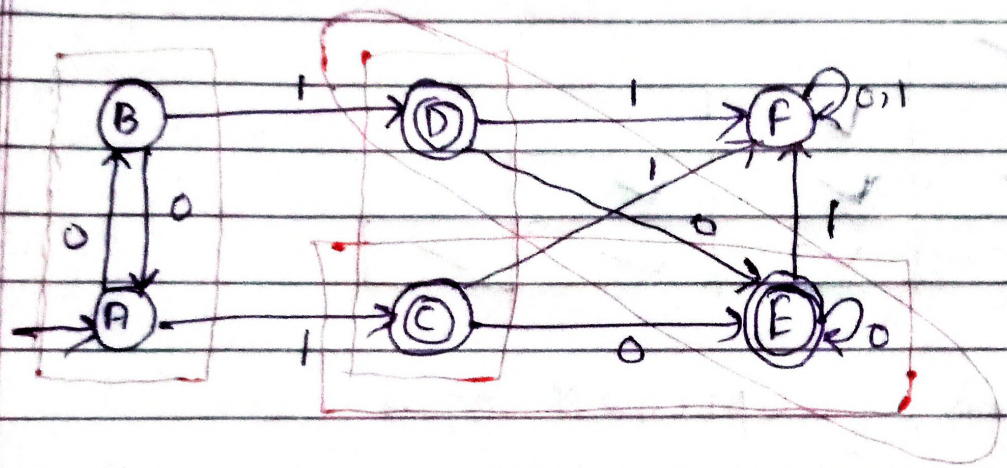
• (F,B) → $\delta(F,0) = F$, $\delta(F,1) = F$
$\delta(B,0) = A$, $\delta(B,1) = D$
F,A is marked (F,D) is marked
we can mark (F,B)

→ Repeat this until no more markings can be made

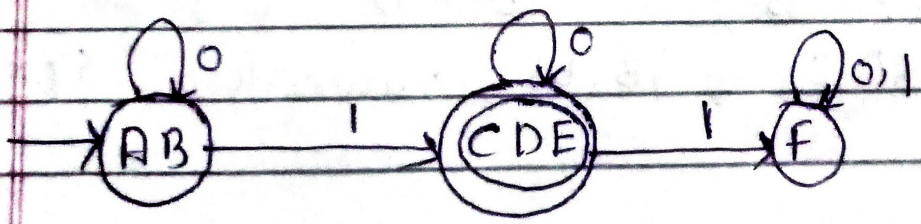
→ after ~~getti~~ repeating we getting same pair of states unmarked, so stop.

* combine all the unmarked pair.

(A, B) (C, D) (E, C) (E, D)

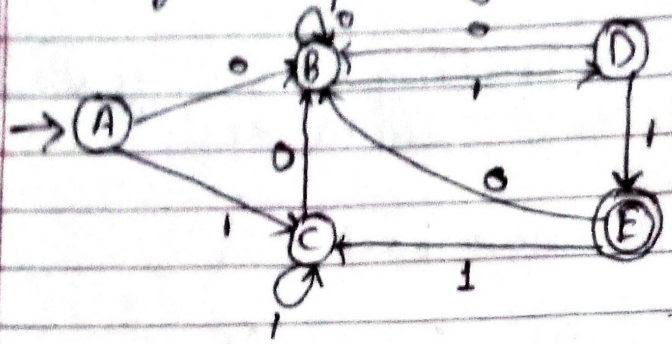


Minimized DFA



States	Inputs	
	0	1
→AB	AB	CDE
(CDE)	CDE	F
F	F	F

Q. Minimize the following DFA using Table filling Method



	A	B	C	D	E
A					
B	✓				
C		✓			
D	✓	✓	✓		
E	✓	✓	✓	✓	

Now check : unmarked states

* (B, A)

$\delta(B,0) = B$ } $\delta(B,1) = D$ }
$\delta(A,0) = B$ } $\delta(A,1) = C$ }

Both pairs {D,C} & {B, B} unmarked, {B,B} pair doesn't exits.

* (C, A)

$\delta(C,0) = B$ } $\delta(C,1) = C$ }
$\delta(A,0) = B$ } $\delta(A,1) = C$ }

pair (B, B) & (C, C) not exists.

* (C, B)

$\delta(C,0) = B$ } $\delta(C,1) = C$ }
$\delta(B,0) = B$ } $\delta(B,1) = D$ }

{B, B} not exists & C, D is unmarked

* (D, A)

$\delta(D,0) = B$ } $\delta(D,1) = E$ }
$\delta(A,0) = B$ } $\delta(A,1) = C$ }

pair (BB) not exists But pair {E, C} is marked
So (D, A) also marked

* (D, B)

$\delta(D,0) = B$ } $\delta(D,1) = E$ }
$\delta(B,0) = B$ } $\delta(B,1) = D$ }

{BB} not exist But {E, D} is marked States.
So (D, B) will be marked

* (D, C)

$\delta(D,0) = B$ } $\delta(D,1) = E$ }
$\delta(C,0) = B$ } $\delta(C,1) = C$ }

pair (BB) not exists but (E C) pair is marked
So (D, C) will be marked.

Now check again Pair (B, A)

* (B, A)

$\delta(B,0) = B$ } $\delta(B,1) = D$ }
$\delta(A,0) = B$ } $\delta(B,1) = C$ }

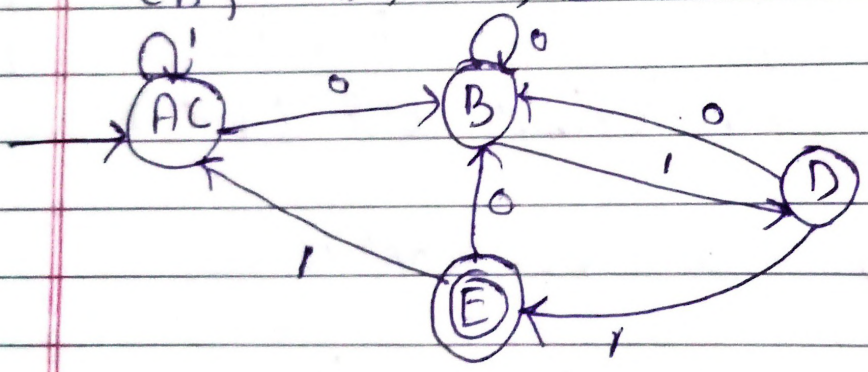
(BB) not exists, but pair (D, C) is marked,
So (B, A) will be marked

* (C,A)
$\delta(C,0) = B$ } $\delta(C,1) = C$ }
$\delta(A,0) = B$ } $\delta(A,1) = C$ }
Both pair (B,B) & (C,C) nad exist

* (C,B)
$\delta(C,0) = B$ } $\delta(C,1) = C$ }
$\delta(B,0) = B$ } $\delta(B,1) = D$ }
(B B) nat exists But (C,D) is marked
So (C,B) will be marked

now ~~mer~~ combine all unmarked pair

CA, B, D, E



Minimized DFA.

ε-NFA { NFA with ε-Transition }

Non deterministic finite automata (NFA) is a finite automata where for some cases when a specific input is given to the current state, the machine goes to multiple states or more than 1 states. It can contain ε move. It can contain ε move. It can be represented as five tuples M (Q, Σ, δ, q0, F)

where

Q:

Σ:

q0:

F:

δ: $\delta \rightarrow Q \times (\Sigma \cup \lambda) = 2^Q$

* NFA with ε move :- If any FA contains ε transition or move, the finite automata is called NFA with ε move

ε-closure :- ε-closure for a given state A means a set of states which can be reached from the state A with only ε (null) move including the state A itself.

Steps for converting NFA with ε to DFA

Step 1: We will take the ε-closure for the starting state of NFA as a starting state of DFA

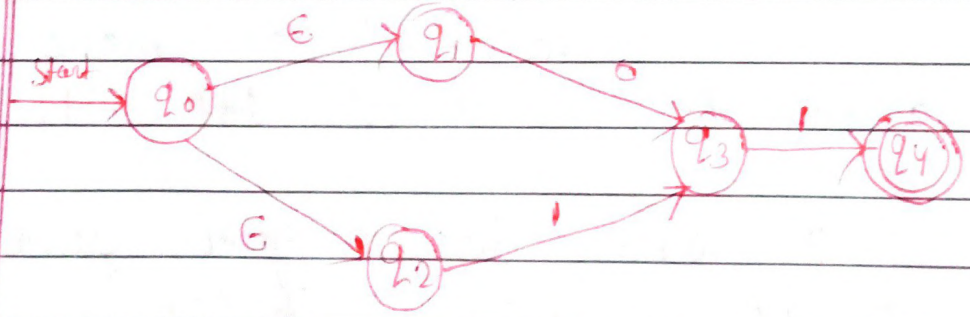
Step 2:- Find the states for each input symbol that can be traversed from the present. That means the union of transition value and their closure for each state of NFA present in the current state of DFA.

Step 3:- If we found a new state, take it as current state & repeat step 2.

Step 4:- Repeat step 2 & Step 3 until there is no new state present in the transition table of DFA.

Step 5:- Mark the states of DFA as a finite state which contains the final state of NFA

Eg:- Convert the NFA with ε into its equivalent DFA.



Soln:- Let us obtain ε-closure of each state

ε-closure {q0} = {q0, q1, q2}

ε-closure {q1} = {q1}

ε-closure {q2} = {q2}

ε-closure {q3} = {q3}

ε-closure {q4} = {q4}

Now,

Let ε-closure $\{q_0\} = \{q_0, q_1, q_2\}$ be state A

Hence

$\delta(A,0) = \varepsilon$-closure $\{\delta(q_0,q_1,q_2),0\}$

$= \varepsilon$-closure $\{\delta(q_0,0) \cup \delta(q_1,0) \cup \delta(q_2,0)\}$

$= \varepsilon$-closure $\{q_3\}$

$= \{q_3\}$ — call it as state B

$\delta(A,1) = \varepsilon$-closure $\{\delta(q_0,q_1,q_2),1\}$

$= \varepsilon$-closure $\{\delta(q_0,1) \cup \delta(q_1,1) \cup \delta(q_2,1)\}$

$= \varepsilon$-closure $\{q_3\}$

$= \{q_3\}$ — B

Now - for state B

$\delta(B,0) = \varepsilon$-closure $\{\delta(q_3,0)\}$

$= \phi$

$\delta(B,1) = \varepsilon$-closure $\{\delta(q_3,1)\}$

$= \varepsilon$-closure $\{q_4\}$

$= \{q_4\}$ call it as state C

Now - for state C

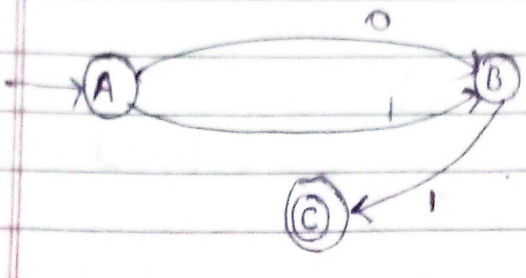
$\delta(C,0) = \varepsilon$-closure $\{\delta(q_4,0)\}$

$= \phi$

$\delta(C,1) = \varepsilon$-closure $\{\delta(q_4,1)\}$

$= \phi$

The NFA will be



for DFA

	0	1
$\{q_0, q_1, q_2\}$	$\{q_3\}$	$\{q_3\}$
$\{q_3\}$	$\{\phi\}$	$\{q_4\}$
$\{q_4\}$	$\{\phi\}$	$\{\phi\}$
$\{\phi\}$	$\{\phi\}$	$\{\phi\}$

$q_0, q_1, q_2 \rightarrow A$

$\delta\{(q_0, q_1, q_2), 0\}$
$\rightarrow \epsilon$ (closure $\{(q_0,0) \cup (q_1,0) \cup (q_2,0)\}$
$\rightarrow \epsilon$ - closure $\{\phi \cup q_3 \cup \phi\}$
$\rightarrow \epsilon$ - closure $\{q_3\}$
$\rightarrow \{q_3\}$ — B

$\{\delta\{(q_0, q_1, q_2), 1\}$
$\delta\{(q_0,1) \cup (q_1,1) \cup (q_2,1)\}$
ϵ-closure $\{\phi \cup \phi \cup q_3\}$
ϵ-closure $\{q_3\}$
$\{q_3\}$ — B

for state B
$\delta\{(q_3, 0)\}$
$\rightarrow \epsilon$-closure $\{q_3, 0\}$
$\rightarrow \epsilon$ - closure $\{\phi\}$
$\rightarrow \phi$ — C

$\delta\{(q_3, 1)\}$
$\rightarrow \epsilon$ - closure $\{q_3, 1\}$
$\rightarrow \epsilon$ - closure $\{q_4\}$
$\rightarrow \{q_4\}$ — D

$\delta(\phi, 0)$

ε-closure $(\phi, 0)$

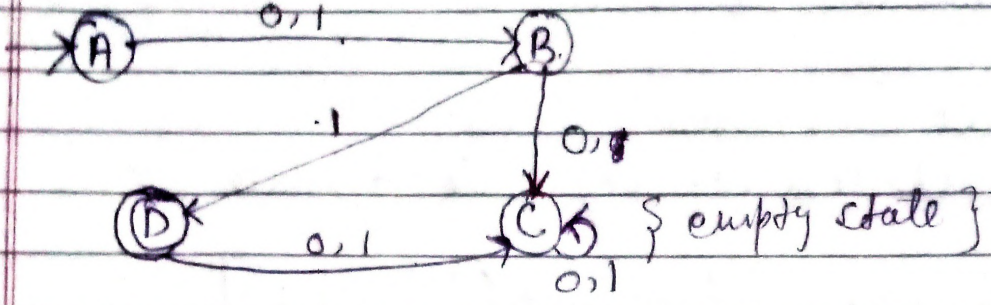
$\Rightarrow \{\phi\}$

$\delta(\phi, 1)$

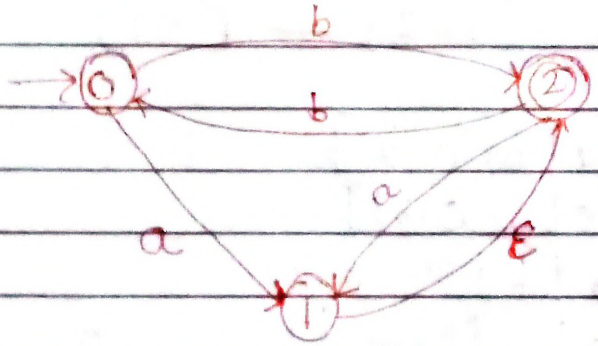
ε-closer $(\phi, 1)$

$\Rightarrow \{\phi\}$

resulted DFA will be



Ques Convert the NFA ε to DFA and gives definition of ε-closure in FA.



ε-closure FA:-

It means that if there's an ε-transition [a transition where the automaton can move without consuming any input) from one state to another, then the epsilon closure of a state is the set of all states reachable from that state using only ε-transitions.

It helps in understanding all possible states the automaton can be in, including those reachable through epsilon transitions

First ε-NFA to NFA conversion:

ε closure of each states

ε closure (0) = {0}
ε-closure (1) = {1, 2}
ε closure (2) = {2}

Now
let consider
ε-closure (0) = {0} is a state [q0]

Hence
$\delta(q_0, a) = \varepsilon\text{-closure}[\delta(q_0, a)]$
$= \varepsilon\text{-closure}[\delta(0, a)]$
$= \varepsilon\text{-closer}[1]$
find ε closure of [1] = {1, 2} — [q1]

$\delta(q_0, b) = \varepsilon\text{-closure}[\delta(q_0, b)]$
$= \varepsilon\text{-closure}[\delta(0, b)]$
$= \varepsilon\text{-closure}[2]$
find ε-closure of [2] ⇒ {2} — [q2]

* Now for state [q1]

$\delta[q_1, a] = \varepsilon\text{-closure}[\delta(q_1, 0)]$
$= \varepsilon\text{-closure}(\delta\{(1, a) \cup (2, a)\})$
$= \varepsilon\text{-closure}[\delta(\phi \cup 1)]$
$= \varepsilon\text{-closer}[1]$
⇒ [1, 2] — [q1]

$\delta[q_1, b] = \varepsilon\text{-closure}[\delta(q_1, b)]$
$= \varepsilon\text{-closure}[\delta\{(1, b) \cup (2, b)\}]$
$= \varepsilon\text{-closure}[\delta(\phi \cup 0)]$

ε closure [0]

⇒ {0} → [q_0]

for state [q_2]

$\delta[q_2, a]$ = ε closure ($\delta(q_2, a)$)

= ε closure ($\delta(2, a)$)

= ε closure { }

= {1, 2} → [q_1]

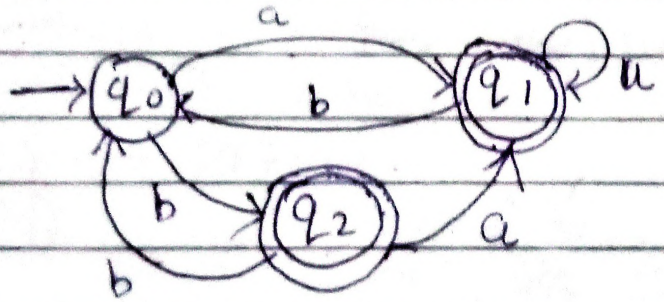
$\delta[q_2, b]$ = ε closure ($\delta(q_2, b)$)

= ε closure ($\delta(2, b)$)

= ε closure {0}

= {0} → [q_0]

resultant NFA



For DFA

		a	b
A	{0}	{1, 2}	{2}
B	{1, 2}	{1, 2}	{0}
C	{2}	{1, 2}	{0}

δ{0, a} ⇒ ε-closure (0, a) = {1} = ε closure of 1

{1, 2}

δ{0, b} = ε-closure (0, b) = {2}

$\{\delta\{1,2\}, a\}$

ε-closure $\delta[(1,a) \cup (2,a)]$

ε-closer $\delta(\phi \cup 1) = \varepsilon$-closure (1)

$= \{1,2\}$

$\{\delta\{1,2\}, b\}$

ε-closure $\delta[(1,b) \cup (2,b)]$

ε-closure $\delta[\phi \cup \textcircled{0}] = \varepsilon$-closure (0)

$\Rightarrow \{0\}$

$\}\delta\{2,a)\}$

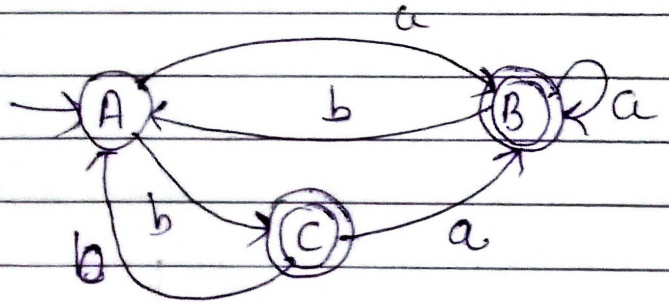
ε-closure $\delta[(2,a)]$

ε-closure $\{1\} = \{1,2\}$

$\}\delta(2,b)\}$

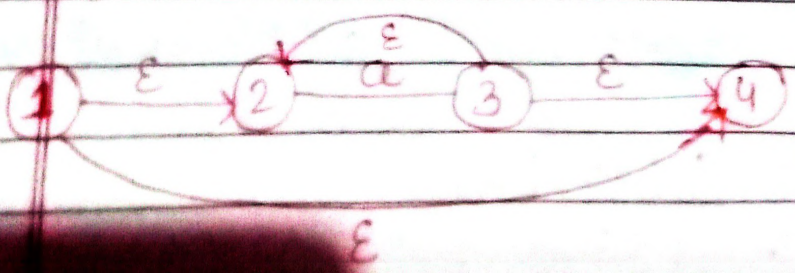
ε-closure $\delta[(2,b)]$

ε-closure $\{0\} = \{0\}$



Both DFA & NFA are same for the given ε-NFA.

Ques Compute the epsilon-closure for the given NFA. convert it into DFA.



ε closure of each states

ε closure of (1) = {1, 2, 4}

ε closure (2) = {2}

ε-closure (3) = {3, 2, 4}

ε closure (4) = {4}

FOR DFA

	a	b
{1, 2, 4} - A	{3, 2, 4}	
{3, 2, 4} - B	{3, 2, 4}	

$\delta\{(1,2,4),a\}$ = ε closure $\{(1,a) \cup (2,a) \cup (4,a)\}$

= ε-closure $\{\phi \cup 3 \cup \phi\} = \{3\} \Rightarrow \{3,2,4\}$

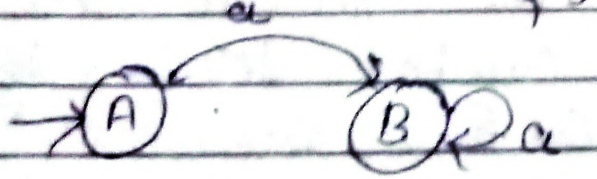
~~$\delta\{(1,2,4),b\}$ = ε-closure $\{(1,b) \cup (2,b) \cup (4,b)\}$~~

~~= ε-closure $\{\phi \cup$~~

$\delta((3,2,4),a)$ = ε- closure $\{(3,a) \cup (2,a) \cup (4,a)\}$

= ε-closer $\{\phi \cup 3 \cup \phi\}$ = ε-closure $\{3\}$

= $\{3,2,4\}$



— DFA

: Finite Automata with output :

① Moore Machine — { Edward F Moore }
② Mealy Machine. — { George H Mealy }

* Moore & Mealy Machine
⇒ Both moore & mealy machine are special case of DFA.
⇒ Both acts like O/P producers rather than Language acceptors.
⇒ In moore & mealy machine no need to define the final states.
⇒ No concepts of Dead states & no concepts of final states.
⇒ Mealy & Moore Machines are equivalent.

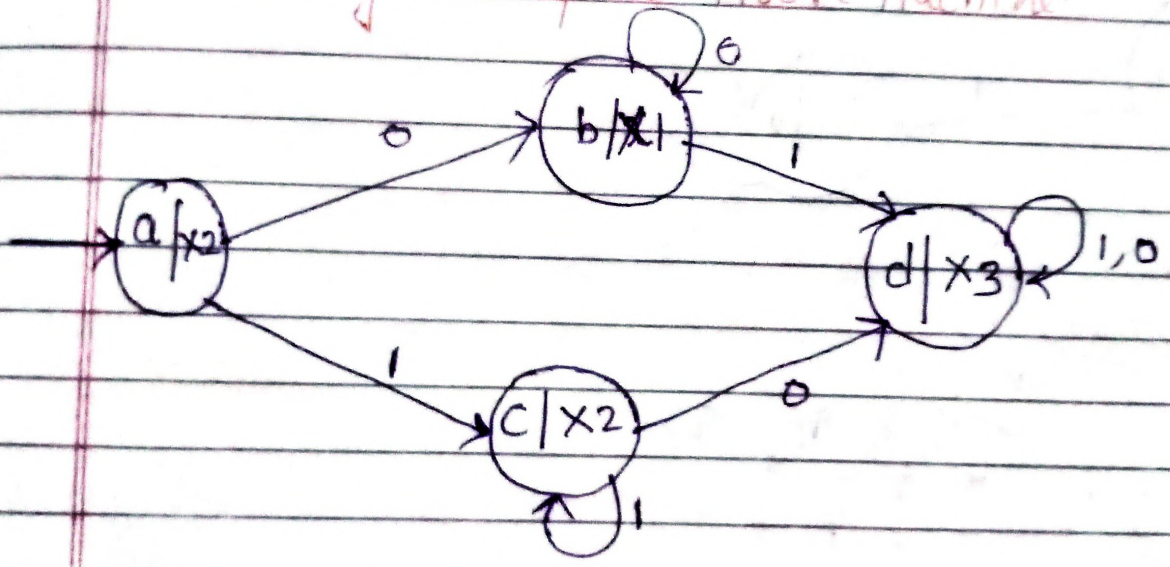
* Moore Machine :-

• Moore machine is an FSM whose outputs depends on only the present state.
• A moore machine can be described by a 6 tauple (Q, Σ, O, δ, X, q0)
where
Q : is a finite set of states
Σ : is a finite set of symbols called input alphabet
O : is a finite set of symbol called output alphabet
δ : is the transition function where δ : Q × Σ → Q
X : is the output transition function where X : Q → O
q0 : is the initial state from where any input is processed (q0 ∈ Q).

State table of Moore Machine

Present state	Next state		output
	i/p: 0	i/p: 1	λ
→ a	b	c	x2
b	b	d	x1
c	c	d	x2
d	d	d	x3

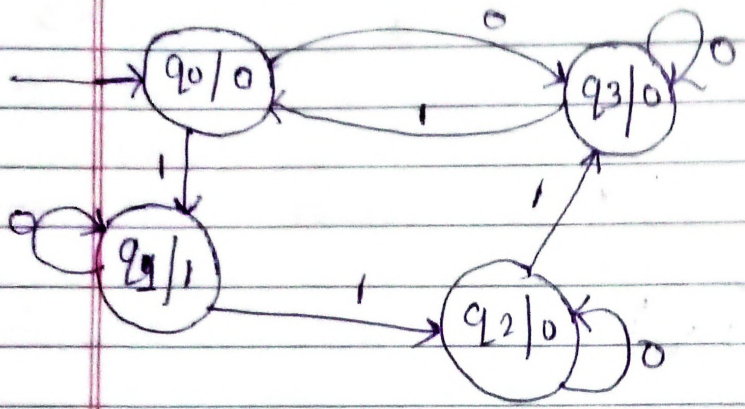
State diagram of the Moore Machine



- In moore machine for every state output is associated.
- Moore machine response for empty string ε.
- If the length of i/p string is n, then length of o/p string will be n+1

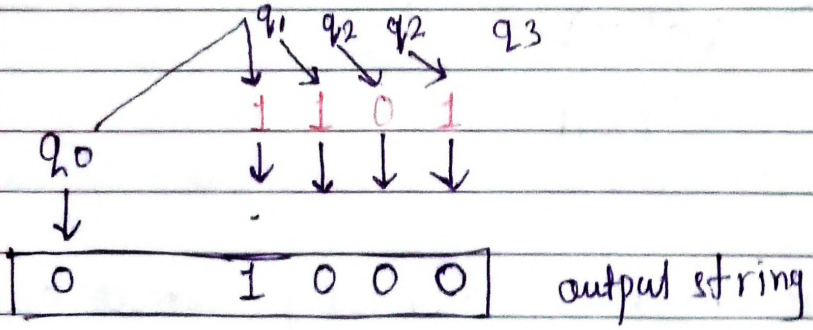
Eg:- When length of string is n, then length of o/p string is n+1.

Present state	Next state		Output
	a=0	a=1	
→ q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0



Let i/p string is 1101

Length => (n = 4)



Length of o/p string is = 5.

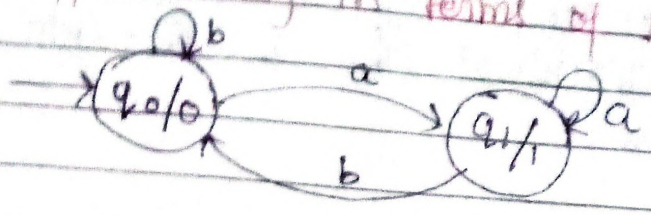
Which is n+1 = 4+1 => 5

↓

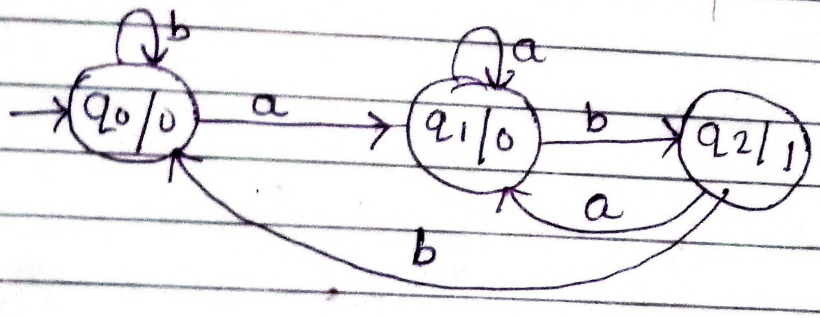
o/p string length

Construction of Moore Machine:

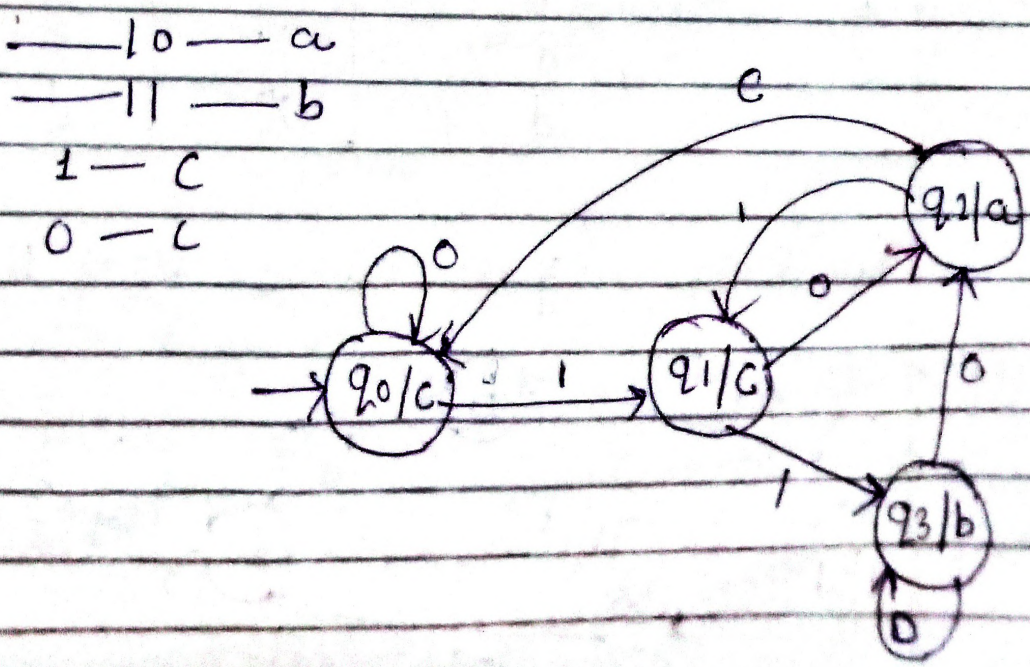
Que Construct a moore machine take all the string of a's & b's as i/p and counts the no of a's in the i/p string in terms of 1, $\Sigma = \{a, b\}$ $\Delta = \{0, 1\}$?



Que Construct a Moore machine take all the string of a's & b's as i/p and counts the no of occurrence of sub string 'ab' in terms of 1, $\Sigma = \{a, b\}$, $\Delta = \{0, 1\}$?



Que Construct a Moore machine wherever $\Sigma = \{0, 1\}$, $\Delta = \{a, b, c\}$ machine should give o/p a, if the i/p string ends with 10, b if i/p string end with 11, c otherwise?



Mealy

* Mealy Machine:-

⇒ A mealy machine is an FSM whose output depends on the present state as well as the present input.

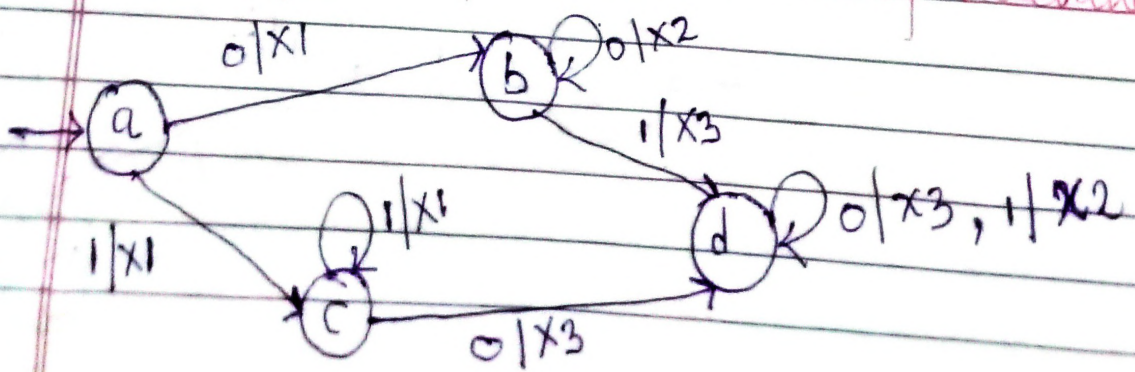
⇒ It can be described by a 6 tuple $(Q, \Sigma, O, \delta, \lambda, q_0)$ where-

Q: Set of states
Σ: input alphabet
O: output alphabet
δ: $Q \times \Sigma \rightarrow Q$ [i/p transition function]
λ: $Q \times \Sigma \rightarrow Q$ [o/p transition function]
q_0: initial state

* State table of a Mealy Machine

Present state	Next state			
	input = 0		input = 1	
	State	o/p	State	o/p
→ a	b	X1	c	X1
b	b	X2	d	X3
c	d	X3	c	X1
d	d	X3	d	X2

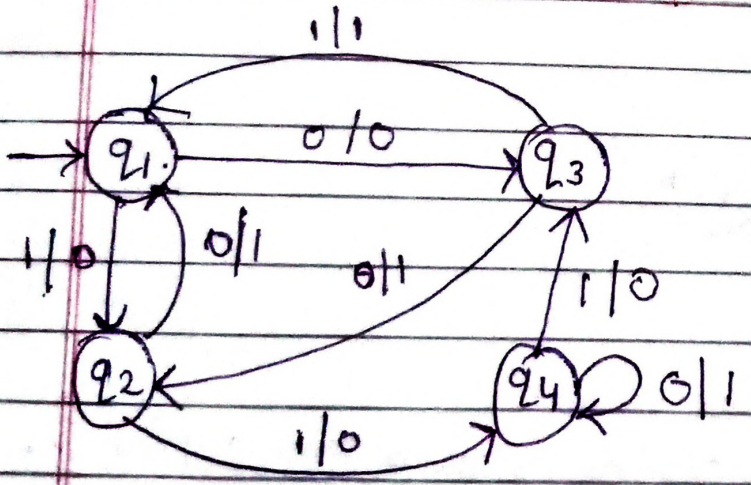
State diagram of the above Mealy Machine



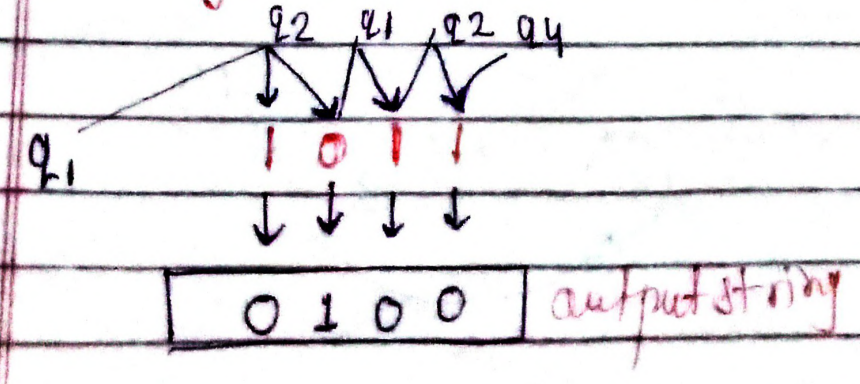
=> Melay Machine do not response for empty string ε.
=> If the Length of i/p string is [n] then Length of o/p string will be [n].

Eg: When the Length of i/p string is 'n' is melay machine then Length of output string is also n.

Present state	Next state			
	a=0		a=1	
	State	o/p	State	o/p
→q1	q3	0	q2	0
q2	q1	1	q4	0
q3	q2	1	q1	1
q4	q4	1	q3	0



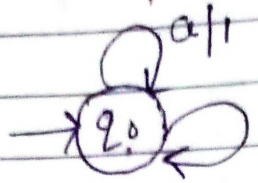
Let i/p string is 101
Length (n) = 4



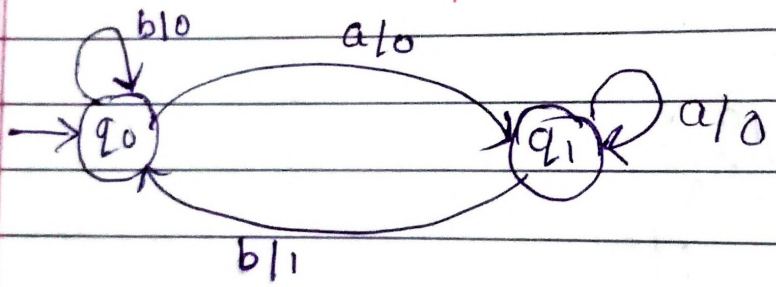
Length of o/p str is also 4

* Construct of Melay Machine:

Ques Construct a Melay machine take all the string of a's and b's as i/p and counts the no of a's in the i/p string in terms of 1, $\Sigma = \{a, b\}$, $\Delta = \{0, 1\}$?

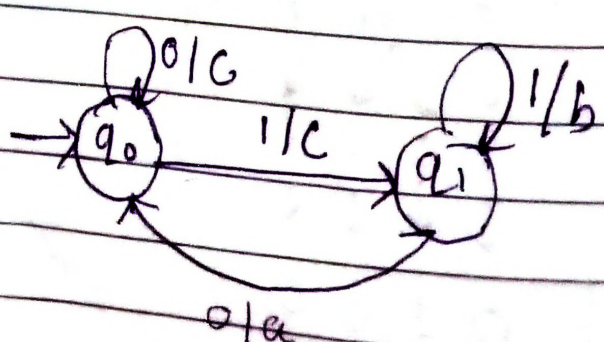


Ques Construct a Melay machine take all the string of a's & b's as i/p and counting the no of occurrences of sub-string 'ab' in terms of 1, $\Sigma = \{a, b\}$ $\Delta = \{0, 1\}$?

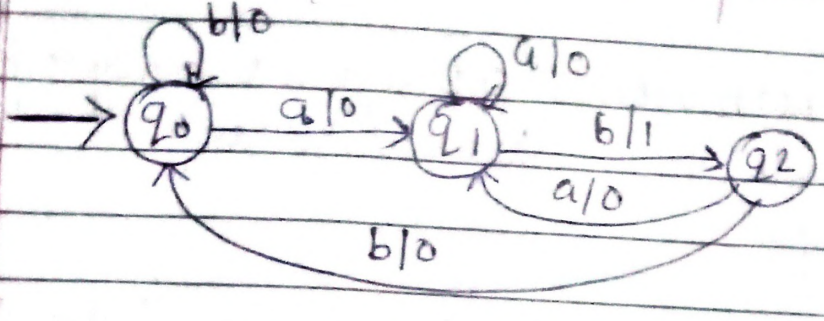


Ques Construct a Melay machine where $\Sigma = \{0, 1\}$, $\Delta = \{a, b, c\}$ machine should give o/p a, if the i/p string ends with 10, b if i/p strings ends with 11, c otherwise?

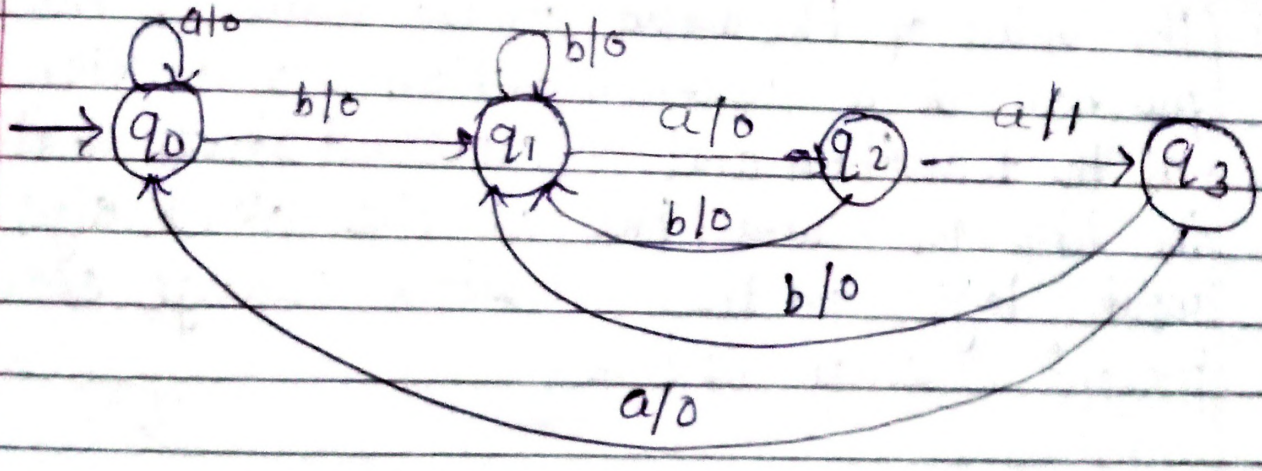
10 — a
11 — b
1 — c
0 — c



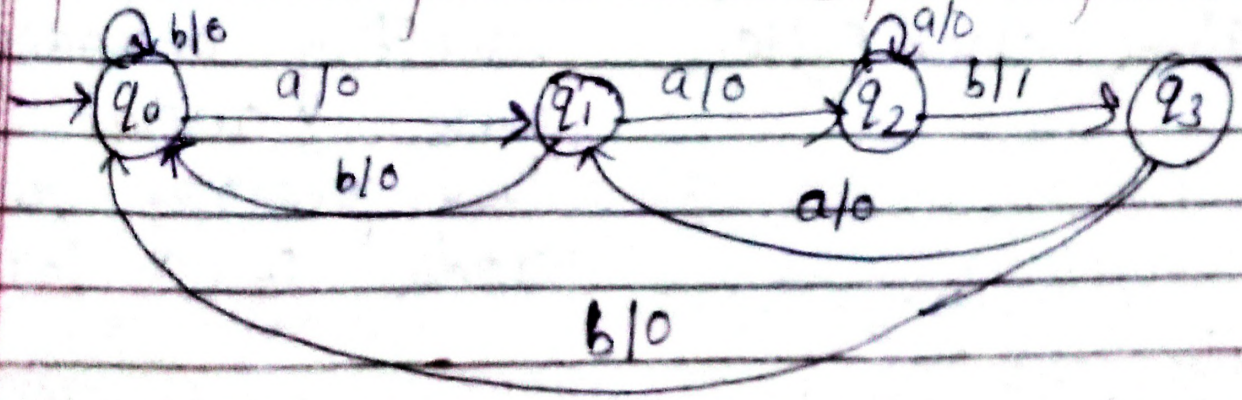
Sol Construct a melay machine that takes set of all string over {a, b} as i/p & print 1 as o/p for every occurrence of 'ab' as a substring.



Sol Construct a Melay machine that takes set of all string over {a, b} any counts no of occurance of substring 'baa'



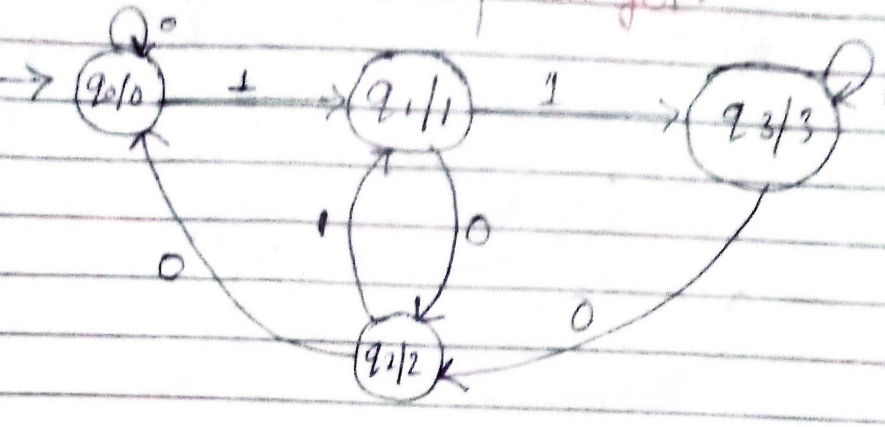
Sol Design a Melay M/C which count the occurrence of substring 'acb' in i/p string.



Melay Machine VS Moore Machine

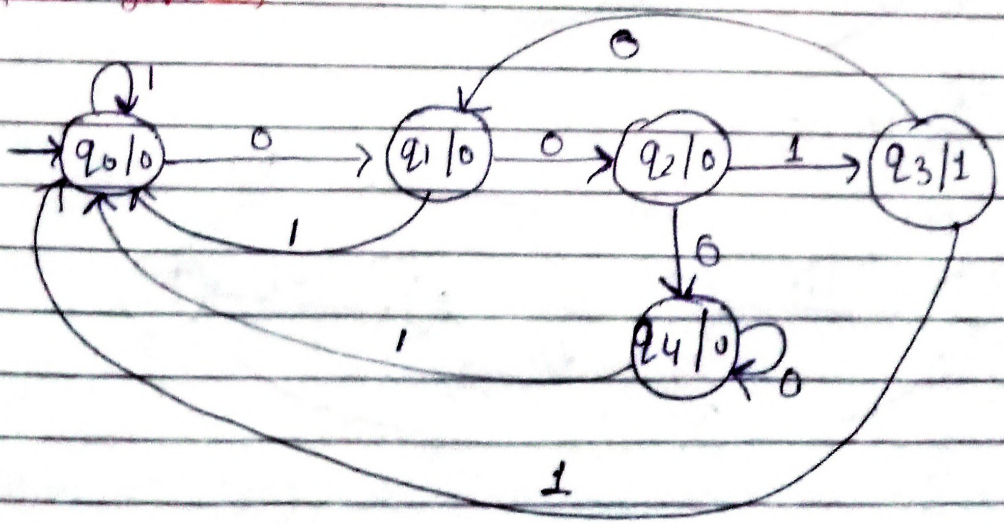
	Melay Machine	Moore Machine.
(i)	output depends both upon the present state & the present input	(i) o/p depends only upon the present state.
(ii)	Generally, it has fewer states than moore machine.	(ii) has more states than Melay Machine
(iii)	The value of the output function is a function of the transition and the changes, when the input logic on the present state is done.	(iii) The value of the o/p function is a function of the current state & the change at the clock edges, whenever state change occur.
(iv)	Melay Machine react faster to inputs They generally react in the same clock cycle.	(iv) In moore machine, more logic is required to decode the outputs resulting in ~~moore machine~~ more circuit delays They generally react one clock cycle later.
(v)	output length = i/p length $n = n$	(v) if input length = n then o/p $n+1$
(vi)	It is difficult to design	(vi) It is easy to design

Sol construct a Moore M/C which calculates residue mode 4 for each binary string treated as binary integer.

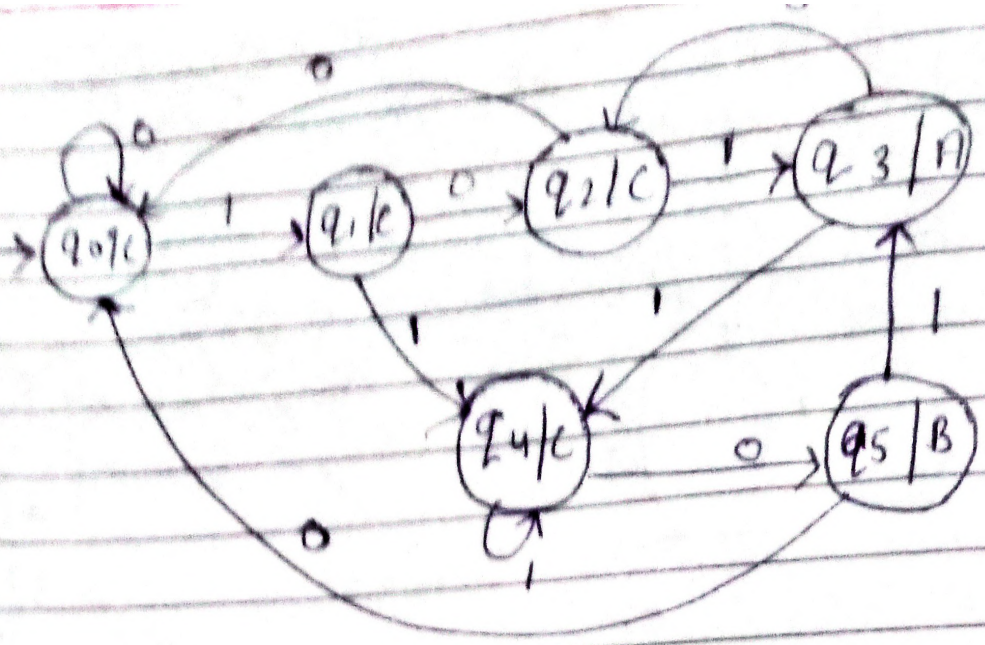


000 - 0
001 - 1
010 - 2
011 - 3
100 - 4
101 - 5
110 - 6
111 - 7

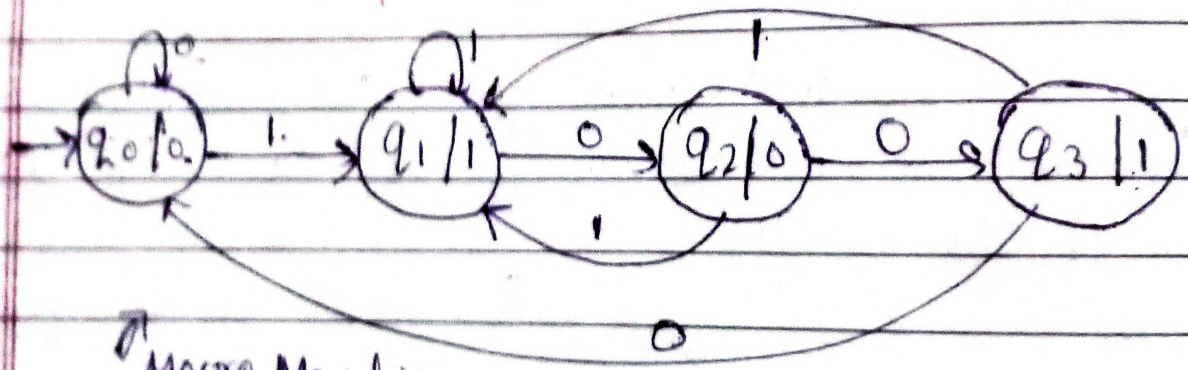
Ans Design a Moore m/c that gives an o/p 1 if input of binary sequence a '1' is preceded by exactly two zeroes.



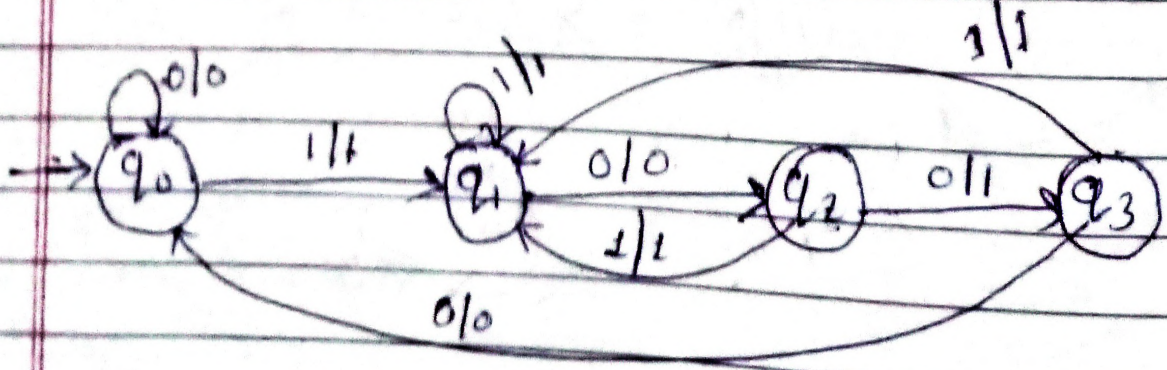
Sol Design a Moore M/C for a binary I/P sequence if it ends in 101 o/p is A, if it ends in 110 o/p is B.



Ans Design a Moore & Mealy Machine to convert each occurrence of substring 00 by 01



Moore Machine



Mealy Machine

Moore Machine to Mealy Machine

Input: Moore Machine
output: Mealy Machine

Step 1:- Take a blank Mealy Machine transition table format.

Step 2:- copy all the moore Machine transition states into this table format

Step 3:-

Check the present states and their corresponding outputs in the moore machine state table; if for a state q_i output is m, copy it into the output columns of the Mealy Machine state table wherever q_i appears in the next state

Eg:

Present State	Next state a=0	Next state a=1	output
→ a	d	b	1
b	a	d	0
c	c	c	0
d	b	a	1

Step 1 & 2

Now we apply algorithm to convert to Mealy M/C

Present state	Next state a=0		Next state a=1	
	State	output	State	output
→ a	d		b	
b	a		d	
c	c		c	
d	b		a	

Steps:

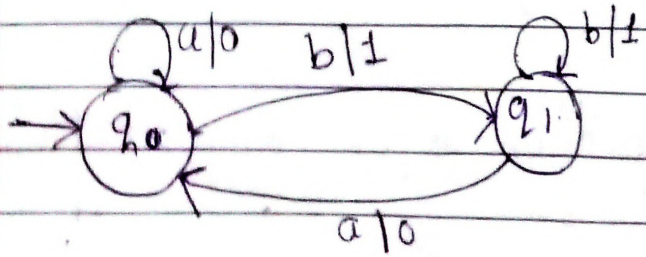
Present state	Next state a=0		a=1	
	State	o/p	State	o/p
→ a	d	1	b	0
b	a	1	d	1
c	c	0	c	0
d	b	0	a	1

Eg: Convert Moore Machine into its equivalent Mealy M/C

Q	a	b	output
→ q_0	q_0	q_1	0
q_1	q_0	q_1	1

Mealy M/C

Q	a	o/p	b	o/p
→ q_0	q_0	0	q_1	1
q_1	q_0	0	q_1	1



Eg:

Melay Machine to Moore Machine

Input: Melay Machine
Output: Moore Machine
Step1:

Calculate the number of different outputs for each state (Qi) that are available in the state table of the Melay machine.

Step2:

If all the outputs of Qi are same copy state Qi. If it has 'n' distinct outputs break Qi into n states as Qin where n = 0, 1, 2 ....

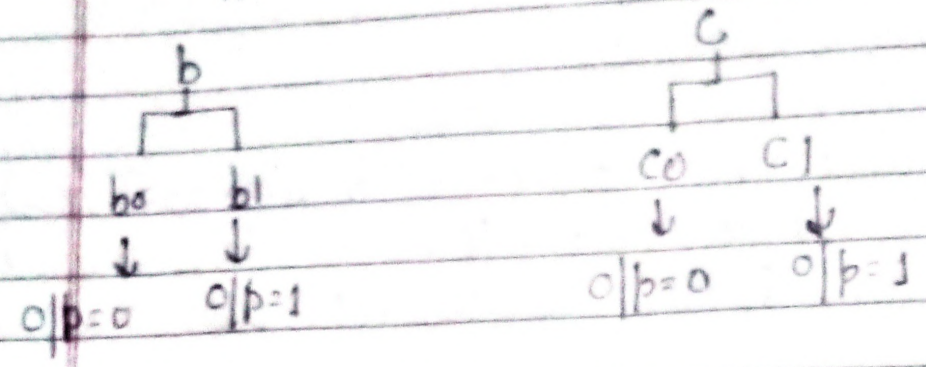
Step3:- If the output of the initial state is 1, insert a new initial state at the beginning which gives 0 output.

Eg:- Let us consider the following Mealy Machine.

Present State	Next state a=0		Next state a=1	
	State	o/p	State	o/p
→a	d	0	b	1
b	a	1	d	0
c	c	1	c	0
d	b	0	a	1

⇒ a, d both states having same output on different input, so no need to break these states

but states 'b' & 'c' produce different outputs on different inputs so we divide.



State	a=0	a=1	output
→ a	d, 0	b1, 1	
b0	a, 1	d, 0	
b1	a, 1	d, 0	
c0	c1, 1	c0, 0	
c1	c1, 1	c0, 0	
d	b0, 0	a, 1	

Final Mealy table

State	a=0	a=1	o/p
→ a	d	b1	1
b0	a	d	0
b1	a	d	1
c0	c1	c0	0
c1	c1	c0	1
d	b0	a	0

Page No.
Date : / /

Eg: convert the given Melay M/c to its equivalent Moore machine.

States	a	o/p	b	o/p
→q0	q3	0	q1	1
q1	q0	1	q3	0
q2	q2	1	q2	0
q3	q1	0	q0	1

i/p
from q0 → a/b → output 1 { gives same o/p }

but on state q1, q2, gives different o/p
so we split q1 state into two state
similarly for q2

q1
q10 q11
↓ ↓
output=0 output=1

q2
q20 q21
o/p=0 o/p=1

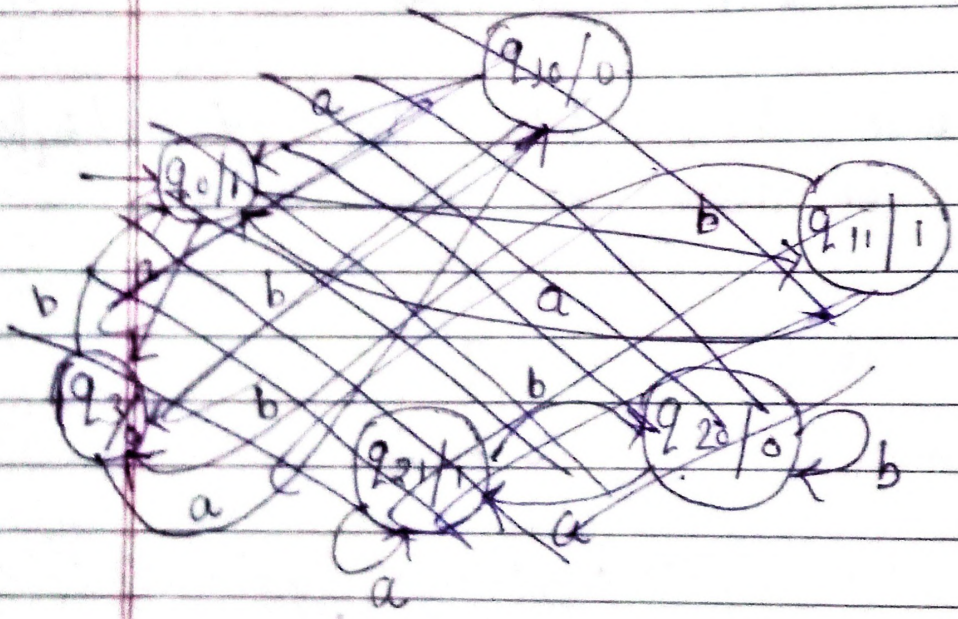
q3 give same output on different i/p symbol
so no need to split q3.

State	a	b	output
→q0	q3, 0	q11, 1	
→q10	q0, 1	q3, 0	
q11	q0, 1	q3, 0	
q20	q21, 1	q20, 0	
q21	q21, 1	q20, 0	
q3	q10, 0	q0, 1	

Page No.
Date

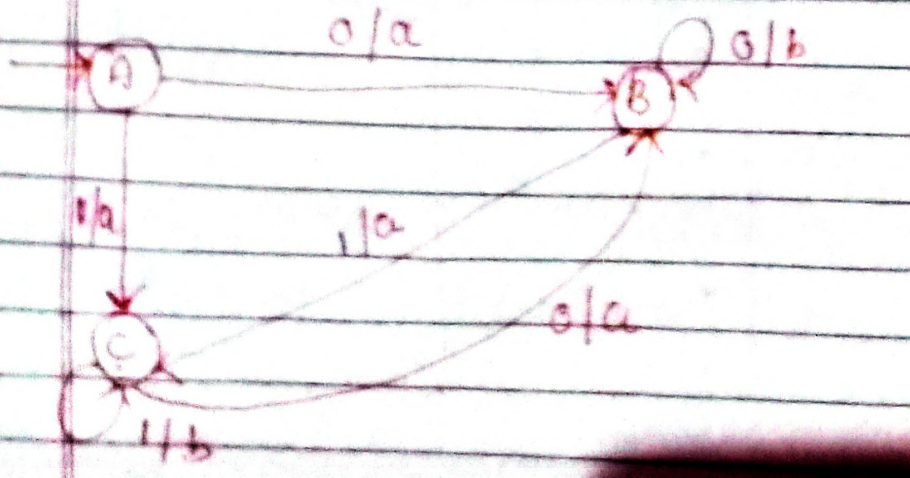
Final Mare table

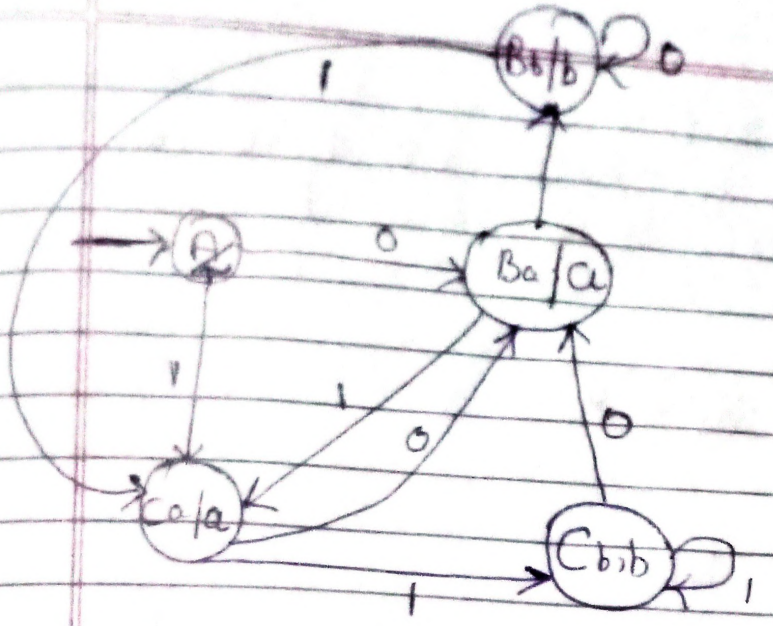
State	a	b	output
→ q_0	q_3	q_{11}	1
q_{10}	q_0	q_3	0
q_{11}	q_0	q_3	1
q_{20}	q_{21}	q_{20}	0
q_{21}	q_{21}	q_{20}	1
q_3	q_{10}	q_0	0



Sol convert the following Mealy Machine to its equivalent Moore machine.

$\Sigma = \{0, 1\}$ $\Delta = \{a, b\}$ — output



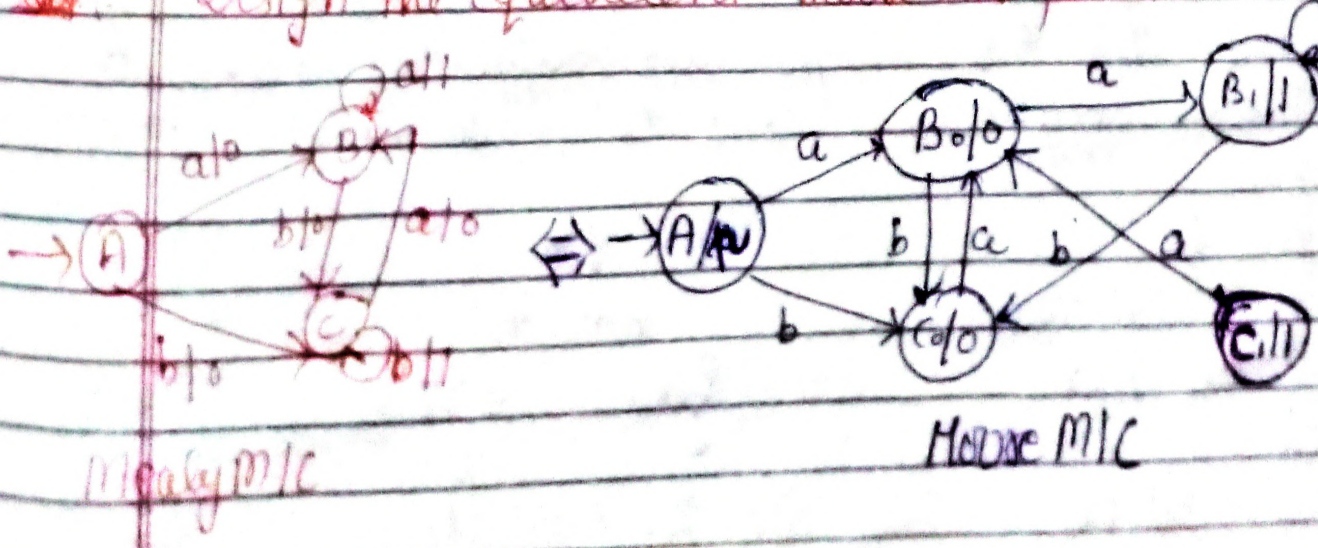


* When we convert Mealy M/C into Moore M/C then no of states increased

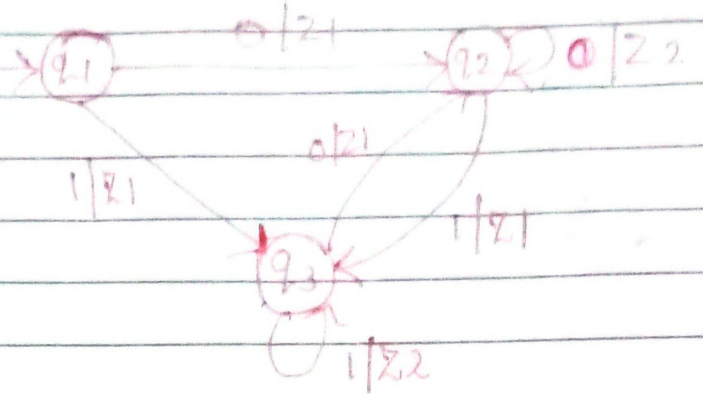
If Mealy m/c having x states & y inputs. then moore m/c having no of states at max

$(x \times y)$

Ques: Design the equivalent Moore m/c for it



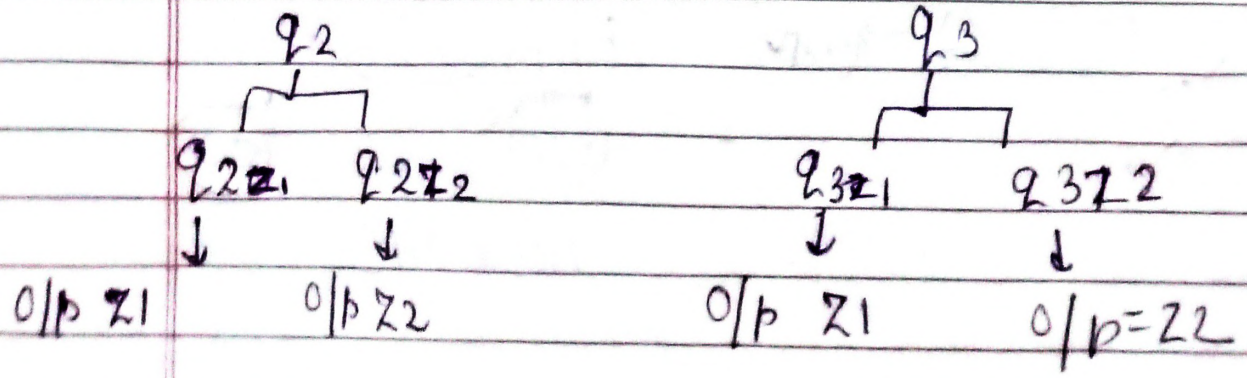
Que: Describ Mealy and Moore FIC with Example. convert the given mealy machine as show in fig. into moore machine.



State	0	o/p	1	o/p
→q1	q2	z1	q3	z1
q2	q2	z2	q3	z1
q3	q2	z1	q3	z2

State q1 on input 0 & 1 having the same output z1, so no need to break the states.

But q2 & q3 produce different output on different inputs. so we divide.

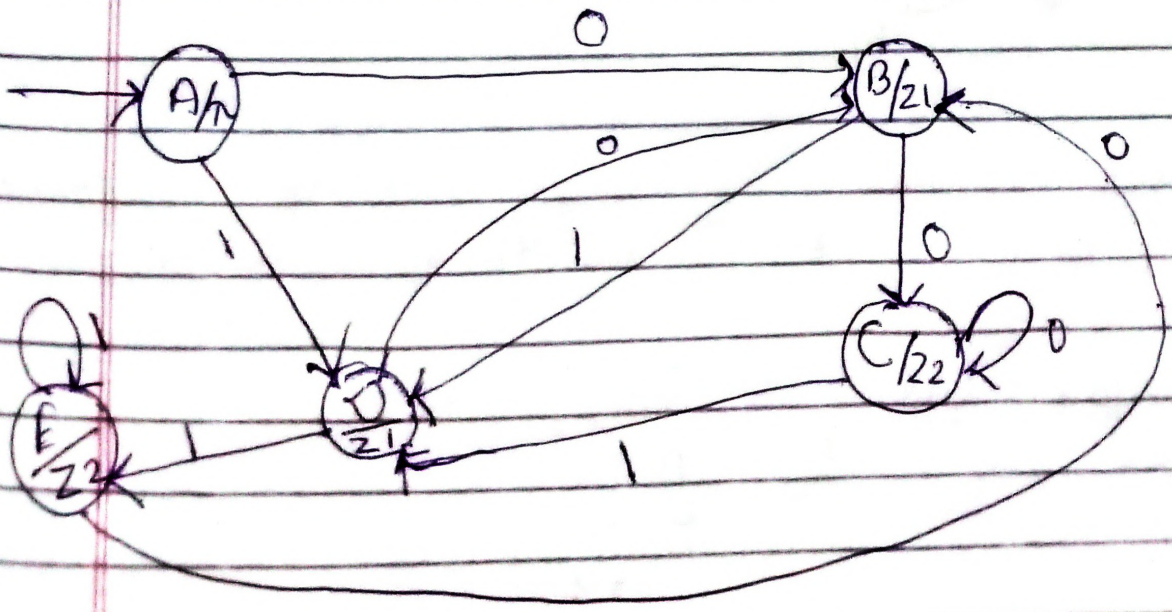


			output
→ q1	q20, Z1	q30, Z1	
q2Z1	q21, Z2	q30, Z1	Z1
q2Z2	q21, Z2	q30, Z1	Z2
q30	q20, Z1	q31, Z2	Z1
q3Z1	q20, Z1	q31, Z2	Z2

⇒

State	0	1	output
→ q1	q2Z1, Z1	q3Z1, Z1	
q2Z1	q2Z2, Z2	q3Z1, Z1	
q2Z2	q2Z2, Z2	q3Z1, Z1	
q3Z1	q2Z1, Z1	q3Z2, Z2	
q3Z2	q2Z1, Z1	q3Z2, Z2	

State	0	1	output
→ q1 (A)	q2Z1 (B)	q3Z1 (D)	
q2Z1 (B)	q2Z2 (C)	q3Z1 (D)	Z1
q2Z2 (C)	q2Z2 (C)	q3Z1 (D)	Z2
q3Z1 (D)	q2Z1 (B)	q3Z2 (E)	Z1
q3Z2 (E)	q2Z1 (B)	q3Z2 (E)	Z2

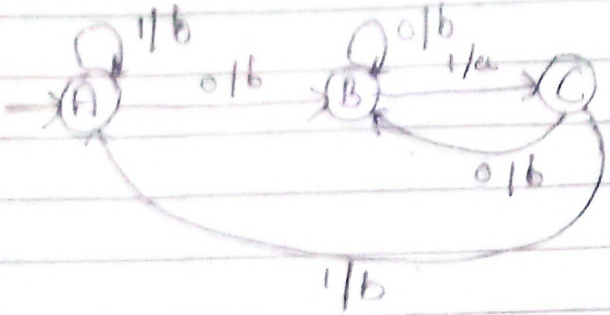


moore machi

* Importance conversion from moore machine to mealy machine conversion

Construct a mealy machine that prints 'a' whenever the sequence '01' is encountered in any input binary string.

$\Sigma = \{0,1\}$ $\Delta = \{a,b\}$



i/p 0110 1000
o/p b a b b b b b b

Eg Construct a mealy machine accepting the language consisting of strings from Σ^* where $\Sigma = \{a,b\}$ and the strings should end with either aa or bb.

aa - 1
bb - 1

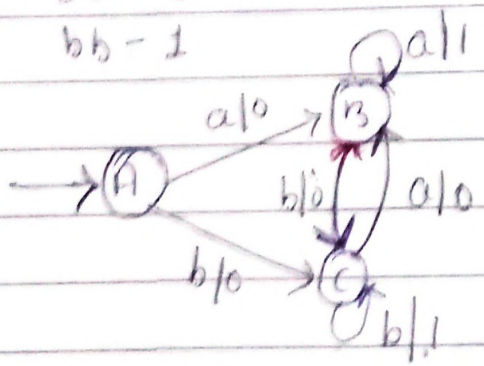


abb ba
baa

Eg Construct a mealy machine that gives 2's complement of any binary input (assume that the least carry bit is neglected)

2's complement = 1's complement + 1

Eg: 10100
1's 01011
+1
2's 01100

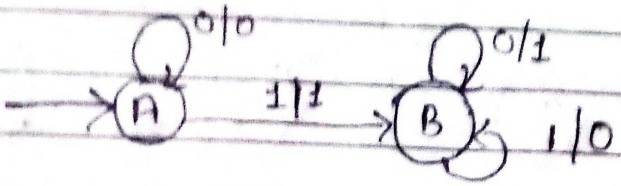
Eg: 11100
00011
+1
00100 2's compliment

Eg: 1111
0000 2's
+1
0001 2's

1111
↓
0001 2's complement

MSB LSB

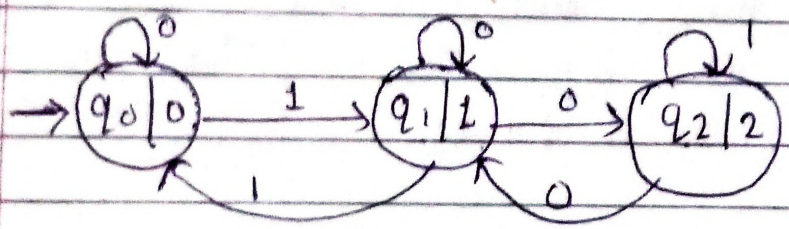
Eg: 10100
↓↓↓↓
2's 01100
(complement)
↓
after getting 1's all bits after 1 will be complement.



1 0 1 0 0
↑ ↑ ↑ ↑ ↑
B B B<A ←A
0 1 1 0 0

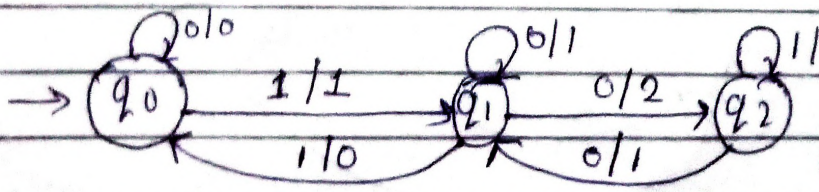
Que Design a moore machine to determine residue modulo 3 for each binary string treated as binary integer

5÷3=2 ↓ 010
3÷3=0 ↓ 000
4÷3=1 ↓ 001



Moore

0 - 000
1 - 001
2 - 010
3 - 011
4 - 100
5 - 101
6 - 110
7 - 111



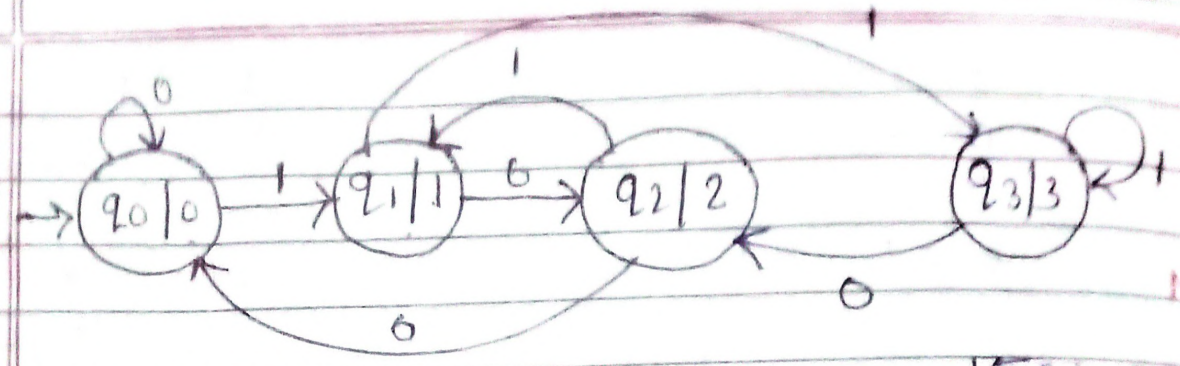
Mealy

Que Design a Moore & Mealy machine which calculate residue mode 4 for each binary string treated as binary integer

4÷4 = 0 → Binary 0000
5÷4 = 1 → 0001
6÷4 = 2 = 0010

7÷4 = 3 → 0011
8÷4 = 0 → 0000

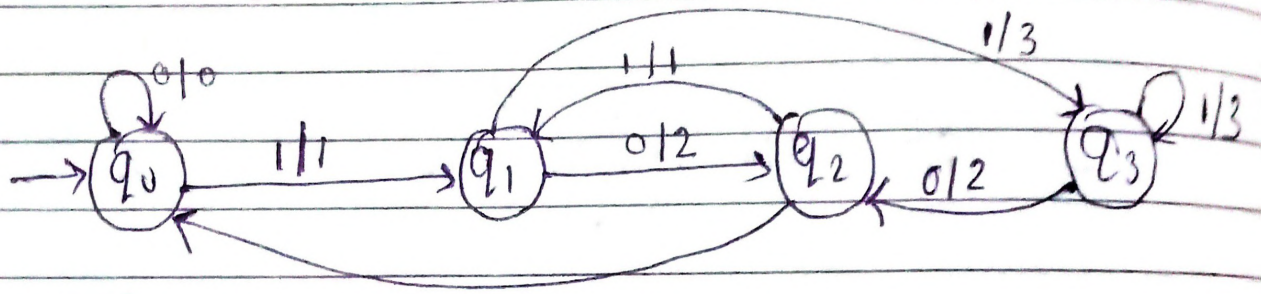
0 - 0000
1 - 0001
2 - 0010
3 - 0011
4 - 0100
5 - 0101
6 - 0110
7 - 0111
8 - 1000
9 - 1001
10 - 1010



15 → 1010

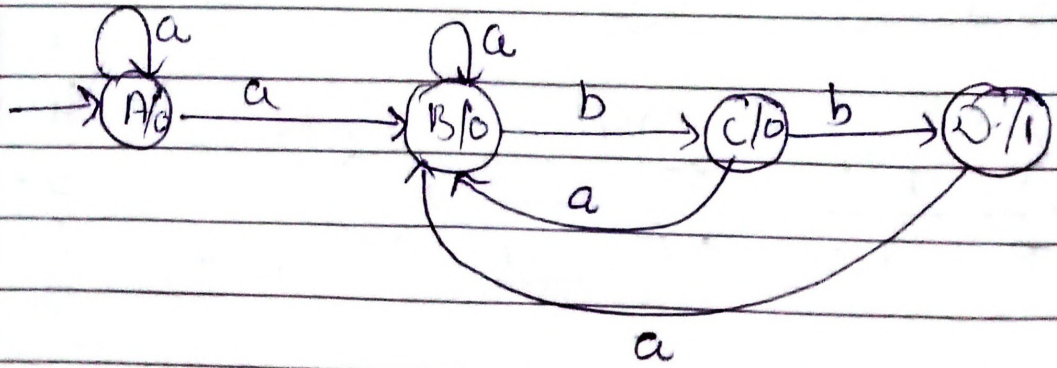
15 ÷ 4 = 3

Binary of 3: 0011



Mealy m/c

Q) Construct a moore machine that counts the occurences of the sequences 'abb' in any input strings over {a, b} ∑ = {a, b}



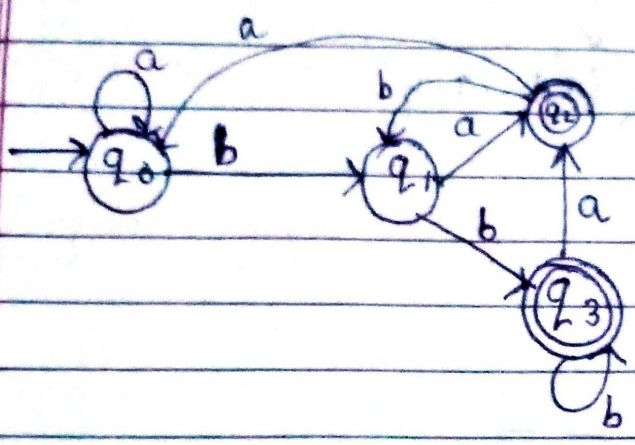
③ Design a minimal DFA that accepts all strings over the alphabet $\Sigma = \{a, b\}$ such that the accepted string 2nd from right end is always b.

$\Sigma = \{a, b\}$ — 2nd from right end.

L = _ _ _ _ _ b _

ba bb

two choice



In these case if position is right side is 2 then no of state, 4

$Q^n = 2^2 = \boxed{4}$

* trick to solve for large position.

if a's position is 3rd from right side then no of state $= 2^3 = \boxed{8}$

dfa machine of this question is page no-37

	b	a
→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_6	q_7
q_4	q_0	q_1
q_5	q_2	q_3
q_6	q_4	q_5
q_7	q_6	q_7

now, how to make final state

Lower half state will be final. means no of state 8, then $\frac{8}{2}$ = 4 - final state

NOTE

Which symbol you have to accept, put in 2nd column means we want check 'a' position