UNIT-1

INTRODUCTION TO FINITE AUTOMATA

FORMAL LANGUAGE AUTOMATA THEORY

UNIT - I

Automata = an abstract computing devices.

→ Automata theory is the study of computation.
↓
Processing
(processes input to output)

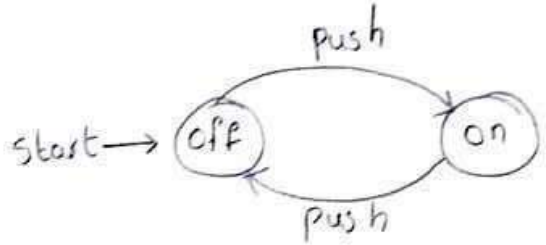
→ Automata theory deals with logic of computation with respect to simple machines, referred to as automata.

→ Through automata, we understand how machines compute functions and solve problems.

→ Alan Turing (1912-1954) is the father of modern computer, had developed a machine called Turing m/c.

→ Turing m/c is the basic model that describes the entire functionality of all the machines (i.e., set of i/p's, set of outputs and the processor).

Example. Finite Automata model for the switch is



→ For all finite automata states are represented by circles. In this example we have two states 'on' and 'off'. Arcs blw states are labeled by 'inputs', which represent external influences on the system.

Structural Representation:

→ There are two important notations that play an important role in the study of automata and their applications.

① Grammars

② Regular Expressions

→ Grammars are useful models when designing software that processes data with recursive structure.

Ex: A grammatical rule like $E \Longrightarrow E+E$ states that an expression is formed by taking any two expressions and connecting them by plus sign.

→ Regular Expressions: R.E denote the structure of data, especially text strings.

UNIX style R.E is '[A-Z][a-z]* []][A-Z][A-Z]' represent capitalized words followed by a space and two capital letters. This expression represents the patterns in text that could be a city and state Hyderabad TE.

Automata and complexity:

→ Automata is the study of limits of computation.

There are two important issues:

① What can a computer do at all? This study is called "decidability".

② what can a computer do efficiently? This study is called "intractability". An the problems tha can be solved by computer using no more time is called 'tractable".

The Central concepts of Automata Theory :

Three things used to design a machine :

① Alphabet (set of symbols)

② Strings (list of symbols from an alphabet)

③ Language (a set of strings from the same alphabet).

Alphabet :

→ An alphabet is a finite non-empty set of symbols.

→ Representation : Σ (sigma)

common alphabets include

- $\Sigma = \{0, 1\}$, the binary alphabet.
- $\Sigma = \{a, b, \ldots z\}$, set of all lower case letters.
- $\Sigma = \{0, 1, 2, \ldots 9\}$, set of all ASCII characters or digits.
- $\Sigma = \{a\text{-}z, A\text{-}Z, 0\text{-}9\}$ Alphanumeric

Strings :

→ A string or word is a finite collection of symbols selected from the alphabets (Σ).

Ex: 01101 is a string from binary alphabet $\Sigma = \{0, 1\}$.

→ The Empty string is the string with zero occurences of symbols. Represented by ϵ ('Epsilon').

Length of string :

→ The "length" of the string is denoted by $|w|$ and it is the number of positions for the symbol in the string.

Ex: $w = 01101$ has length $= 5$ i.e, $|w| = 5$

Powers of an alphabet :

→ If Σ is an alphabet, we can express the set of all strings of certain length from that alphabet by using an exponential notation. "Σ^k" represent set of string of length K.

Ex $\Sigma^0 = \epsilon$

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

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

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

$\Sigma^3 = \{aaa, abc, aab, bbb, aac, \ldots\}$

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

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$ is set of non empty strings

$\Sigma^* = \Sigma^+ \cup \{\epsilon\}$.

Concatenation of strings:

→ Let x & y be two strings. Then xy denotes the concatenation of x & y i.e. the string formed by making a copy of x and following it by a copy of y.

Ex: $x = ab$ & $y = cd$ then $xy = abcd$.

Reverse of string:

→ Reverse of the string can be achieved by simply interchanging over last symbols.

Ex $w = abc$ the $w^R = cba$

Languages:

→ A set of strings which are chosen from some Σ^*, where Σ is a particular alphabet is called a language.

Ex - The language of all strings consisting of n 0's followed by n 1's for some $n \geq 0$. $\{\epsilon, 01, 0011, 000111, \ldots\}$

- The set of strings of 0's and 1's with equal number of each: $\{\epsilon, 01, 10, 0101, 0011, 1001, \ldots\}$
- Set of binary numbers whose value is prime.

$\{10, 11, 101, 111, 1011, \ldots\}$

- Σ^* is a language for any alphabet Σ.
- $\emptyset$, the empty language, over any alphabet.

Problems:

→ Problems is the question of deciding whether a given string is a member of some particular language

→ If Σ is an alphabet, and L is a language over Σ, then the problem L is:

- Given a string w in Σ^*, decide whether or not w is in L.

Finite Automata.

① DFA (Deterministic Finite Automata)

② NFA (Non-deterministic Finite automata)

Non-Deterministic Finite Automata :

→ F.A are mainly used for Pattern recognition.
→ It takes string of symbol as i/p and changes its state accordingly.
→ when desired symbol is found, then transition occurs.
→ A the time of transition, the automata can either move to the next state or stay in same state.
→ Finite automata has two states, Accept state or Reject state.

Formal Description of FA

A FA is a collection of 5- tuple $(Q, \Sigma, q_0, F, \delta)$ where:

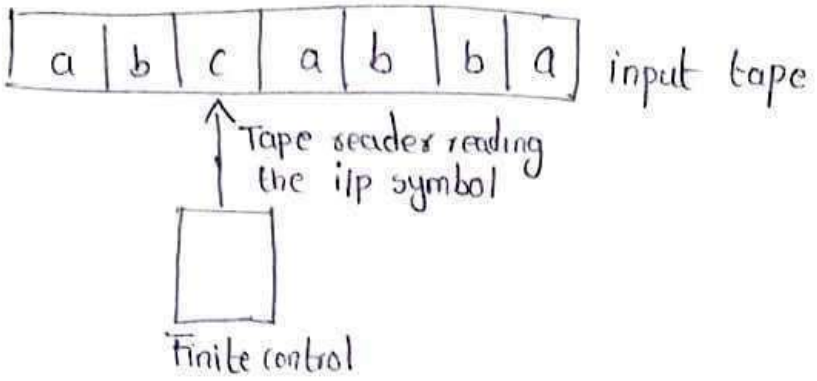
1. Q: finite set of states
2. Σ: Finite set of the input symbol.
3. q_0: Initial state
4. F: Final state.
5. δ: Transition function. (movement of string from one state to another state)

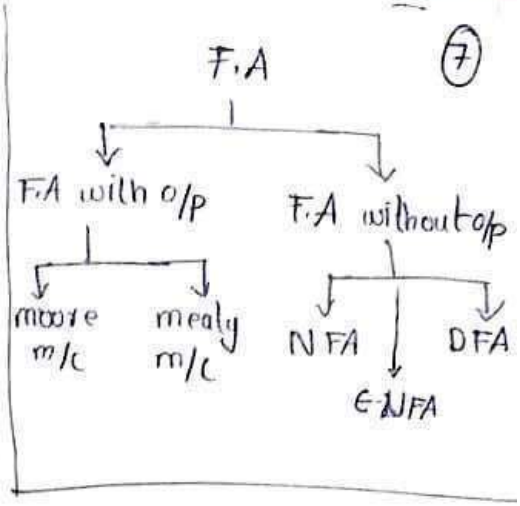
Finite Automata model

F.A can be represented by input tape and finite control.

Input tape: It is a linear tape having some number of cells. Each input symbol is placed in each cell.

Finite control: The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one by one from left to right, and at a time only one input symbol is read.





Deterministic Finite Automata (DFA)

→ DFA is in a single state after reading any sequence of inputs.

→ The term 'deterministic' refers to fact that on each input there is one and only one state to which the automaton can transition from its current state.

→ DFA does not accept the null move i.e., DFA cannot change state without any input characters.

Definition of a DFA:

→ A DFA is a collection of 5-tuples

$$F.A = \{Q, \Sigma, q_0, F, \delta\}$$

1. Q : Finite set of states.
2. Σ : Finite set of the input symbol.
3. q_0 : initial state.
4. F : final state.
5. δ : Transition function that takes as arguments a state and an input symbol and returns a state.

$$\delta : Q \times \Sigma \longrightarrow Q \quad | \quad \delta(q, a) \longrightarrow p$$

How a DFA Processes strings:

→ First understand how DFA decides whether or not to "accept" a sequence of input symbols.

→ Suppose $a_1 a_2 a_3 \ldots a_n$ is a sequence of input symbols, we start with start state q_0, transition function δ is

$$\delta(q_0, a_1) = q_1$$

Then we process next input symbol a_2 by evaluating $\delta(q_1, a_2)$ which enters state q_2.

→ we continue in this manner finding states $q_3, q_4 \ldots q_n$ such that $\delta(q_{i-1}, a_i) = q_i$ for each i.

If q_n is a member of F, then the input $a_1 a_2 \ldots a_n$ is accepted, if not it is "rejected".

example: specify a DFA that accepts all and only the strings of 0's and 1's that have the sequence 01 somewhere in the string.

Language L is as follows

$\{ w \mid w$ is of the form $x01y$ for some strings x and y consisting of 0's and 1's only $\}$

(or)

$\{ x01y \mid x$ and y are any strings of 0's and 1's $\}$

→ strings in this language include 01, 0001, 11010, ... and strings not in this language include ϵ, 0, 111000, ...

Simpler Notations for DFA's:

There are two preferred notations for describing automata:

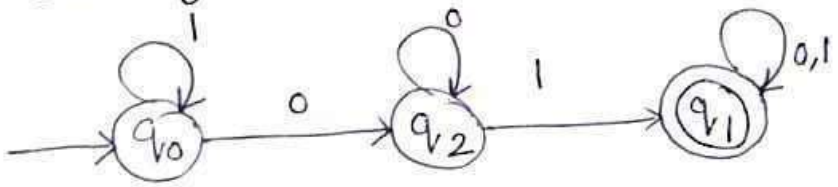
1. Transition diagram (which is a graph)
2. Transition Table (which is a tabular listing of δ function, which by implication tells us the set of states and the input alphabet)

Transition diagram:

→ A transition diagram for a DFA $A=(Q, \Sigma, \delta, q_0, F)$ is defined as

- For each state in Q there is a node.
- For each state q in Q and each input symbol a in Σ let $\delta(q,a)=P$, then transition diagram has an arc from node q to node p labeled a.
- There is an arrow into the start state q_0, labeled 'start'
- Nodes corresponding to accepting states are marked by a double circle. states not in F have a single circle.

Ex. Transition diagram for DFA accepting all strings with a substring 01.



Transition Tables:

→ Transition table is a tabular representation of a function like δ that takes two arguments and returns a value.

→ Rows correspond to states and columns correspond to the inputs.

Ex: Transition table for DFA accepting all strings with a substring 01.

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

→ Extended transition function ($\hat{\delta}$) is a function that takes a state q and a string w and returns a state p - the state the automaton reaches when starting in state q and process the input w.

Ex: suppose w is a string of the form xa, a is the last symbol of w. If $w = 1101$ is broken in to $x = 110$ & $a = 1$

then $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$

$\hat{\delta}(q_0, \epsilon) = q_0$ i/p = 1101

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

$\hat{\delta}(q_0, 11) = \delta(\hat{\delta}(q_0, 1), 1) = \delta(q_0, 1) = q_0$

$\hat{\delta}(q_0, 110) = \delta(\hat{\delta}(q_0, 11), 0) = \delta(q_0, 0) = q_2$

$\hat{\delta}(q_0, 1101) = \delta(\hat{\delta}(q_0, 110), 1) = \delta(q_2, 1) = q_1$

The Language of DFA:

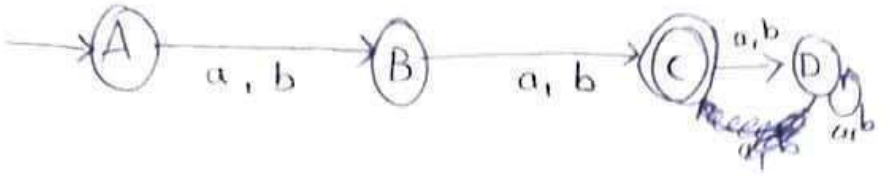
→ The language of DFA $A = (Q, \Sigma, \delta, q_0, F)$ is denoted by $L(A)$, is defined by

$$L(A) = \{ w \mid \hat{\delta}(q_0, w) \text{ is in } F \}$$

i.e., the language of A is the set of strings w that take the start state q_0 to one of the accepting states.

Example 1: construct a DFA over $\Sigma = \{a, b\}$ where the length is equal to 2.

Solution: The language will be $L = \{aa, ab, ba, bb\}$

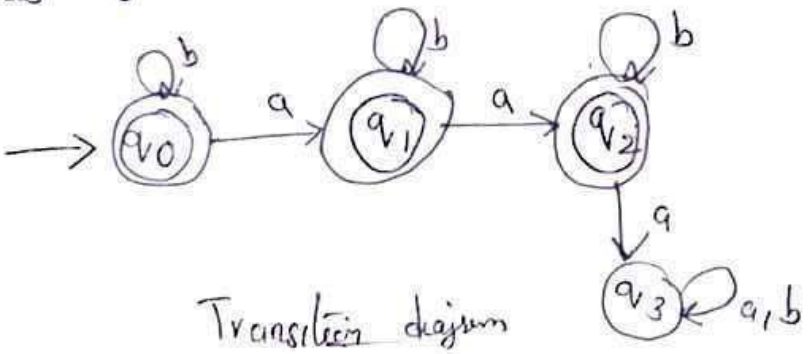


- First write down lang.
- Try to accept all the strings
- Maintain DFA restrictions

Example 2.

Design DFA which accepts all the strings not having more than two a's over $\Sigma = \{a, b\}$.

Solution: In this DFA maximum two a's are accepted. If we try to accept third a then it should not lead us to final state.



Transition diagram

	a	b
→ q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_3	q_3

Example : Design DFA for the following languages shown below $\Sigma = \{a, b\}$

(a) $L = \{w \mid w$ does not contains the substring $ab\}$

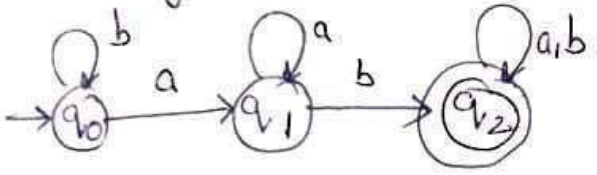
(b) $L = \{w \mid w$ contains neither the substring ab nor $ba\}$

(c) L $\{w \mid w$ is any string that does not contain exactly two a's$\}$

(d) $L = \{w \mid w$ is any string except a & b$\}$

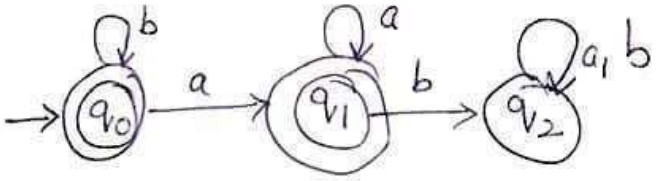
Solution:

(a) For designing the DFA specified by language L we will first design DFA that contain substring ab.

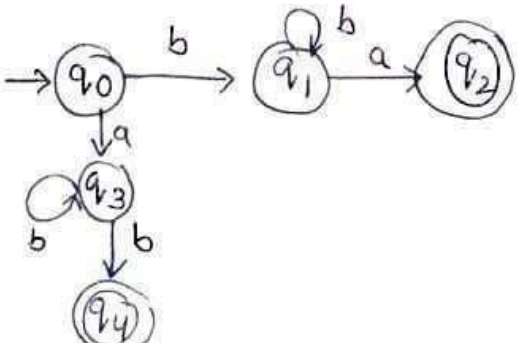


Now, For DFA that accepts a language L, which do not contain substring ab, we will simply change the accept state (ie., final state to non final state and non final states will be made final states.

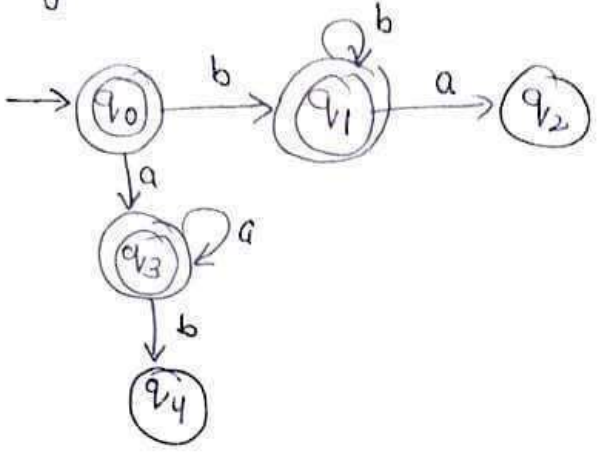
The DFA will then be as follows.



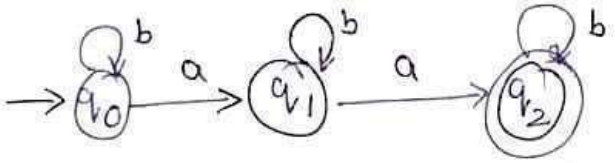
(b) The DFA that contains ab or ba is



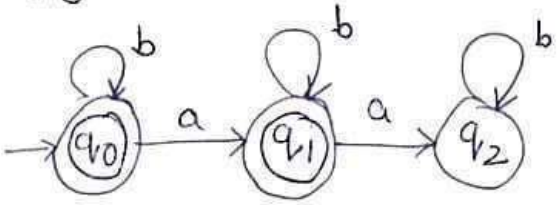
Now if we change non-final state to final and final state to non final state then the DFA is as follows. This DFA accepts strings which do not contain the substring ab or ba.



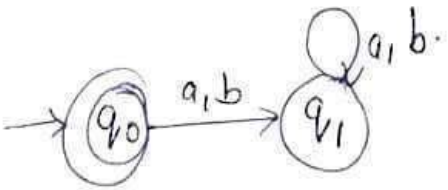
(c) The DFA that contains a language having two a's exactly:



Now we will exchange the final and non-final states and then the DFA which will be obtained will represent a language that doesn't contain exactly two a's.

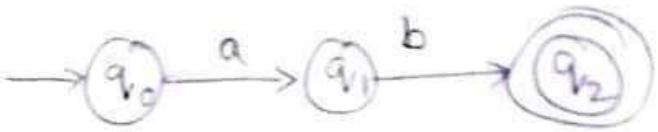


(d) The DFA that does not contain a, b is



Ex: Obtain a DFA to accept strings of a's & b's starting with the string ab

Solution: It is clear that the string should start with 'ab' and so the minimum string ab, we need three states and the machine can be written as

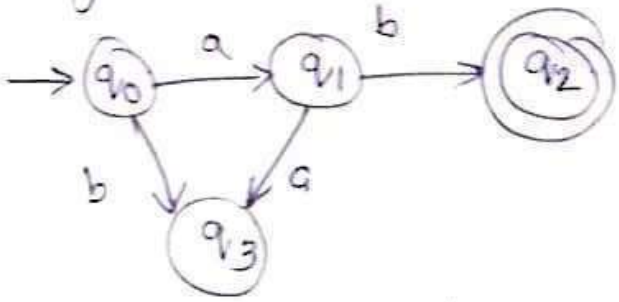


q_0 - starting / initial state

q_2 - Final / accepting state

→ since the starting char shouldn't be 'b' the input symbol 'b' is pointed to the rejected / dead state q_3 from q_0.

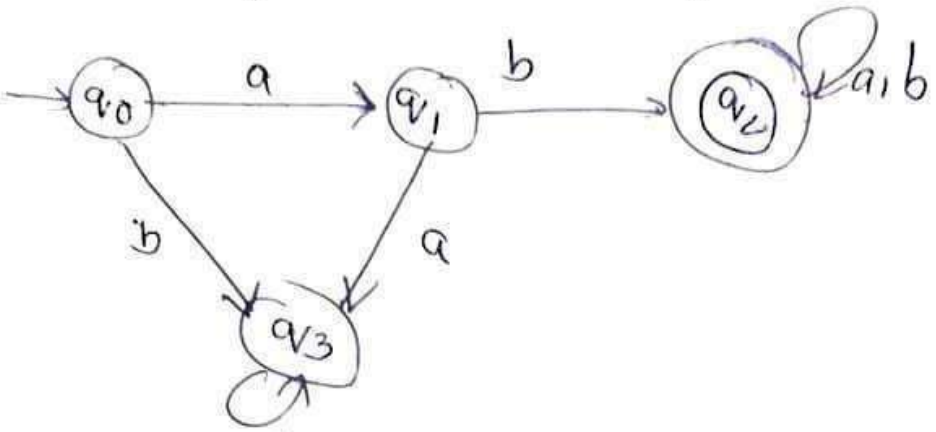
→ since the second char shouldn't be 'a' the I/p symbol 'a' is pointed to dead state q_3 from q_1



→ whenever the string is not starting with ab, the machine will be in state q_3 which is dead state

→ After the string ab, the string containing any combination of a's & b's can be accepted and so remain in the state q_2 only

→ The complete m/c to accept the string of a's & b's starting with the string ab is as follows



The language accepted by DFA can be represented as

$L = \{ ab(a+b)^n / n \geq 0 \}$ (or)

$L = \{ ab(a+b)^* \}$

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

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

$q_0 - \{q_0\}$

$F - \{q_2\}$

Transition table :

states	a	b
→ q_0	q_1	q_3
q_1	q_3	q_2
(*) q_2	q_2	q_2
q_3	q_3	q_3

(2) Ex: Draw a DFA to accept string of 0's and 1's ending with the string '011'. (16)

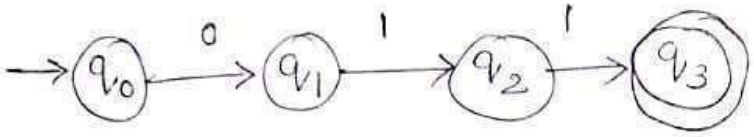
Solution: Given $\Sigma = \{0, 1\}$

L is All the strings ending with string 011.

So

$L = \{011, 000\ldots011, 111\ldots011, 0101100\ldots011, \ldots\ldots\}$

→ The basic string is 011, so draw FA for this using $Q = \{q_0, q_1, q_2, q_3\}$



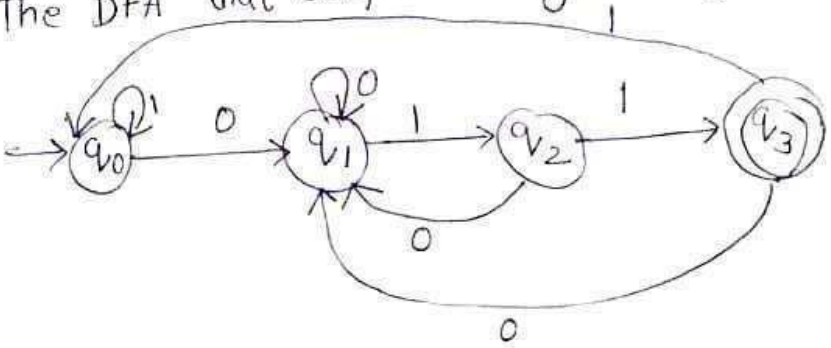
→ Since the starting symbol can be anything the input symbol '1' on q_0 is self looped.

→ Since the ending must be 011 the input '0' on q_1 is self-looped.

→ The input '0' on q_2 is pointed to q_1 so that it can end as 011.

→ '0' and '1' input on q_3 is pointed to q_1 and q_0 respectively.

→ The DFA that accepts strings ending with 011 is as follows



Transition table

states	symbols 0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_3
(q_3)	q_1	q_0

Non - Deterministic Finite Automata (NFA):

→ The transition from a state can be to multiple next states for each input symbol.

→ NFA Permits empty string transitions.

A NFA is represented essentially like a DFA

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

where ① Q is a finite set of states.

② Σ is a finite set of input symbols

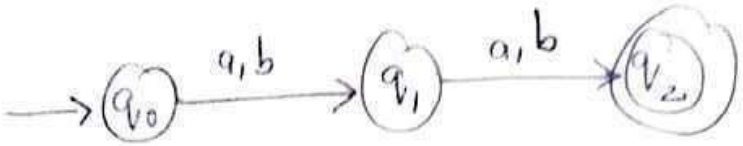
③ q_0, a member of Q, is the start state.

④ F, a subset of Q, is the set of final or accepting states

⑤ δ, the transition function that takes a state in Q and an input symbol in Σ as arguments and returns a subset of Q.

Example: construct NFA over $\Sigma = \{a, b\}$ where the length is equal to 2.

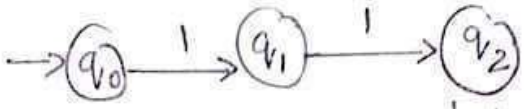
Solution: The language will be $L = \{aa, ab, ba, bb\}$



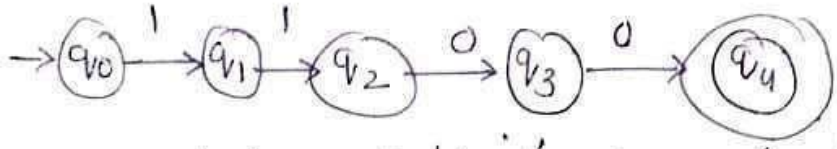
Example:

Design an NFA with $\Sigma = \{0, 1\}$ in which double '1' is followed by double 'zero'.

Soln: The FA with double '1' is as follows

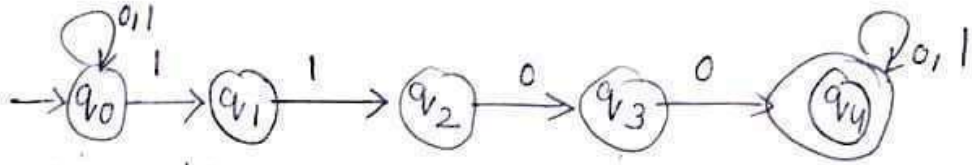


→ It should be immediately followed by '0'



→ Now before double '1' there can be any string of 0's & 1's, similarly after double '0' there can be any string of 0's & 1's.

→ Hence NFA becomes

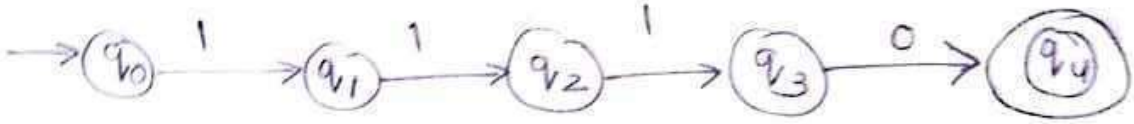


Transition table is as follows:

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

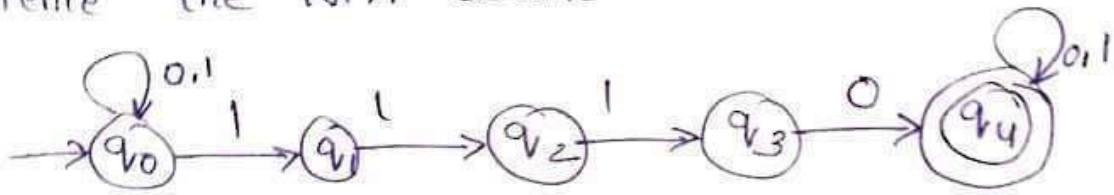
Example(1) Design a NFA in which all the string contain a substring 1110.

Solution The language consists of all the strings containing 1110. The partial transition diagram can be:



→ Now as 1110 could be the substring Hence we will add the inputs 0's and 1's so, that the substring 1110 of the language can be maintained.

→ Hence the NFA becomes:



Transition table:

States / input	0	1
→ q_0	q_0	$\{q_0, q_1\}$
q_1	ϕ	q_2
q_2	ϕ	q_3
q_3	q_4	ϕ
* q_4	q_4	q_4

Consider the string 111010

$\delta(q_0, 111010) = \delta(q_0, 11010)$

$= \delta(q_0, 1010)$

$= \delta(q_2, 1010)$

$= \delta(q_3, 010)$

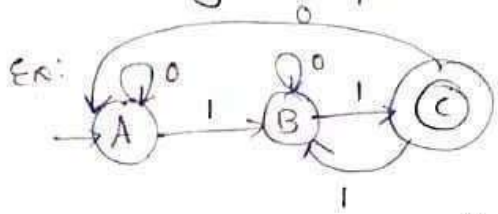
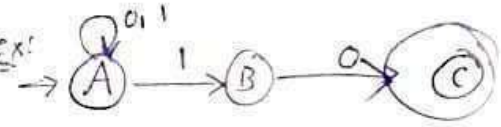
$= \delta(q_4, 10)$

$= \delta(q_4, 0)$

$= \delta(q_4, \epsilon)$

→ Hence q_4 is the acceptance state, the string is acceptable.

Differences b/w DFA & NFA

DFA	NFA
1) Deterministic Finite Automata	1) Non-Deterministic ~~finite Automata~~
2) DFA is complete system	2) It may/maynot be complete.
3) Every DFA is NFA	3) Every NFA may/maynot be DFA. But NFA can be converted to DFA
4) For every input string there exists only one path. Ex: 	4) For every input string there exist 0 or 1 more no of paths Ex: 
5) DFA is as powerful as NFA	5) NFA is as powerful as DFA
6) DFA is more efficient.	6) NFA is comparatively less efficient.
7) $\delta : Q \times \Sigma \rightarrow Q$	7) $\delta : Q \times \Sigma \rightarrow 2^Q$
8) Here we specify what is required to us and what not is required to us	8) Here we specify what is required only.
9) Design complexity is more	9) comparatively less.
10) Dead state may be required.	10) Dead state is not required.
11) DFA can be understood as one machine	11) NFA can be understood as multiple little machines computing at same time.

Applications of Finite Automata :

→ The applications of Finite Automata are :-

1) A finite automata is highly useful to design lexical Analysers.
2) It is useful to design text editors.
3) It is highly useful to design spell checkers.
4) It is useful to design sequential circuit design (transducers).
5) Software for scanning large bodies of text (Eg: web pages) for pattern finding.
6) Software for verifying systems of all types, that have a finite number of states.

(Eg: Stock market transaction, communication / network protocol)

NFA for Text search :

→ NFA are mostly used for performing text search from group of words usually called as Keywords. To determine the occurence of any keyword with in a text.

→ For this reason a NFA is designed. This automata is designed in such a way that it send signals upon seeing a keyword in accepting state.

→ Here the document compressing text is sent from time to time to NFA.

→ So that it finds occurences of the keyword in the text.

NFA to search text is designed as follows:

1) The FA consist of a start state with a transition to every input symbol.

2) There exist K states for the keywords $m_1, m_2, \ldots m_K$

3) A transition from the start state of q_1 on symbol m_1 will exist where in a transition from q_1 to q_2 exist on symbol m_2 and on forth q_K state represents that it is an accepting state and signifies, it has found all the keywords $m_1, m_2, \ldots m_K$.

NFA with Epsilon Transitions:

ϵ – Empty string / Null string

→ In this type of machines changing of state is possible without reading the strings.

→ Such transitions are labeled with ϵ-transition.

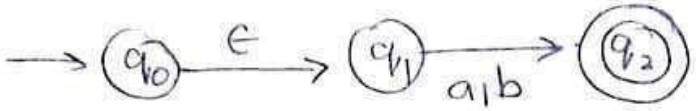
→ The symbol ϵ does not belong to any alphabet.

A ϵ-NFA is a $A = (Q, \Sigma, \delta, q_0, F)$

- Q is set of states.
- Σ is the alphabet of input symbols.
- q_0 is initial state.
- F is the set of final states.
- $\delta : Q \times \Sigma\epsilon \rightarrow P(Q)$ is the transition function.

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

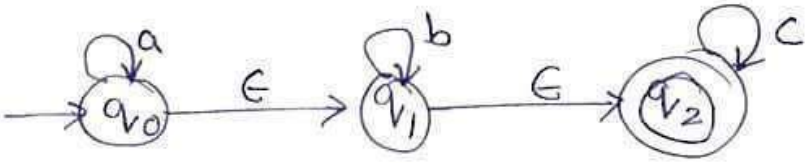
Example: A NFA accepting a language having length exactly 1 over $\Sigma = \{a, b\}$.



Example: Construct NFA with 'ϵ' which accepts a language consisting the strings of any number of a's followed by any number of b's. Followed by any number of c's.

Solution: Here any number of a's or b's or c's means zero or more in number. That means there can be zero or more a's followed by zero or more b's followed by zero or more c's.

Hence NFA with ϵ can be



The transition table is:

state \ input	a	b	c
→ q_0	q_0	ϕ	ϕ
q_1	ϕ	q_1	ϕ
*q_2	ϕ	ϕ	q_2

ϵ-closure$(q_0) = \{q_0, q_1, q_2\}$

ϵ-closure$(q_1) = \{q_1, q_2\}$

ϵ-closure$(q_2) = \{q_2\}$

we can parse the string aabbcc as follows:

$\delta(q_0, aabcc) - \delta(q_0, abcc)$

$- \delta(q_0, bcc)$

$- \delta(q_0, \epsilon bcc)$

$- \delta(q_1, bcc)$

$- \delta(q_1, cc)$

$- \delta(q_1, \epsilon cc)$

$- \delta(q_2, c)$

$- \delta(q_2, c)$

$- \delta(q_2, \epsilon)$

$= q_2$

Conversion of NFA with ∈ to NFA without ∈.

In this method we try to remove all the ∈ transitions from the given NFA. The method will be.

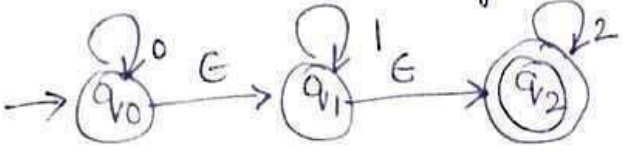
1. Find out all the ∈-transitions from each state from Q. This will be called as ∈-closure{q_i} where $q_i \in Q$.
2. Then δ' transitions can be obtained. The δ' transitions means an ∈-closure on δ moves.
3. Step 2 is repeated for each input symbol and for each state of given NFA.
4. Using the resultant states the transition table for equivalent NFA without ∈ can be built.

Rule for conversion

$$\delta'(q, a) = \in\text{-closure}\left(\hat{\delta}(\hat{\delta}(q, \in), a)\right)$$

where $\hat{\delta}(q, \in) = \in\text{-closure}(q)$

Example: convert the given NFA with ∈ to NFA without ∈.



Solution: we will first obtain ∈-closure of each state i.e. we will find out ∈ reachable states from current state.

Hence $\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$

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

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

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

$= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0),0))$

$= \epsilon\text{-closure}(\delta(q_0,q_1,q_2),0)$

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

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

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

$\delta'(q_1,0) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0,\epsilon),1))$

$= \epsilon\text{-closure}(\delta(q_0,q_1,q_2),1))$

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

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

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

$= \epsilon\text{-closure}(\delta(q_1,q_2),0))$

$= \epsilon\text{-closure}(\delta(q_1,0) \cup \delta(q_2,0))$

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

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

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

$= \epsilon\text{-closure}(\delta(q_1,q_2),1)$

$= \epsilon\text{-closure}(\delta(q_1,1) \cup \delta(q_2,1))$

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

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

$= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2),0)$

$= \epsilon\text{-closure}(\delta(q_2,0))$

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

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

$= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2),1))$

$= \epsilon\text{-closure}(\delta(q_2,1)) = \epsilon\text{-closure}(\phi) = \phi$

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

$= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0),2)) = \epsilon\text{closure}(\delta(q_0,q_1,q_2),2))$

$= \epsilon\text{-closure}(\delta(q_0,2) \cup \delta(q_1,2) \cup \delta(q_2,2))$

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

$\delta'(q_1,2) = \epsilon\text{-closure}(\delta(\delta'(q_1,\epsilon),2)$

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

$= \epsilon\text{-closure}(\delta(q_1,q_2),2) = \epsilon\text{-closure}(\delta(q_1,2) \cup \delta(q_2,2))$

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

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

$= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2),2)$

$= \epsilon\text{-closure}(\delta(q_2,2)) = \epsilon\text{-closure}(q_2) = \{q_2\}$

Finally:

$\delta'(q_0,0) = \{q_0,q_1,q_2\}$

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

$\delta'(q_2,0) = \phi$

$\delta'(q_0,1) = \{q_1,q_2\}$

$\delta'(q_1,1) = \{q_1,q_2\}$

$\delta'(q_2,1) = \phi$

$\delta'(q_0,2) = \{q_2\}$

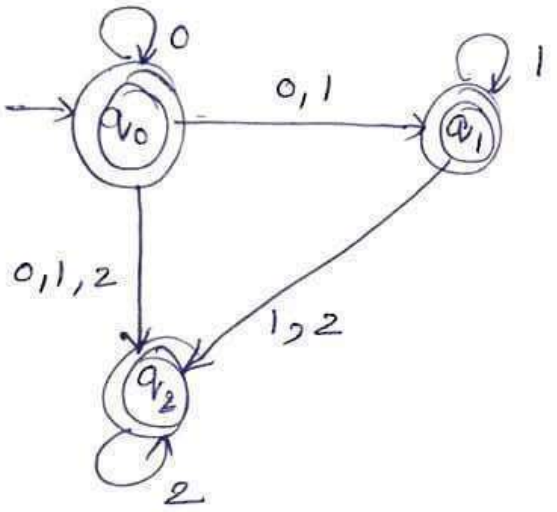
$\delta'(q_1,2) = \{q_2\}$

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

From this Transition table is

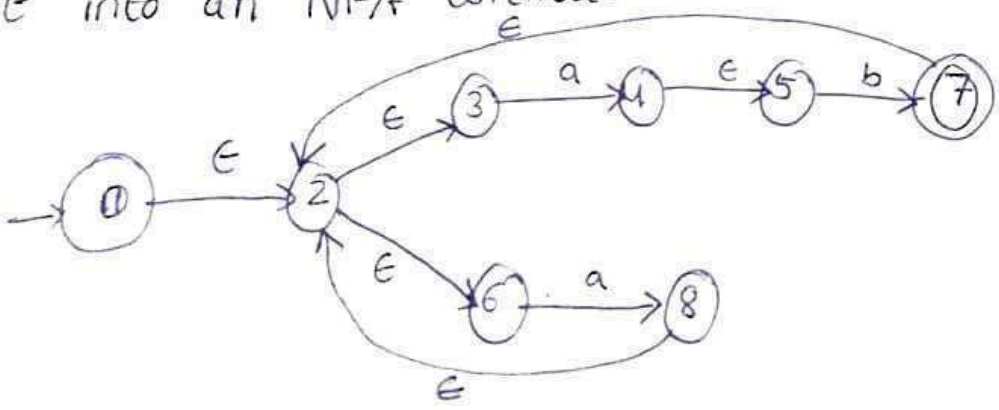
State / input	0	1	2
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

NFA will be



Hence q_0, q_1, q_2 is a final state because ϵ-closures of q_0, q_1, q_2 contains final state.

Example (2) convert the following NFA with ϵ moves convert it into an NFA without ϵ-moves.



Solution : we will first find out ϵ-closure of states

ϵ-closure (1) = {1, 2, 3, 6}

ϵ-closure (2) = {2, 3, 6}

ϵ-closure (3) = {3}

ϵ-closure (4) = {4, 5}

ϵ-closure (5) = {5}

ϵ-closure (6) = {6}

ϵ-closure (7) = {2, 3, 6, 7}

ϵ-closure (8) = {2, 3, 6, 8}

Now apply input transitions on states.

$\delta'(1,a)$ = ϵ-closure (δ ($\delta'(1,\epsilon)$, a))

= ϵ-closure(δ (ϵ-closure(1), a))

= ϵ-closure(δ(1, 2, 3, 6), a))

= ϵ-closure(δ(1,a) ∪ δ(2,a) ∪ δ(3,a) ∪ δ(6,a))

= ϵ-closure (4, 8) = {2, 3, 4, 5, 6, 8}

$\delta'(1,b)$ = ϵ-closure (δ($\delta'(1,\epsilon)$, b))

= ϵ-closure(δ (1, 2, 3, 6), b)

= ϵ-closure(δ(1,b) ∪ δ(2,b) ∪ δ(3,b) ∪ δ(6,b))

= ϵ-closure (∅) = ∅

$\delta'(2,a)$ = {2, 3, 4, 5, 6, 8}

$\delta'(2,b)$ = {∅}

$\delta'(3,a)$ = {4, 5}

$\delta'(3,b)$ = ∅

$\delta'(4,a)$ = ∅

$\delta'(4,b)$ = {2, 3, 6, 7}

$\delta'(5,a)$ = ∅

$\delta'(5,b)$ = {2, 3, 6, 7}

$\delta'(6,a)$ = {2, 3, 6, 8}

$\delta'(6,b)$ = ∅

$\delta'(7,a)$ = {2, 3, 4, 5, 6, 8}

$\delta'(7,b)$ = ∅

$\delta'(8,a)$ = {2, 3, 4, 5, 6, 8}

$\delta'(8,b)$ = ∅

Transition table is

state \ input	a	b
1	{2,3,4,5,6,8}	ϕ
2	{2,3,4,5,6,8}	ϕ
3	{4,5}	ϕ
4	ϕ	{2,3,6,7}
5	ϕ	{2,3,6,7}
6	{2,3,6,8}	ϕ
7	{2,3,4,5,6,8}	ϕ
8	{2,3,4,5,6,8}	ϕ

Conversion from NFA to DFA:

Let $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA which accepts the language $L(M)$. There should be equivalent DFA denoted by $M' = (Q', \Sigma', \delta', q_0', F')$ such that $L(M) = L(M')$.

The conversion method will follow following steps:

① The start state of NFA M will be the start for DFA M'. Hence add q_0 of NFA to Q'. Then find the transitions from this start state.

② For each state $[q_1, q_2, q_3, \ldots q_i]$ in Q' the transitions for each input symbol can be obtained as,

(i) $\delta'[q_1, q_2, q_3, \ldots q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a) \ldots \cup \delta(q_i, a)$

$= [q_1, q_2, \ldots q_k]$ may be some state.

(ii) Add the state $[q_1, q_2, \ldots q_k]$ to DFA if it is not already added in Q'.

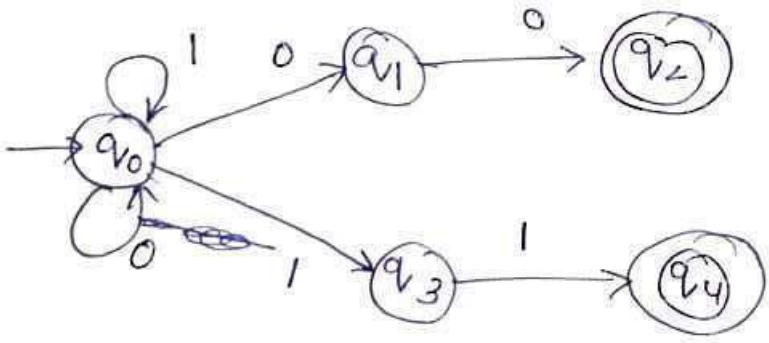
(iii) Then find the transitions for every input symbol from Σ for state $[q_1, q_2, \ldots q_k]$. If we get some state $[q_1, q_2, \ldots q_n]$ which is not in Q' of DFA then add this state to Q'.

(iv) If there is no new state generating then stop the process after finding all the transitions.

3) For the state $[q_1, q_2, \ldots q_n] \in Q'$ of DFA if any one state q_i is a final state of NFA then $[q_1, q_2, \ldots q_n]$ becomes final state. Thus the set of all the final states $\in F'$ of DFA.

Example:

1. Convert the give NFA to its equivalent DFA.



Solution we will first design transition table from given transition diagram.

state \ input	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	$\{q_2\}$	ϕ
*q_2	ϕ	ϕ
q_3	ϕ	$\{q_4\}$
* q_4	ϕ	ϕ

→ As we got a new state $[q_0, q_1]$ we will compute δ' transitions for input 0 & 1.

$$\delta'([q_0,q_1],0) = \delta(q_0,0) \cup \delta(q_1,0)$$
$$= \{q_0,q_1\} \cup \{q_2\}$$
$$= \{q_0,q_1,q_2\}$$

$$\delta'([q_0,q_1],1) = \delta(q_0,1) \cup \delta(q_1,1)$$
$$= \{q_0,q_3\} \cup \{\phi\}$$
$$= \{q_0,q_3\}$$

→ The new state $[q_0,q_3]$ is obtained, we will process it

$$\delta'([q_0,q_3],0) = \delta(q_0,0) \cup \delta(q_3,0)$$
$$= \{q_0,q_1\} \cup \{\phi\}$$
$$= \{q_0,q_1\}$$

$$\delta'([q_0,q_3],1) = \delta(q_0,1) \cup \delta(q_3,1)$$
$$= \{q_0,q_3\} \cup \{q_4\} = \{q_0,q_3,q_4\}$$

→ Now we process newly generated states

$[q_0, q_1, q_2]$ & $[q_0, q_3, q_4]$

$\delta'([q_0, q_1, q_2], 0) = \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)$

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

$\delta'([q_0, q_1, q_2], 1) = \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)$

$= \{q_0, q_3\}$

$\delta'([q_0, q_3, q_4], 0) = \{q_0, q_1\}$

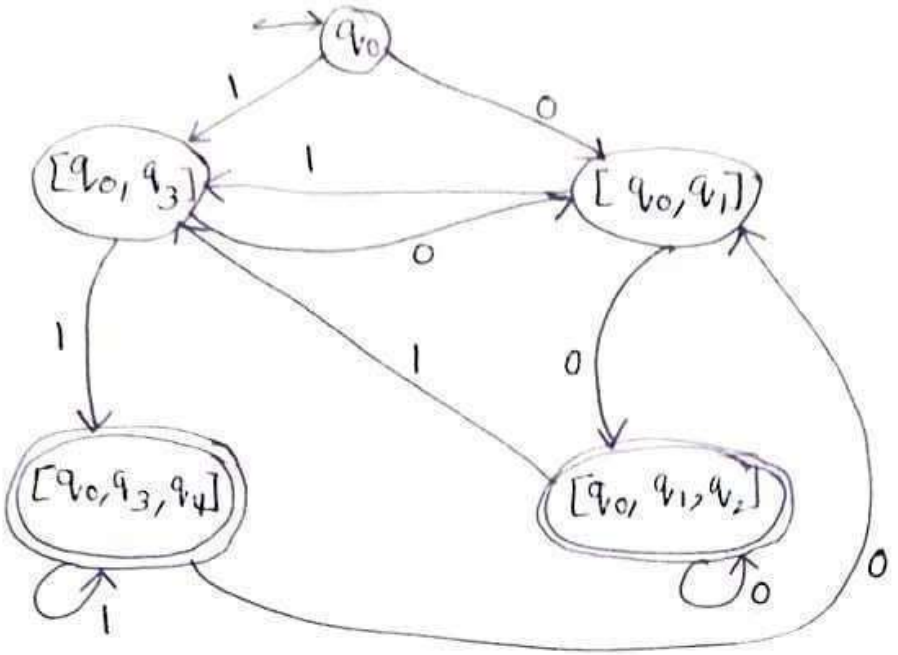
$\delta'([q_0, q_3, q_4], 1) = \{q_0, q_3, q_4\}$

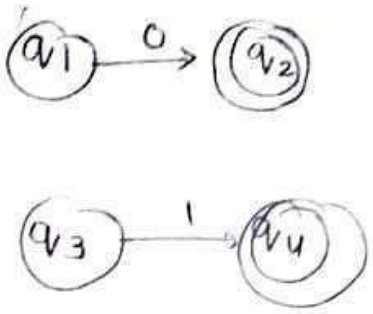
∴ There are no further new states

Transition table is

state \ input	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	$\{q_2\}$	ϕ
*q_2	ϕ	ϕ
q_3	ϕ	$\{q_4\}$
*q_4	ϕ	ϕ
$[q_0, q_1]$	$\{q_0, q_1, q_2\}$	$\{q_0, q_3\}$
$[q_0, q_3]$	$\{q_0, q_1\}$	$\{q_0, q_3, q_4\}$
* $[q_0, q_1, q_2]$	$\{q_0, q_1, q_2\}$	$\{q_0, q_3\}$
* $[q_0, q_3, q_4]$	$\{q_0, q_1\}$	$\{q_0, q_3, q_4\}$

The DFA can be drawn as





Not connected part.

→ Finally Above is the final DFA.

Example:

(2) Let $M = (\{q_0, q_1\}, \{0,1\}, \delta, q_0, \{q_1\})$ be NFA where $\delta(q_0,0) = \{q_0, q_1\}$, $\delta(q_0,1) = \{q_1\}$ $\delta(q_1,0) = \emptyset$ & $\delta(q_1,1) = \{q_0, q_1\}$ construct its equivalent DFA.

Solution: Let the DFA $M' = (Q', \Sigma, \delta', q_0', F')$

Now the δ' function can be computed as

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

$\delta'([q_0], 0) = [q_0, q_1]$.

As in NFA the initial state is q_0, the DFA will also contain the initial state $[q_0]$.

let us draw the transition table for δ function for given NFA.

state \ input	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_1\}$
(q_1)	ϕ	$\{q_0, q_1\}$

δ function for NFA

From the transition table we can compute that there are $[q_0]$, $[q_1]$, $[q_0, q_1]$ states for equivalent DFA. we need to compute the transition from state $[q_0, q_1]$

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

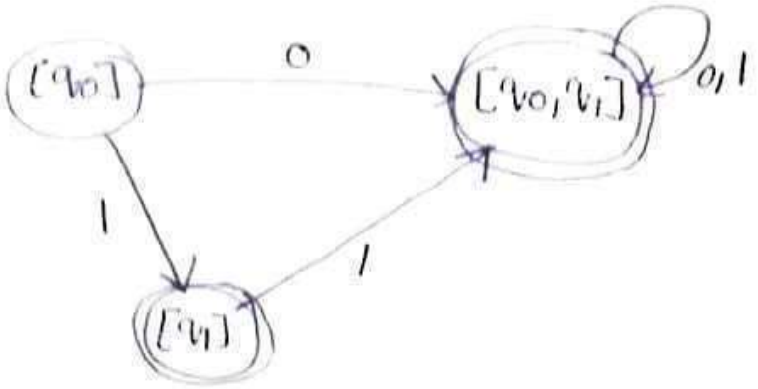
$$\delta'([q_0,q_1],1) = \delta(q_0,1) \cup \delta(q_1,1)$$
$$= \{q_1\} \cup \{q_0,q_1\}$$
$$= \{q_0,q_1\}$$

As in given NFA q_1 is the final state, then in DFA wherever q_1 exists that state becomes a final state. Hence in DFA final states are $[q_1]$ & $[q_0 q_1]$. Therefore set of final states are $F = \{[q_1], [q_0,q_1]\}$

Equivalent DFA is

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

Transition diagram for DFA is



Conversion of NFA with ∈ to DFA:

① Consider $M = (Q, \Sigma, \delta, q_0, F)$ is NFA with ∈. we have to convert this NFA with ∈ to equivalent DFA denoted by

$$M_D = (Q_D, \Sigma, \delta_D, q_0, F_D),$$

Then obtain

∈-closure$(q_0) = \{P_1, P_2, \cdots P_n\}$ then $[P_1, P_2, \cdots P_n]$ becomes a start state of DFA.

Now $[P_1, P_2, P_3 \cdots P_n] \in Q_D$

② we will obtain δ transitions on $[P_1, P_2, P_3 \cdots P_n]$ for each input.

$$\delta_D([P_1, P_2, \cdots P_n], a) = \epsilon\text{-closure}(\delta[P_1,a) \cup \delta(P_2,a) \cup \cdots \delta(P_n,a))$$

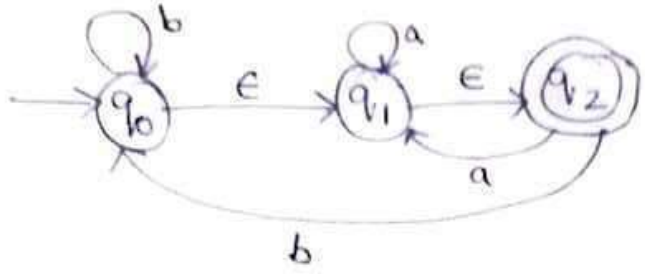
$$= \bigcup_{i=1}^{n} \epsilon\text{-closure}(P_i, a)$$

where a is input $\in \Sigma$.

③ The states obtained $[P_1, P_2, \cdots P_n] \in Q_D$. The states containing final states in P_i is a final state in DFA.

Example

① Convert the following NFA with ϵ to DFA



Solution:

To convert this NFA we will first find ϵ-closures

ϵ-closure$(q_0) = \{q_0, q_1, q_2\}$

ϵ-closure$(q_1) = \{q_1, q_2\}$

ϵ-closure$(q_2) = \{q_2\}$

Now let us start from ϵ-closure of start state.

ϵ-closure$\{q_0\} = \{q_0, q_1, q_2\}$ we will call this state as A

Now let us find transitions on A with every input symbol

$\delta'(A, a) = \epsilon$-closure$(\delta(A, a))$

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

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

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

$= \epsilon$-closure$(q_1) = \{q_1, q_2\}$. let us call it as state B.

$\delta'(A, b) = \epsilon$-closure$(\delta(A, b))$

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

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

$= \epsilon$-closure$(q_0 \cup \phi \cup q_0)$

$= \epsilon$-closure$(q_0) = \{q_0, q_1, q_2\}$ i.e., A

Hence we can state that

$$\delta'(A, a) = B$$

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

Now let us find transitions for state $B - \{q_1, q_2\}$

$$\delta'(B, a) = \delta(\text{closure}(\delta(q_1, q_2), a)$$

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

$$= \epsilon\text{-closure}(q_1 \cup q_1) = \epsilon\text{-closure}(q_1)$$

$$= \{q_1, q_2\} \text{ i.e., } B$$

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

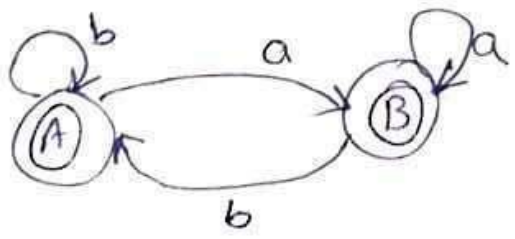
$$= \epsilon\text{-closure}(\delta(q_1, b) \cup \delta(q_2, b))$$

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

$$= \epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\} \text{ i.e., } A$$

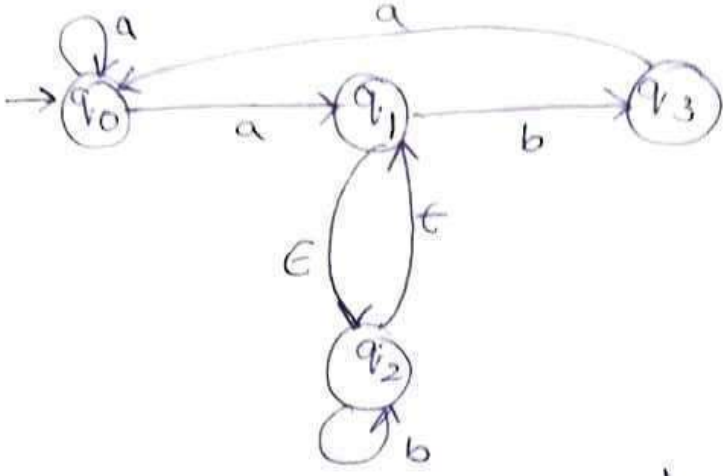
Hence the generated DFA is

	a	b
→ (A)	B	A
(B)	B	A



Example 2

Construct DFA for the following NFA -ε



Solution: we will first write ε-closure of start state q_0.

ε-closure $(q_0) = \{q_0\}$ call it as state A

Now obtain δ transitions for input a & b for state A

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

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

$= \epsilon\text{-closure}(q_0,q_1)$

$= \epsilon\text{-closure}(q_0) \cup \epsilon\text{-closure}(q_1)$

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

$= \{q_0,q_1,q_2\}$ — call it as state B.

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

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

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

Now consider state B ie, $\{q_0,q_1,q_2\}$

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

$= \epsilon\text{-closure}(\delta(q_0,q_1,q_2),a)$

$= \epsilon\text{-closure}(\delta(q_0,a) \cup \delta(q_1,a) \cup \delta(q_2,a))$

$= \epsilon\text{-closure}(\{q_0, q_1\} \cup \phi \cup \phi)$

$= \epsilon\text{-closure}(q_0, q_1) = \epsilon\text{-closure}(q_0) \cup \epsilon\text{-closure}(q_1)$

$= \{q_0, q_1, q_2\}$ i.e., state B.

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

$= \epsilon\text{-closure}(\delta(q_0, q_1, q_2), b))$

$= \epsilon\text{-closure}(\delta(q_0,b) \cup \delta(q_1,b) \cup \delta(q_2,b))$

$= \epsilon\text{-closure}(\phi \cup q_3 \cup q_2)$

$= \epsilon\text{-closure}(q_2, q_3) = \epsilon\text{-closure}(q_2) \cup \epsilon\text{-closure}(q_3)$

$= \{q_1, q_2\} \cup \{q_3\}$

$= \{q_1, q_2, q_3\}$ – call is as state C

The δ transitions on C are obtained as:

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

$= \epsilon\text{-closure}(\delta(q_1, q_2, q_3), a))$

$= \epsilon\text{-closure}(\delta(q_1,a) \cup \delta(q_2,a) \cup \delta(q_3,a))$

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

$= \epsilon\text{-closure}(q_0) = q_0$ i.e., A

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

$= \epsilon\text{-closure}(\delta(q_1, q_2, q_3), b))$

$= \epsilon\text{-closure}(\delta(q_1,b) \cup \delta(q_2,b) \cup \delta(q_3,b))$

$= \epsilon\text{-closure}(q_3 \cup q_2 \cup \phi)$

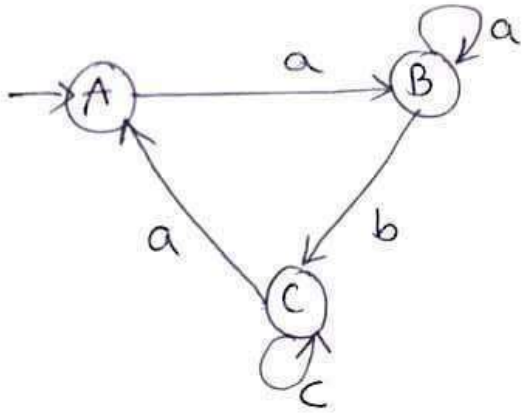
$= \epsilon\text{-closure}(q_2, q_3) = \epsilon\text{-closure}(q_2) \cup \epsilon\text{-closure}(q_3)$

$= \{q_1, q_2\} \cup \{q_3\}$

$= \{q_1, q_2, q_3\}$ i.e., state C

Now no new state is generated. Hence we will write transition table for above computations. (10)

state \ inputs	a	b
→ A	B	ϕ
B	B	C
C	A	C



Finite Automata with output:

There are two types of FA with output and those are:

1. Moore machine.
2. Mealy machine.

Moore machine:

→ Moore machine is a finite state machine in which the next state is decided by current state and current input symbol.

→ The output symbol at a given time depends only on the present state of the machine.

Moore machine is a six tuple representation $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

Q – finite set of states

Σ – finite set of input symbols

Δ – an output alphabet

δ – Transition function such that $Q \times \Sigma \rightarrow Q$

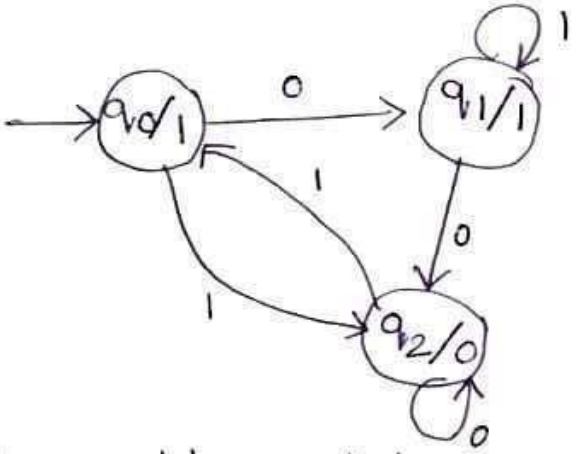
λ – output function $Q \rightarrow \Delta$ (also known as machine function).

q_0 – initial state of machine.

→ In moore machine output is associated with every state. If the length of the input string is 'n' then output string has length 'n+1'.

Example:

Consider the moore machine given below



Transition table will be

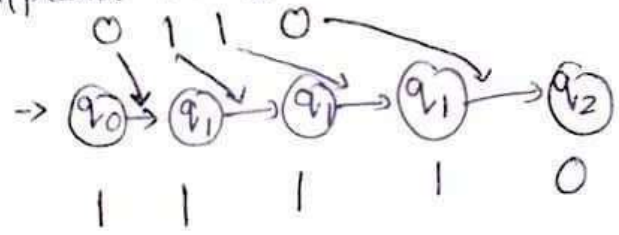
Current state	Next state (δ)		Output (λ)
	0	1	
→q_0	q_1	q_2	1
q_1	q_2	q_1	1
q_2	q_2	q_0	0

→ In moore machine output is associated with every state.

→ In the above given moore machine when machine is in q_0 state the output will be 1.

For the string 0110 then output will be 11110 ie.,

Acceptance of string is



= 11110

Mealy machine:

Mealy machine is a machine in which output symbol depends upon the present input symbol and present state of the machine. The mealy machine can be defined as a six tuple notation

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

Q - finite set of states.

Σ - finite set of input symbols.

Δ - an output alphabet

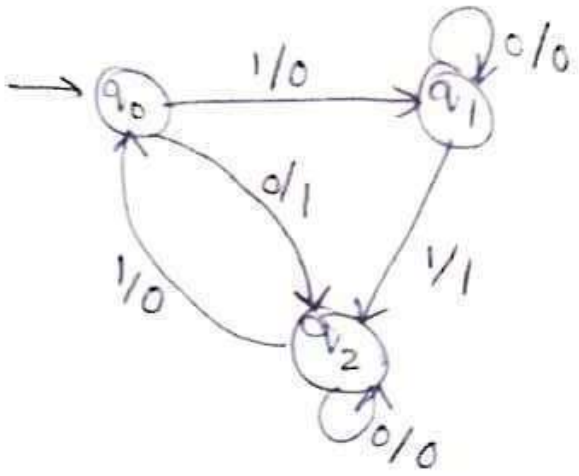
δ - state transition function such that $Q \times \Sigma \to Q$

λ - machine function such that $Q \times \Sigma \to \Delta$

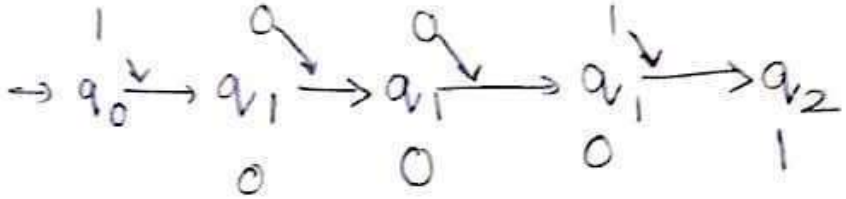
q_0 - initial state.

→ In mealy machine the length of input string is equal to length of output string.

For example.



For input string 1001



Conversion of moore to mealy :

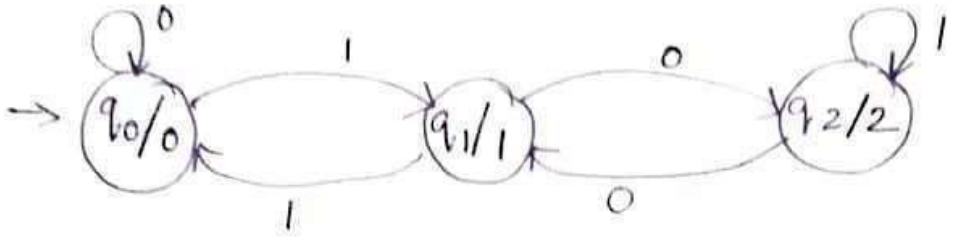
let $M = (Q, \Sigma, \delta, \lambda, q_0)$ be moore m/c, its equivalent mealy machine can be represented as $M' = (Q, \Sigma, \delta, \lambda', q_0)$.

The output function $\lambda'(q, a) = \lambda(\delta(q, a))$

Example convert the following moore m/c into its equivalent mealy m/c.

Q \ Σ	0	1	output (λ)
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

Solution



The output function λ' can be obtained using following rule.

$$\lambda'(q_i, a) = \lambda(\delta(q_i, a))$$

Here we will obtain output for every transition corresponding to input symbol

$\lambda'(q_0, 0) = \lambda(\delta(q_0, 0))$
$= \lambda(q_0)$ ie., output of q_0
$= 0$

$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1))$
$= \lambda(q_1)$
$= 1$

$\lambda'(q_1, 0) = \lambda(\delta(q_1, 0))$
$= \lambda(q_2)$
$= 2$

$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1))$
$= \lambda(q_0)$
$= 0$

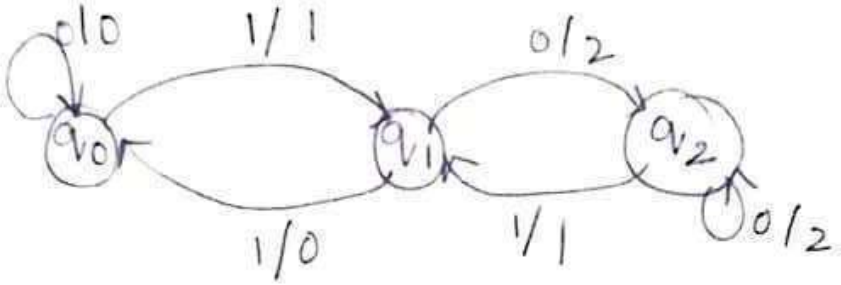
$\lambda'(q_2, 0) = \lambda(\delta(q_2, 0))$
$= \lambda(q_1)$
$= 1$

$\lambda'(q_2, 1) = \lambda(\delta(q_2, 1))$
$= \lambda(q_2)$
$= 2$

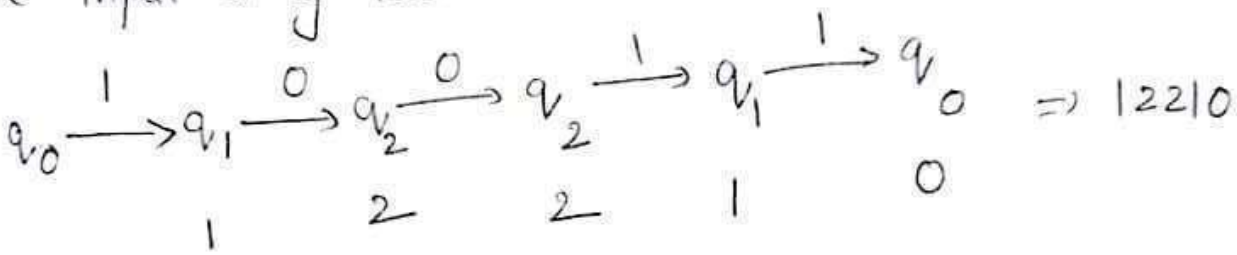
Hence transition table for mealy machine is

Q \ Σ	input = 0		input = 1	
	state	output	state	output
q_0	q_0	0	q_1	1
q_1	q_2	2	q_0	0
q_2	q_1	1	q_2	2

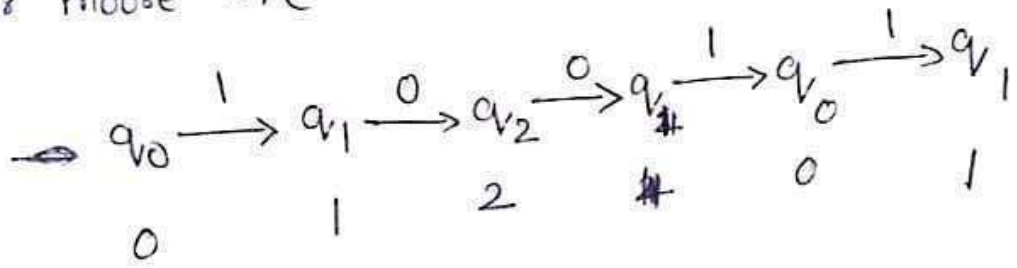
Transition diagram of mealy machine is



The input string 10011 then output for mealy m/c is



For moore m/c



conversion of mealy to moore :

step1 : If transition diagram is given, then convert the transition diagram to transition table.

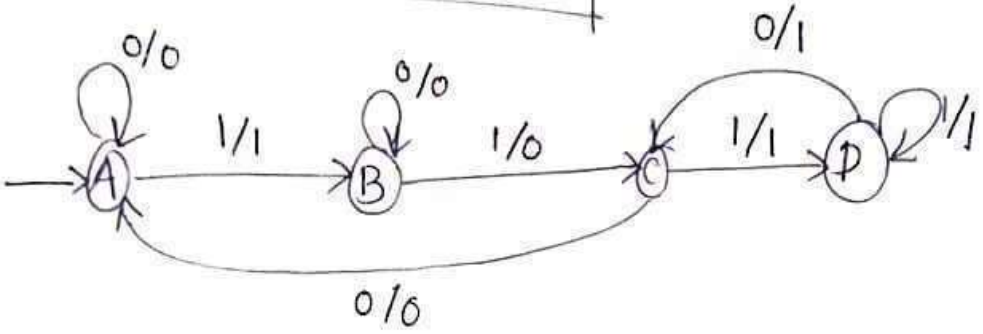
step2: From transition table of mealy machine, find out such kind of states which are showing more than 1 outputs

step3 Then in such cases, divide those states in to more states depending on the outputs associated with them.

Ex Convert the following mealy machine in to its equivalent moore machine. (16)

mealy Transition table.

	0		1	
→ A	A	0	B	1
B	B	0	C	0
C	A	0	D	1
D	A	1	D	1



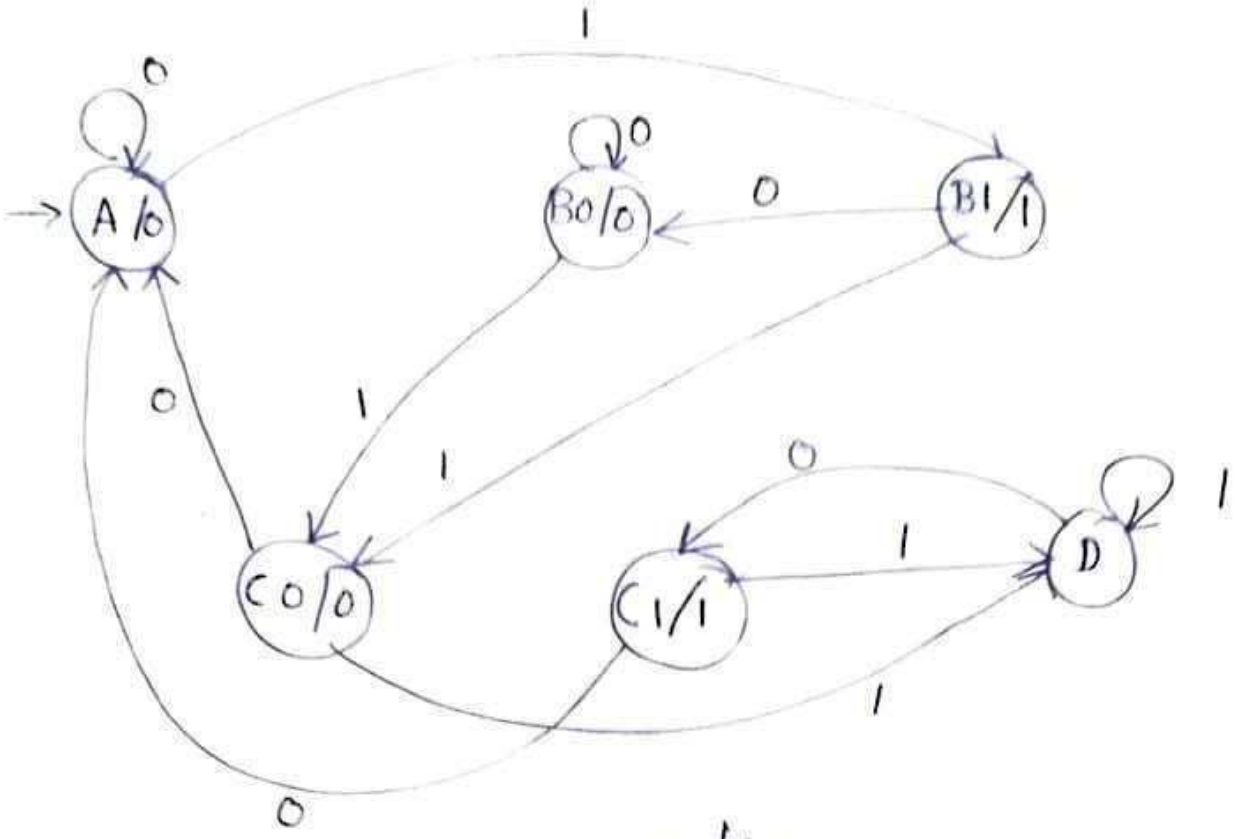
mealy states A B C D

outputs 0 0 1 0 1 1

moore states A B_0 B_1 C_0 C_1 D_1

moore transition table

slate	output 0	output 1	output
→ A	A	B_1	0
B_0	B_0	C_0	0
B_1	B_0	C_0	1
C_0	A	D	0
C_1	A	D	1
D	C_1	D	1



moore machine.

—— * ——