UNIT-I : Fundamentals:

Symbol: A symbol is an abstract entity.
Ex: Letters and digits. (Operators)

Alphabet: An alphabet is a finite set of symbols.
It is defined by 'Σ'.
Ex: {0,1} Σ = {a,b,c}.

power of alphabet is the set of all strings of length 'l' over the alphabet Σ. we use the notation Σ^l,

Ex: $\Sigma^1 = \{0,1\}$ - set of all strings of length '1'

$\Sigma^2 = \{00,01,10,11\}$ - set of all strings of length '2'

$$L^* = \bigcup_{k=0}^{\infty} L^k$$

$$L^+ = \bigcup_{k=1}^{\infty} L^k.$$

String: A string is a finite sequence of symbols.
For example, a, b, and c are symbols and 'abcb' is a Str

The length of the string 'w' denoted by |w|, is the no of symbols composing the string.

Ex: w = abcb |w| = 4

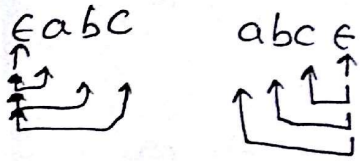
The empty string, denoted by 'ε', is the string consist of zero symbols. |ε| = 0

A prefix of a string is any no-of leading symbols of that string and a suffix is any no-of trailing symbols.

Ex: string 'abc' has

prefixes: ϵ, a, ab, abc

suffixes: ϵ, c, bc, abc



The concatenation of two strings is the string formed by writing first followed by the second, with no intervening space.

The empty string is the identity for the concatenation operator. i.e $\epsilon w = w\epsilon = w$ for each string w

Language:

A language is a set of strings of symbols from some one alphabet.

Ex: ϕ, $\{\epsilon\}$

set of all palindromes over the alphabet $\Sigma = \{0, 1\}$ is an infinite language.

$L = \{\epsilon, 0, 1, 00, 11, 010, 101, \ldots\ldots\}$

Another language is the set of all strings over a fixed alphabet Σ. we denote this language by Σ^*.

If $\Sigma = \{a\}$ then $\Sigma^* = \{\epsilon, a, aa, aaa, \ldots\}$

If $\Sigma = \{0, 1\}$ then $\Sigma^* = \{\epsilon, 0, 1, 01, 10, 00, 11, \ldots\}$

(2)

Operations on languages:

Concatenation:

The concatenation L_1L_2 of languages L_1 and L_2 is defined by

$$L_1L_2 = \{ uv / u \in L_1, v \in L_2 \}$$

Ex: $\Sigma = \{a, b\}$ and L_1 and L_2 are languages over Σ.

$L_1 = \{ab, bb\}$

$L_2 = \{a, b\}$

$L_1L_2 = \{aba, bba, abb, bbb\}$

$L_2L_1 = \{aab, abb, bab, bbb\}$

$$L_1L_2 \neq L_2L_1$$

⇒ Closure operation:

① Star closure (Kleene closure):

This operation defines on set 'S' a derived set S^*, which as members, the empty string and all strings formed by concatenating a finite no of strings in S.

$$S^* = \{ S^0 \cup S^1 \cup S^2 \cdots \} \quad \text{where } S^0 = \epsilon$$

$$S^i = S^{i-1} S \quad \text{for } i > 0$$

Ex: If $S = \{1\}$ then $S^* = \{\epsilon, 1, 11, 111, \cdots\}$

$S = \{0, 1\}$ $S^* = \{\epsilon, 0, 1, 00, 01, 10, 11 \cdots\}$

$S = \{10\}$ $S^* = \{\epsilon, 10, 1010, 101010, \cdots\}$

positive closure

The positive closure of a language is defined as

$$S^+ = \{ S^1 \cup S^2 \cup S^3 \ldots \}$$

Everything is same as star closure except ϵ symbol removed

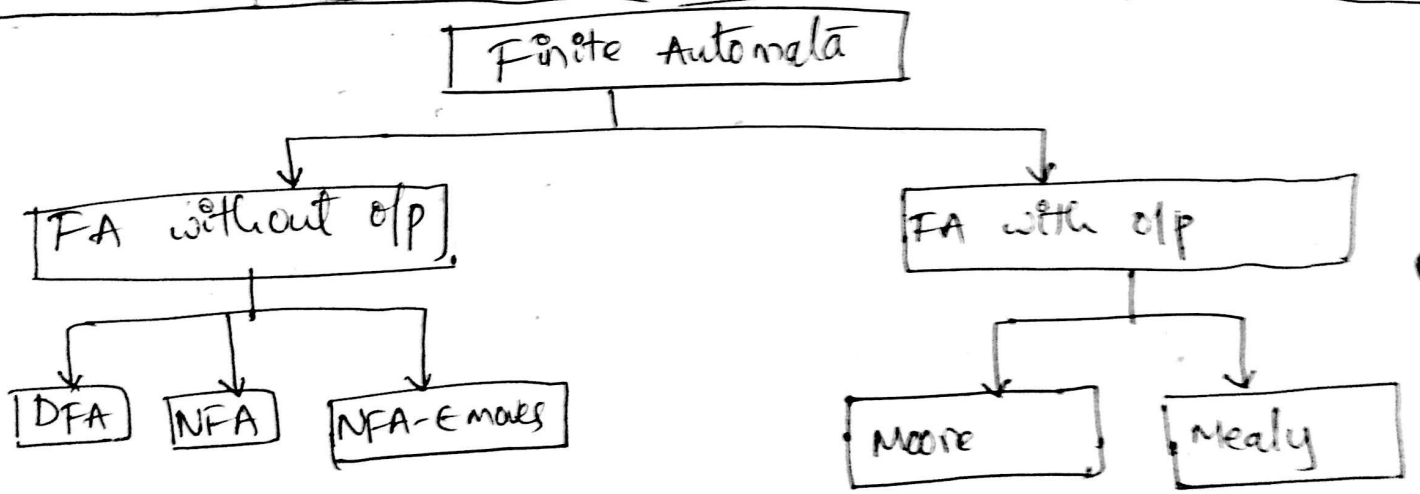
Eg: If $S = \{1\}$ then $S^+ = \{1, 11, 111, \ldots\}$

$S = \{10\}$ $S^+ = \{10, 1010, 101010 \ldots\}$

$$S^* = \bigcup_{i=0}^{\infty} S^k$$

$$S^+ = \bigcup_{i=1}^{\infty} S^k$$

$$S^+ = S^* S$$



Finite Automaton (or) Finite state Machine :-

The finite automaton is a mathematical model of a system, with discrete inputs and outputs. The system can be in any one of a finite no of internal configurations or states.

Ex: The control mechanism of an elevator is a good example of a finite state system. [Automatic machine tools, Automatic packing systems, automatic photo printing machines]

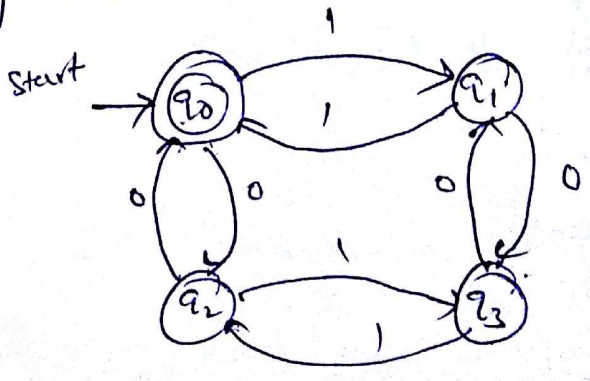
Def:

A Finite automaton (FA) consists of a finite set of states and a set of transitions from state to state that occur on input symbols chosen from an alphabet Σ. For each input there is exactly one transition out of each state. One state, usually denoted q_0, is the initial state, in which the automaton starts. Some states are designated as final or accepting states.

Transition diagram:

It is a directed graph associated with an FA as follows. The vertices of the graph correspond to the states of the FA. If there is a transition from state 'q' to state 'p' on input 'a', then there is an arc labeled 'a' from state 'q' to state 'p' in the transition diagram. The FA accepts a string 'x' if the sequence of transitions corresponding to the symbols of 'x' leads from the start state to an accepting state.

Ex:



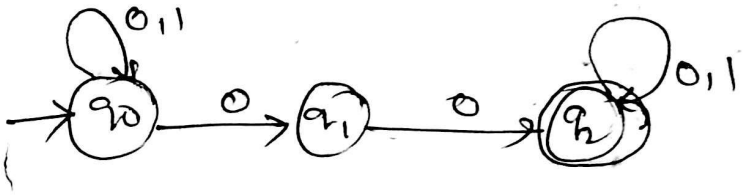
Transition table:

The transition table is a conventional tabular representation of transition function δ that takes two arguments and returns a value.

Ex:

State	Inputs 0	1
→ q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Ex:



State	Inputs 0	1
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	—
(*) q_2	$\{q_2\}$	$\{q_2\}$

applications

1. lexical analyzers
2. switching circuits [control unit]
3. Text editors.
4. Traffic signals.

FSM (or) FA :- (Deterministic Finite Automata)

we formally denote a finite automaton by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

In DFA, for each input symbol, one can determine the state to which the machine will move. hence it is called DFA.

Q is a finite set of states.

Σ is a finite set of input symbols [input alphabet]

$q_0 \in Q$ is an initial state (start state)

$F \subseteq Q$ is the set of final states

δ is the transition function mapping $Q \times \Sigma$ to Q.

i.e $\delta(q, a)$ is a state for each state 'q' and i/p symbol 'a'

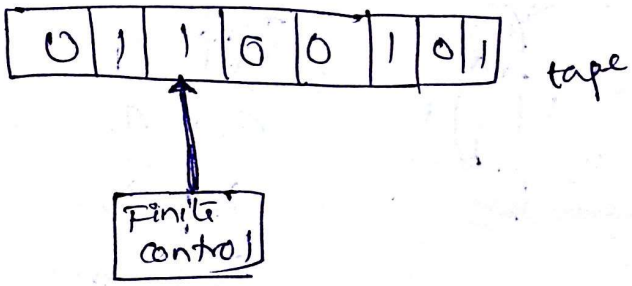
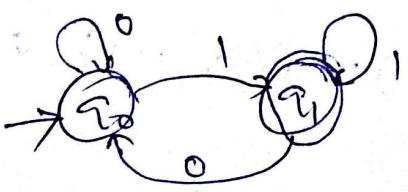


Fig: A finite automaton

A finite control, which is in some state from 'Q', reading a sequence of symbols from Σ written on a tape as shown in the fig. In one move the FA in state 'q' and scanning symbol 'a' enters state $\delta(q, a)$ and moves its head one symbol to the right.

Example:



Extending transition function from single symbol to string:

① $\delta(q, \epsilon) = q$ and

② $\delta(q, wa) = \delta(\delta(q, w), a)$

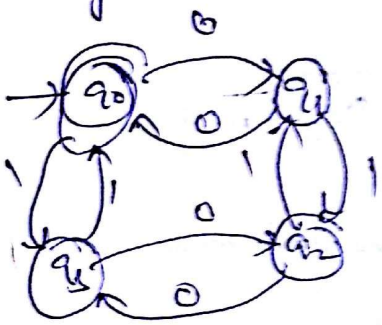
Acceptance

⇒ A string 'x' is said to be accepted by a FA $M = (Q, \Sigma, \delta, q_0, F)$ if $\delta(q_0, x) = p$ for some p in F.

The language accepted by M, designated $L(M)$, is the set $\{x \mid \delta(q_0, x) \text{ is in } F\}$.

A language is a ~~set to be~~ regular set if it is the set accepted by some FA.

Example



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

String: 0001

$\delta(q_0, 0001) \vdash \delta(\delta(q_0, 0), 001)$

$\vdash \delta(q_1, 001)$

$\vdash \delta(q_0, 01)$

$\vdash \delta(q_1, 1) \vdash q_2 \notin$ Final state

0001 is not accepted.

String: 0011

$\delta(q_0, 0011) \vdash \delta(q_1, 011) \vdash \delta(q_0, 11) \vdash \delta(q_3, 1) \vdash q_0 \in F$

0011 is accepted.

Non-Deterministic Finite Automata [NFA]

A NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

Q is a finite non-empty set of states.

Σ is a " " " inputs

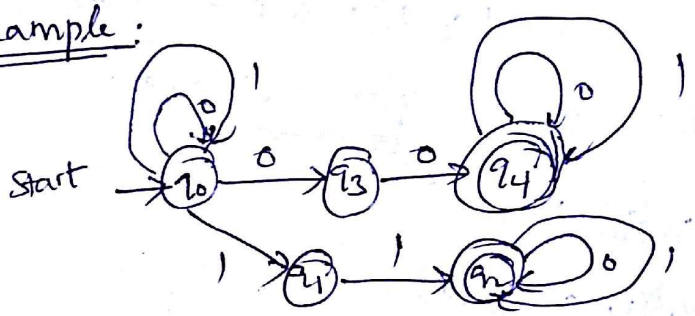
$q_0 \in Q$ is the initial state.

$F \subseteq Q$ is the set of final states.

δ is the transition function mapping from $Q \times \Sigma \rightarrow 2^Q$ which is power set of Q [The set of all subsets of Q]

→ NFA will allow zero, one (or) more transitions from a state on the same input symbol.

Example:



NFA (Transition diagram)

States	Inputs 0	1
→ q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	—	q_2
q_2	q_2	q_2
q_3	q_4	—
(q_4)	q_4	q_4

In NFA, for a particular input symbol, the machine can move to any combination of the states in the machine. Here, the exact state to which the machine moves cannot be determined. Hence it is NDFA.

Acceptance by NFA

If $M=(Q, \Sigma, \delta, q_0, F)$ is an NFA, then $L(M) = \{w \mid \delta(q_0, w) \cap F \neq \phi\}$. That is, $L(M)$ is the set of strings w in Σ^* such that $\delta(q_0, w)$ containing atleast one accepting state.

From the above NFA:

Check the following strings are accepted by NFA or not:

① 01 ② 010 ③ 0100 ④ 01001

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

$= \delta(\{q_0, q_3\}, 1) = \delta(q_0, 1) \cup \delta(q_3, 1)$

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

since $\{q_0, q_1\} \cap F = \phi$, 01 is not accepted.

② $\delta(q_0, 010) = \delta(\{q_0, q_3\}, 10)$

$= \delta(q_0, 10) \cup \delta(q_3, 10)$

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

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

010 is not accepted.

③ $\delta(q_0, 0100) = \delta(\delta(q_0, 0), 100)$

$= \delta(\{q_0, q_3\}, 100)$

$= \delta(q_0, 100) \cup \delta(q_3, 100)$

$= \delta(\{q_0, q_3\}, 00) \cup \phi$

$= \delta(q_0, 00) \cup \delta(q_1, 00)$

$= \delta(\{q_0, q_3\}, 0) \cup \phi$

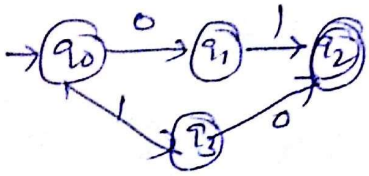
$= \delta(q_0, 0) \cup \delta(q_3, 0) = \{q_0, q_4\} \cap F \neq \phi$

So 0100 is accepted

TUTORIAL - I

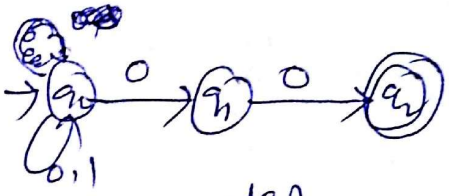
8) Construct a F.A to the following languages:

(1) $L = \{01, 10\}$



(2) set of all strings ending with '00' [$(0+1)^*00$]

$L = \{00, 000, 100, 00\ldots00, 1\ldots00, 0101\ldots00, 1010\ldots00\}$



NFA

This FA is defined as 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

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

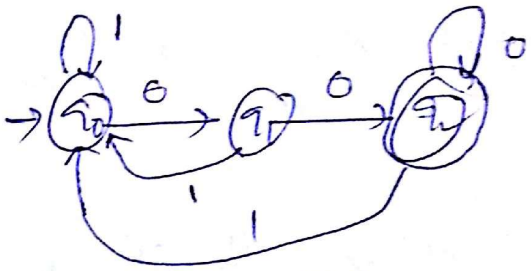
$\Sigma = \{0, 1\}$

$q_0 = q_0$

$F = \{q_2\}$

'δ' is defined as

States	Inputs 0	1
→q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	—
(q_2)	—	—



DFA

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

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

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

$q_0 = q_0$

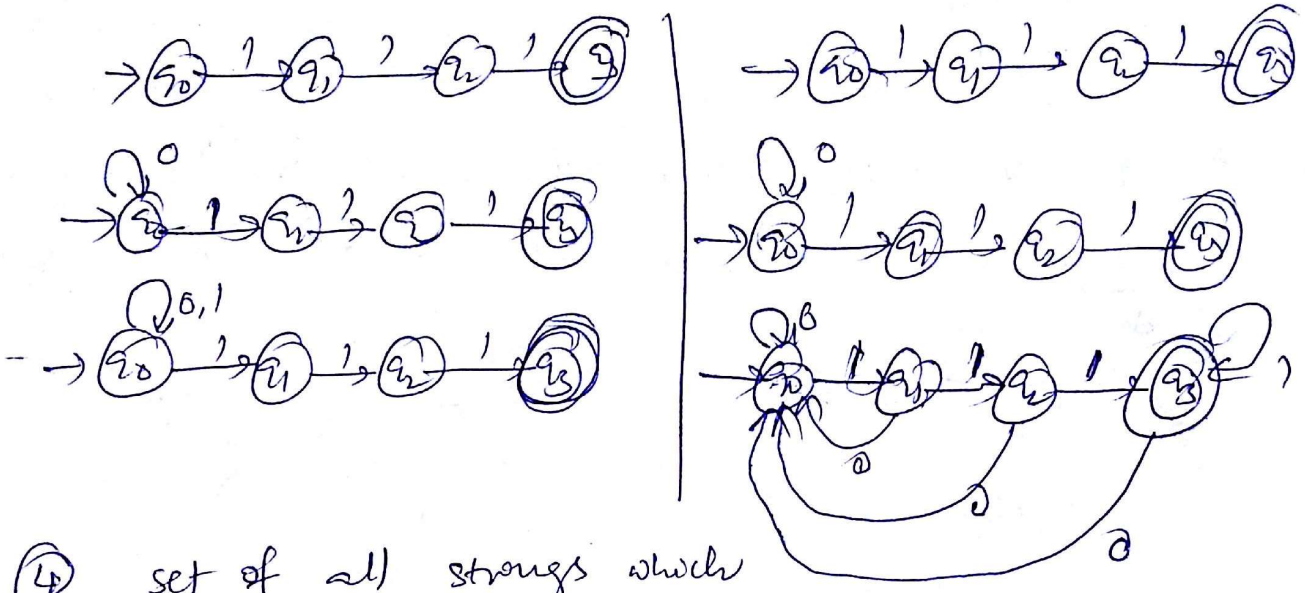
$F = \{q_2\}$

'δ' is defined as

States	Inputs 0	1
→q_0	q_1	q_0
q_1	q_2	q_0
(q_2)	q_2	q_0

(3) set of all strings ending with 111 from $\Sigma = \{0, 1\}$

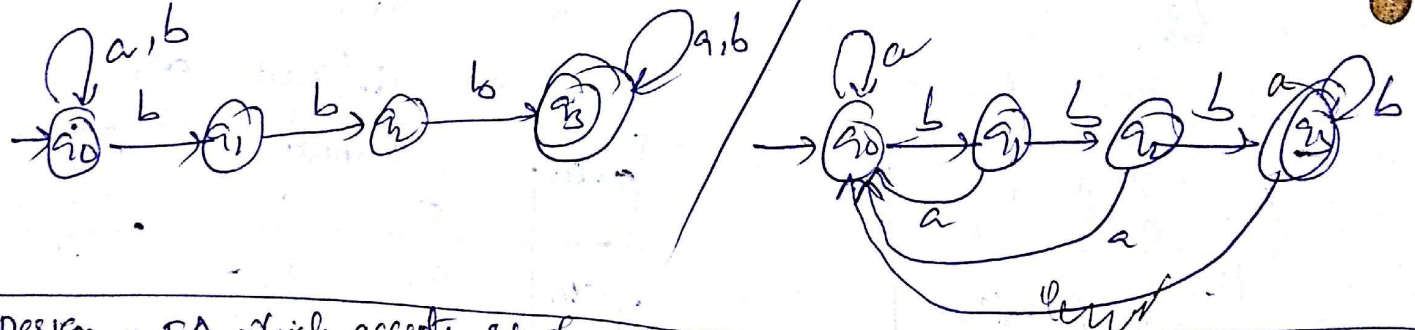
$L = \{ 111, 0111, 0\cdots0111, 1\cdots111, 0101\cdots111, 1010\cdots111 \cdots \}$



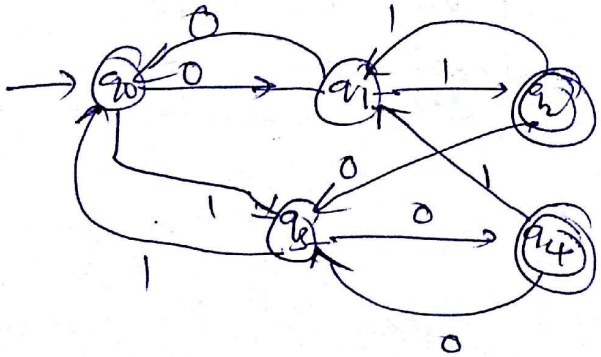
(4) set of all strings which consists of three consecutive b's for the alphabet $\{a, b\}$

$L = \{ bbb, aa\cdots bbb, bbb\cdots b, aa\cdots bbb\cdots bbb, abab\cdots bbbabab\cdots, bababa\cdots bbbabab\cdots \}$

$(a+b)^* bbb (a+b)^*$



(5) Design a FA which accepts set of all strings containing odd no of 0's and odd no of 1's.

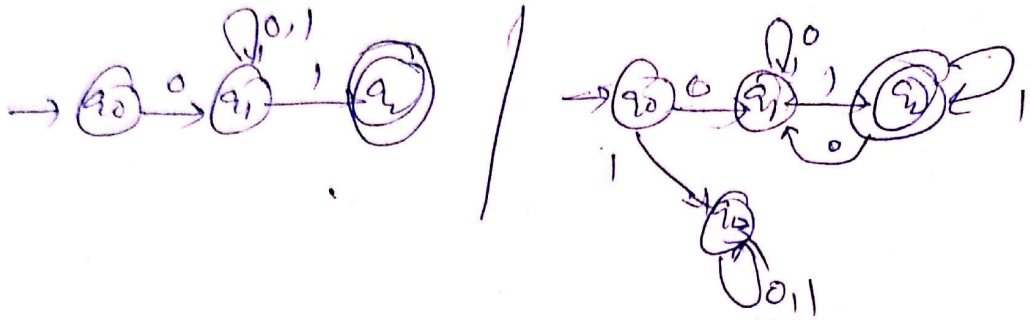


$L = \{ 01, 10, 0111, 01111, \cdots 1000, 100000, 1110, 1111 \cdots \}$

(5) set of all strings begin with '0' and end with '1'

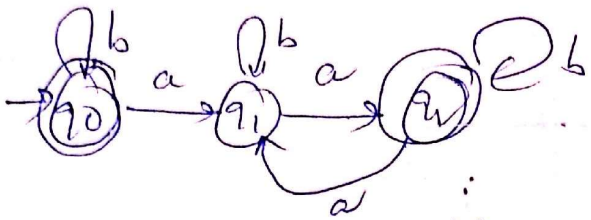
$0\ (0+1)^*\ 1$

$L = \{ 01, 0\cdot0\cdots01, 01\cdots11, \quad 0\ 010101\ 1, 0\ 101010\cdots1 \ \ldots \}$



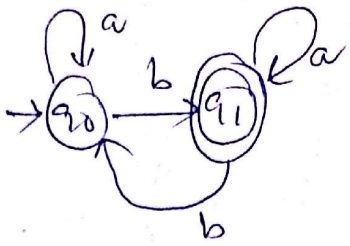
(6) FA which accepts even no of a's over the alphabet $\Sigma = \{a, b\}$

$L = \{ \epsilon, aa, aaaa, \ldots, bb\ldots aa, ababbb\ldots, bababb\ldots \}$

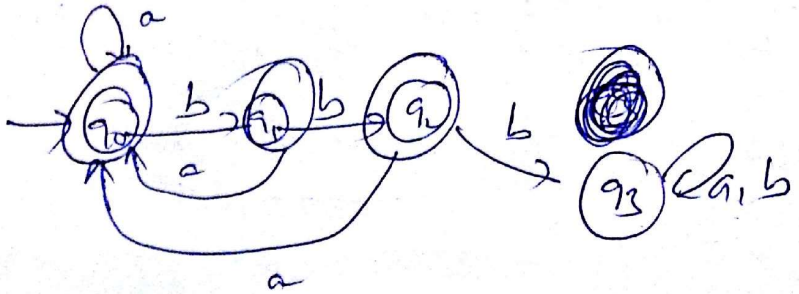


(7) FA which accepts odd no of b's over $\{a, b\}$

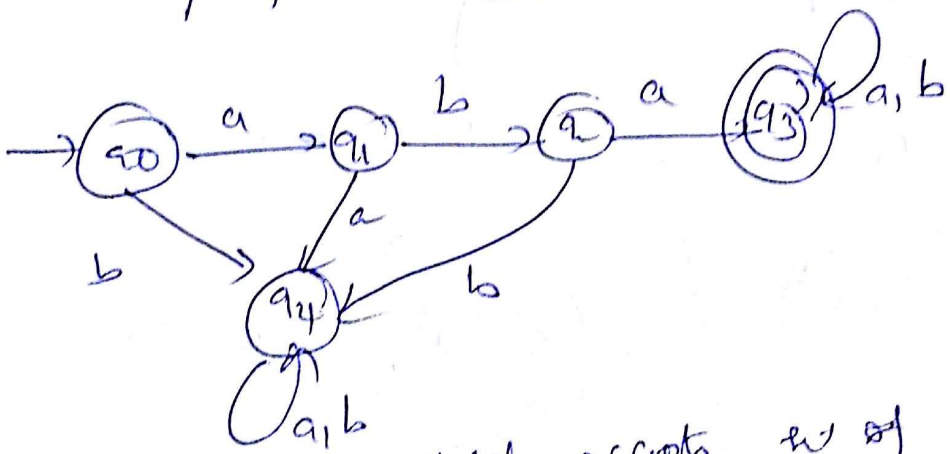
$L = \{ b, aaa\ldots b, baa\ldots, bbb, \ldots, ababb, aba, babba\ldots \}$



(8) Design a DFA which accepts set of all strings should not contain three consecutive b's.

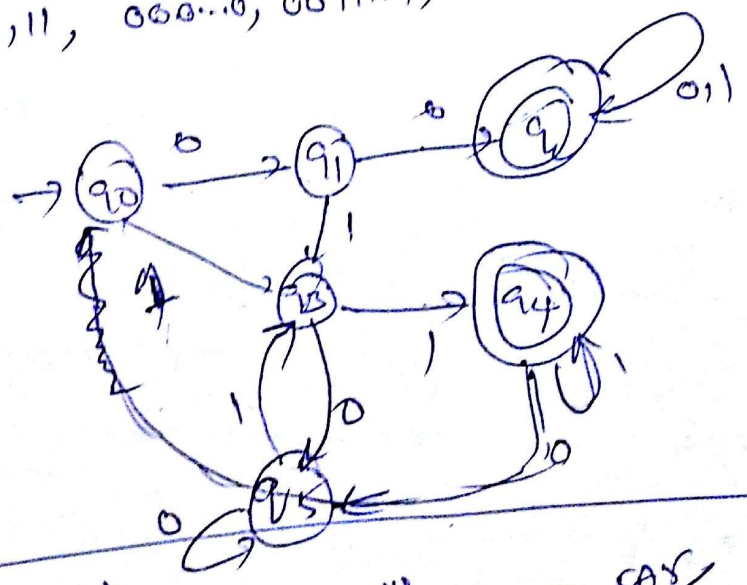


(8) Design a FA which accepts set of all strings whose prefix is 'aba'



(9) Design a FA which accepts set of all strings start with 00 and ending with 11

L = { 00, 11, 000...0, 00 11...1, 001010..., 00...011, 0101...11, 00 11, ... }

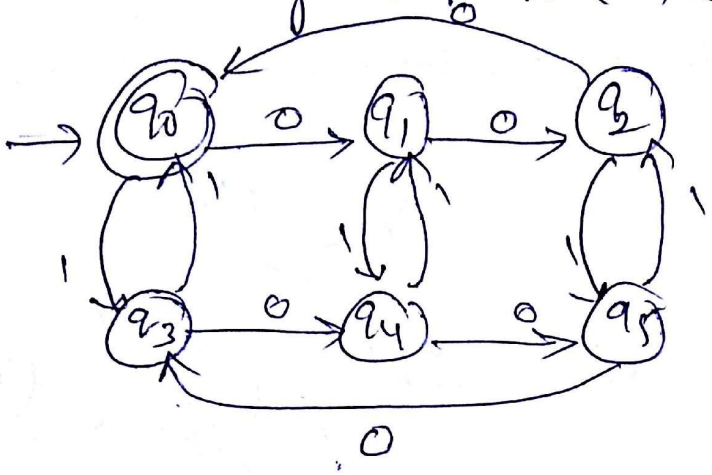


01101

Scanned by CamScanner

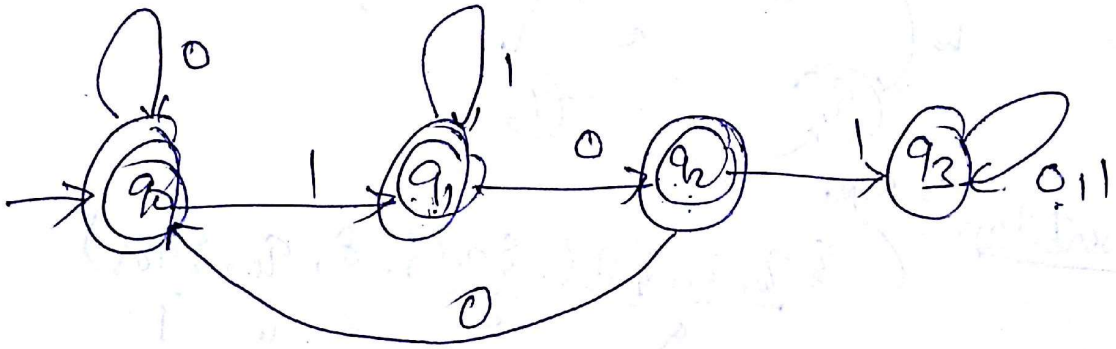
Q) Design a DFA which accepts set of all strings over $\{0,1\}$ in which no-of 0's are divisible '3' and no-of 1's are divisible by '2'. $L = \{\epsilon, aaa, bb, 00011, 11000, 01010, \ldots\}$

Sol:



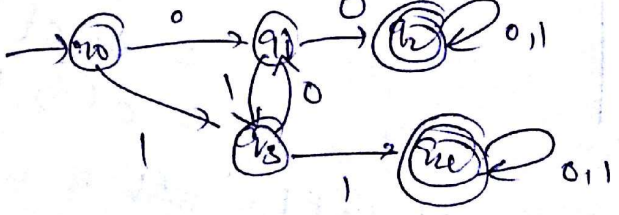
Q) Design a DFA which accepts strings of 0's and 1's should not contain "10" as a substring.

Sol: $L = \{\epsilon, 0, 1, 00, 01, 10, 11, 1111\ldots10, 11\ldots100, 0111111\ldots\}$



Q) Design a FA which accepts all strings with either two consecutive 0's (or) two consecutive 1's over the alphabet $\Sigma = \{0,1\}$.

Sol:

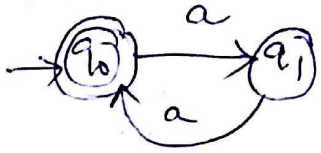


Q). Design a DFA to accept set of all strings over $\{a,b\}$ consists of even no of a's and even no of b's.

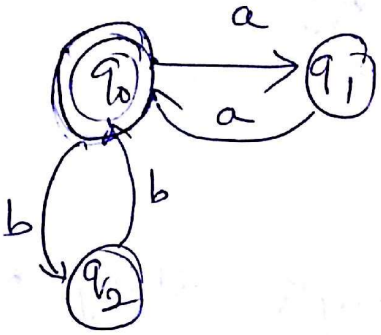
Sol: $L = \{\in, aa, bb, abab, baba, aaaa, bbbb, \ldots\}$

Step 1:

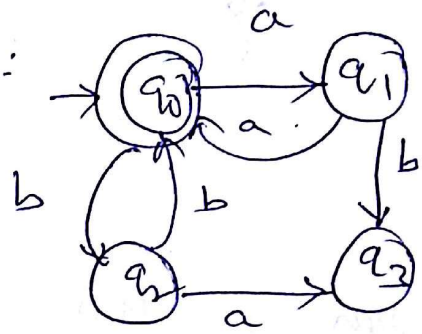
Step 2:



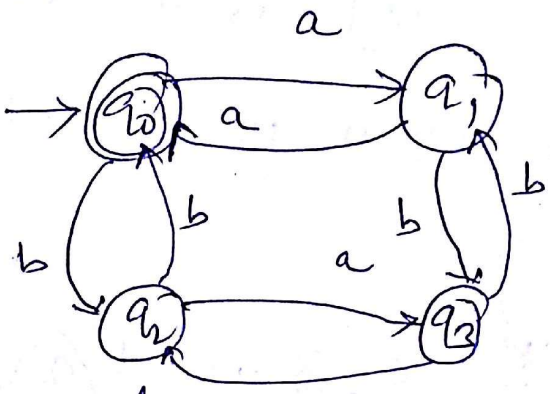
Step 3:



Step 4:



Step 5:



Transition diagram

5-tuple Representation

$$(\underbrace{\{q_0, q_1, q_2, q_3\}}_{Q}, \underbrace{\{a,b\}}_{\Sigma}, \underbrace{\delta}_{\delta}, \underbrace{q_0}_{q_0}, \underbrace{\{q_0\}}_{F})$$

where 'δ' is defined as

Transition table

Inputs / States	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_0	q_3
q_2	q_3	q_0
q_3	q_2	q_1

verification

$$\begin{aligned}
\delta(q_0, abab) &= \delta(\delta(q_0,a), bab) \\
&= \delta(q_1, bab) \\
&= \delta(\delta(q_1,b), ab) \\
&= \delta(q_3, ab) \\
&= \delta(\delta(q_3,a), b) \\
&= \delta(q_2, b) = q_0
\end{aligned}$$

Since '$q_0 \in F$', 'abab' is accepted

II. FINITE AUTOMATA

~~UNIT II~~

The equivalance of DFA and NFA: [NFA to DFA]

* **Theorem:**

For every NFA, there exists a DFA which simulates the behaviour of NFA. Alternatively, if L is the set accepted by NFA, then there exists a DFA which also accepts L.

Proof: Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA accepting L.

we construct DFA M' as follows:

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

(i) $Q' = 2^Q$ (any state in Q' is denoted by $[q_1, q_2, \cdots q_j]$, where $q_1, q_2, \cdots q_j \in Q$);

(ii) $q_0' = [q_0]$; (Initial state)

(iii) F' is the set of all subsets of Q containing an element of F.

(iv) Construct δ' as follows:

$$\delta'([q_1, q_2, \cdots q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \cdots \delta(q_i, a).$$

$$\delta'([q_1, q_2, \cdots q_i], a) = [p_1, \cdots p_j] \text{ iff } \delta(\{q_1, \cdots q_i\}, a) = \{p_1, p_2, \cdots p_j\}$$

⇒ [we start the construction of δ' for $[q_0]$, we continue by considering only states appearing earlier under i/p columns and construct δ' for such states. we halt when no more new states appear under the i/p column]

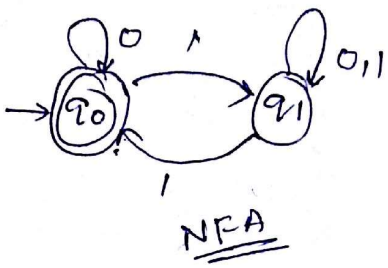
↳ The finite automata M_1 and M_2 are said to be equivalent if given language is accepted by both.

ie $L(M_1) = L(M_2)$.

Example:

(1) Construct DFA equivalent to $M = (\{q_0, q_1\}, \{0,1\}, \delta, q_0, \{q_1\})$

δ is given by its state table:



NFA

States / Σ	0	1
→ q_0	q_0	q_1
q_1	q_1	q_0, q_1

Sol:

(i) $[q_0]$ is the initial state.

(ii) $[q_0]$ and $[q_0, q_1]$ are the final states as these are the only states containing q_0; and

(iii) δ' is defined by the state table given below:

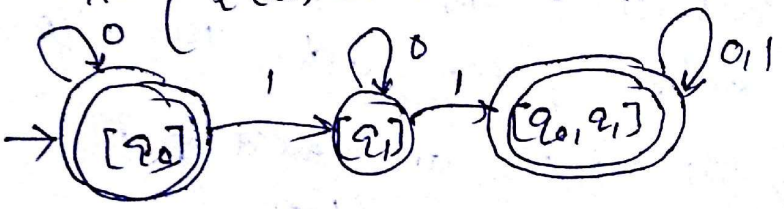
States / Σ	0	1
→ $[q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

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

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

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

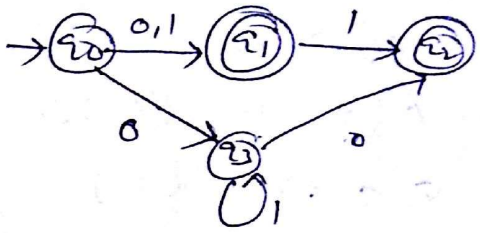
DFA is $M_1 = (\{[q_0], [q_1], [q_0, q_1]\}, \{0,1\}, \delta', [q_0], \{[q_0], [q_0, q_1]\})$



(7) Construct a D.F.A equivalent to NFA

$M = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, \{q_{1,2}\})$ where δ is

δ	0	1
~~$\rightarrow q_0$~~	~~$q_0 q_1$~~	~~q_0~~
$\rightarrow q_0$	q_1, q_3	q_1
q_1	ϕ	q_2
q_2	—	—
q_3	q_3	q_3

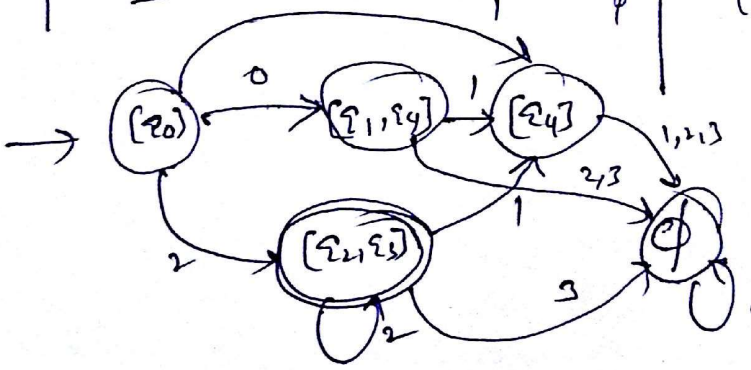


δ	0	1
$[q_0]$	$[q_1, q_3]$	$[q_1]$
$[q_1]$	ϕ	$[q_2]$
$[q_2]$	ϕ	ϕ
$[q_1, q_2]$	$[q_1, q_2, q_3]$	$[q_1, q_3]$
$[q_1, q_2, q_3]$	$[q_1, q_3]$	$[q_2, q_3]$
$[q_2, q_3]$	$[q_2]$	$[q_3]$
$[q_3]$	$[q_3]$	$[q_3]$

(8) Construct a DFA equivalent to NFA

States	inputs 0	ϕ	1	2
$\rightarrow q_0$	q_1, q_4		q_4	q_2, q_3
q_1	—		q_4	—
q_2	—		—	$q_2 q_3$
q_3	—		q_4	—
q_4	—		—	—

States	inputs 0	1	2
$\rightarrow [q_0]$	$[q_1, q_4]$	$[q_4]$	$[q_2, q_3]$
$[q_1, q_4]$	ϕ	$[q_4]$	$[\phi]$
$[q_4]$	ϕ	ϕ	ϕ
$[q_2, q_3]$	ϕ	$[q_4]$	$[q_2, q_3]$
ϕ	ϕ	ϕ	ϕ



(3)

(a) Convert the following NFA into equivalent DFA

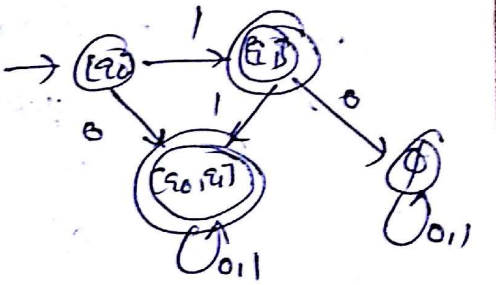
Sol:

$M = (\{q_0, q_1\}, \{0,1\}, \delta, q_0, \{q_1\})$

States	Inputs 0	1
→ q_0	q_0, q_1	q_1
q_1	ϕ	q_0, q_1

$M_1 = (2^Q, \{0,1\}, \delta', [q_0], F)$

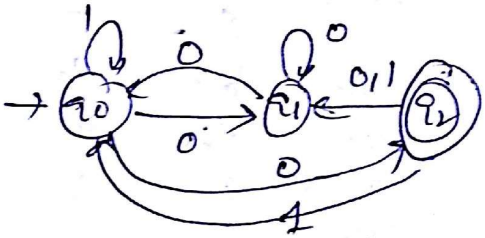
States	Inputs 0	1
→ $[q_0]$	$[q_0, q_1]$	$[q_1]$
$[q_1]$	ϕ	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$
ϕ	ϕ	ϕ



(b) Construct a DFA equivalent to the following NFA

$M = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_2\})$ check δ' s

δ	0	1
→ q_0	q_1, q_2	q_0
q_1	q_0, q_1	ϕ
(*) q_2	q_1	q_0, q_1



Sol

δ'	0	1
→ $[q_0]$	$[q_1, q_2]$	$[q_0]$
(*) $[q_1, q_2]$	$[q_0, q_1]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0]$
(*) $[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$

3 (b) Find a DFA equivalent to

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

δ is given in the table

States	Inputs a	Inputs b
$\rightarrow q_0$	q_0, q_1	q_2
q_1	q_0	q_1
(q_2)	—	$q_0 q_1$

Sol: $M_1 = (2^Q, \{a, b\}, \delta', [q_0], F')$

where $F = \{[q_2], [q_0, q_2], [q_0, q_1, q_2]\}$

States	Inputs a	Inputs b
$\rightarrow [q_0]$	$[q_0, q_1]$	q_2
(q_2)	ϕ	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_1, q_2]$
$([q_1, q_2])$	$[q_0]$	$[q_0, q_1]$

(a) Construct a deterministic F.A equivalent to

$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta', q_0, \{q_3\})$. δ' is given in the table

States	Inputs a	Inputs b
$\rightarrow q_0$	q_0, q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_3
(q_3)	—	q_2

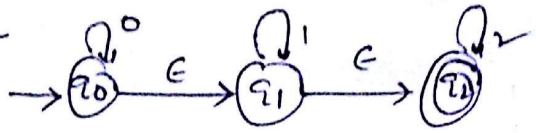
Sol: $M_1 = (2^Q, \{a, b\}, \delta', [q_0], F)$

States	Inputs a	Inputs b
$\rightarrow [q_0]$	$[q_0, q_1]$	q_0
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$
$([q_0, q_1, q_2])$	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$
$([q_0, q_1, q_2, q_3])$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$

(2)

Finite Automata with ϵ-moves :-
(epsilon)

Ex: ①



F.A with ϵ-moves

A NFA with ϵ-moves is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ with all components as before, but δ, the transition-function, maps

$Q \times (\Sigma \cup \{\epsilon\})$ to 2^Q.

The transition-table for the above example is given below:

States	Inputs 0	1	2	ϵ
$\rightarrow q_0$	$\{q_0\}$	ϕ	ϕ	$\{q_1\}$
q_1	ϕ	$\{q_1\}$	ϕ	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$	ϕ

$\delta(q, a)$ for the NFA

Extension of transition function from δ to $\hat{\delta}$:

→ The extended transition-function $\hat{\delta}$ maps $Q \times \Sigma^*$ to 2^Q.

→ $\hat{\delta}(q, w)$ is the set of all states p such that one can go from q to p along a path labeled w, perhaps including edges labeled ϵ

*→ In constructing $\hat{\delta}$ it will be important to compute the set of states reachable from a given state q using ϵ transitions only.

ϵ-CLOSURE(q) is used to denote the set of all vertices p such that there is a path from q to p labeled ϵ.

↳ In the above example

$$\begin{cases} \epsilon\text{-CLOSURE}(q_0) = \{q_0, q_1, q_2\} = \hat{\delta}(q_0, \epsilon) \\ \epsilon\text{-CLOSURE}(\hat{\delta}(q_0, a)) = \hat{\delta}(q_0, a) = \phi \end{cases}$$

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

④

$\hat{\delta}$ is defined as follows:

(i) $\hat{\delta}(q, \epsilon) = \epsilon\text{-CLOSURE}(q)$

(ii) For w in Σ^* and a in Σ,

$\hat{\delta}(q, wa) = \epsilon\text{-CLOSURE}(P)$, where $P = \{p \mid$ for some r in $\hat{\delta}(q,w)$, p is in $\delta(r,a)\}$

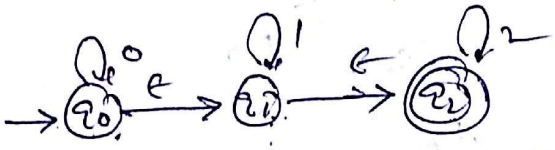
It is convenient to extend δ and $\hat{\delta}$ to set of states by

(iii) $\hat{\delta}(R, a) = \bigcup_{q \in R} \delta(q,a)$, and

(iv) $\hat{\delta}(R, w) = \bigcup_{q \in R} \hat{\delta}(q,w)$ for the set of states R.

we define $L(M)$, the language accepted by $M = (Q, \Sigma, \delta, q_0, F)$ to be $\{w \mid \hat{\delta}(q_0, w)$ contains a state in $F\}$.

Ex:



$\hat{\delta}(q_0, \epsilon) = \epsilon\text{-CLOSURE}(q_0) = \{q_0, q_1, q_2\}$

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

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

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

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

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

$\hat{\delta}(q_0, 01) = \epsilon\text{-CLOSURE}(\delta(\hat{\delta}(q_0, 0), 1))$

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

$= \epsilon\text{-CLOSURE}(q_1)$

$= \{q_1, q_2\}$

Equivalence of N.F.A's with and without ϵ-moves:

Theorem:

* If L is accepted by an NFA with ϵ-transitions, then L is accepted by an NFA without ϵ-transitions.

Procedure to convert NFA with ϵ moves to NFA without ϵ-moves:

Let $M=(Q, \Sigma, \delta, q_0, F)$ is a NFA with ϵ moves then there exists $M'=(Q, \Sigma, \hat{\delta}, q_0, F')$ without ϵ moves:

① First find ϵ-CLOSURE of all states in the design.

② Calculate extended transition functions using following conversion formulae:

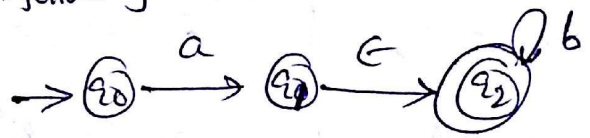
(i) $\hat{\delta}(q,a) = \epsilon\text{-CLOSURE}(\delta(\hat{\delta}(q,\epsilon),a))$

(ii) $\hat{\delta}(q,\epsilon) = \epsilon\text{-CLOSURE}(q)$

③ F' is a set of all states whose ϵ-CLOSURE contains a final state in F.

Example:

Convert following NFA with ϵ-moves to NFA without ϵ-moves:



Sol: ① $\epsilon\text{-CLOSURE}(q_0) = \{q_0\} = \hat{\delta}(q_0,\epsilon)$

$\epsilon\text{-CLOSURE}(q_1) = \{q_1,q_2\} \checkmark = \hat{\delta}(q_1,\epsilon)$

$\epsilon\text{-CLOSURE}(q_2) = \{q_2\} \checkmark = \hat{\delta}(q_2,\epsilon)$

② $\hat{\delta}(q_0,a) = \epsilon\text{-CLOSURE}(\delta(\hat{\delta}(q_0,\epsilon),a))$

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

$= \epsilon\text{-CLOSURE}(q_1)$

$= \{q_1,q_2\}$

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

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

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

$\hat{\delta}(q_1,a) = \epsilon\text{-CLOSURE}(\delta(\hat{\delta}(q_1,\epsilon),a))$

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

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

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

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

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

$\hat{\delta}(q_1,b) = \epsilon\text{-CLOSURE}(\delta(\hat{\delta}(q_1,\epsilon),b))$

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

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

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

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

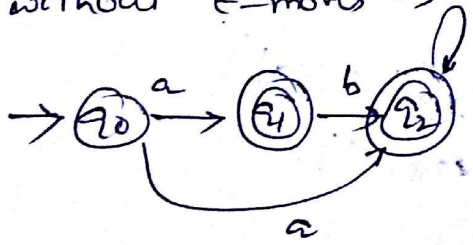
$= \epsilon\text{-CLOSURE}(q_2) = q_2$

$\hat{\delta}$	a	b
$\rightarrow q_0$	$\{q_0,q_2\}$	ϕ
(q_1)	ϕ	q_2
(q_2)	ϕ	q_2

(iii) Final states are q_1, q_2 because

* $\epsilon\text{-CLOSURE}(q_1)$ contains final state
  $\epsilon\text{-CLOSURE}(q_2)$ contains final state.

NFA without ϵ-moves is



Mealy and Moore models:

Finite automata with outputs:

The value of the output function $z(t)$ in the most general case is a function of the present state $q(t)$ and present input $x(t)$, i.e

$$z(t) = \lambda(q(t), x(t))$$

where λ is called the output function. This generalised model is usually called Mealy machine.

If the output function $z(t)$ depends only on the present state and independent of the current input, the output function may be written as

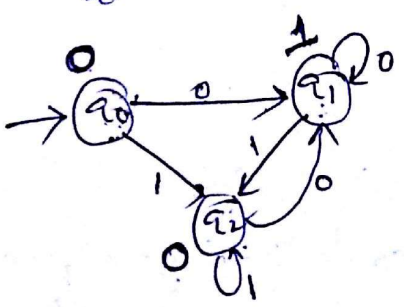
$$z(t) = \lambda(q(t))$$

This restricted model is called Moore machine.

Def: The Moore machine is a six-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where

(i) Q is a finite set of states;
(ii) Σ is the input alphabet;
(iii) Δ is the output alphabet;
(iv) δ is the transition function $Q \times \Sigma$ into Q
(v) λ is the output function mapping Q into Δ; and
(vi) q_0 is the initial state.

Ex:



i/p: 101010
o/p: 0 0 1 0 1 0

$q_0 \xrightarrow{1} q_2 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_1$

0 0 1 0 1 0

Def: A Mealy machine is a six-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where all the symbols except λ have the same meaning as in the Moore machine. λ is the output function mapping $Q \times \Sigma$ into Δ.

Ex: A Moore machine

present state	Next state δ $a=0$	$a=1$	output λ
$\rightarrow q_0$	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

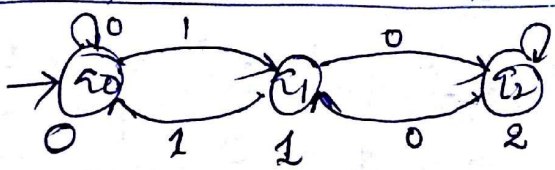
For the input string 0111, the transitions of states is given by $q_0 \xrightarrow{0} q_3 \xrightarrow{1} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2$.

The output string is 00010

A Mealy machine

present state	Next state $a=0$ state	$a=0$ output	$a=1$ state	$a=1$ output
$\rightarrow q_1$	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

(Ex)



I/P : 1000

$q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{0} q_2 \xrightarrow{0} q_2$

1 2 1 2

O/P : 1212

Equivalence of Moore and Mealy machines:

Let M be a Mealy (or) Moore machine. Define $T_M(w)$, for the input string w, to be the output produced by M on input w. There can never be exact identity b/w the functions T_M and $T_{M'}$ if M is a Mealy machine and M' a Moore machine, because $|T_M(w)|$ is one less than $|T_{M'}(w)|$ for each w.

→ However, we may neglect the response of a Moore machine to input ϵ and say that Mealy machine M_1 and Moore machine M' are equivalent if for all inputs w, $b\,T_M(w) = T_M(w)$, where b is the output of M' for its initial state.

(Moore to Mealy)

Theorem: If $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is a Moore machine, then there is a Mealy machine M_2 equivalent to M_1.

Proof: Let $M_2 = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$ and define $\lambda'(q,a)$ to be $\lambda(\delta(q,a))$ for all states 'q' and input symbols a.

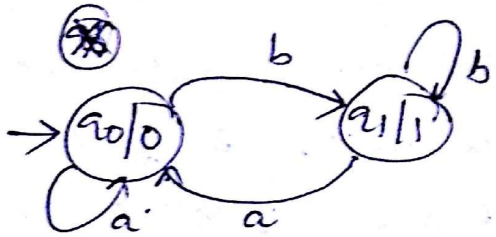
Then M_1 and M_2 enter the same sequence of states on the same input, and with each transition M_2 emits the output that M_1 associates with the state entered.

- Moore machine → n input symbols → n+1 outputs.
- Mealy machine → n input symbols → n outputs

(3) convert the following Moore machine into equivalent Mealy machine

$M = (\{q_0, q_1\}, \{a, b\}, \{0, 1\}, \delta, \lambda, q_0)$

present state	Next state a	Next state b	o/p
→ q_0	q_0	q_1	0
q_1	q_0	q_1	1



Sol: only o/p changes in the Mealy machine.

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

$\lambda'(q_0, a) = \lambda(\delta(q_0, a)) = \lambda(q_0) = 0$

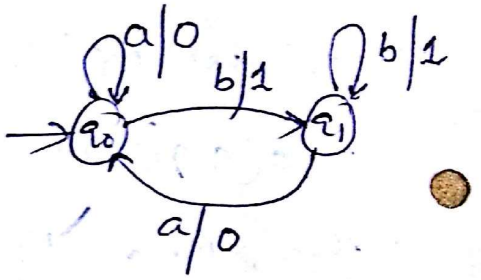
$\lambda'(q_0, b) = \lambda(\delta(q_0, b)) = \lambda(q_1) = 1$

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

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

Mealy machine is

present state	Next state a: state	a: o/p	Next state b: state	b: o/p
→ q_0	q_0	0	q_1	1
q_1	q_0	0	q_1	1



ex: i/p : ababa

(Moore) o/p : 001010 [$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0$]

0 0 1 0 1 0

(Mealy) o/p : 01010 [$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0$]

0 1 0 1 0

omit the first output in Moore machine and the remaining outputs in both the machines are same. So, they are equivalent.

(Mealy to Moore)

Theorem: Let $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ be a Mealy machine, then there is a Moore machine M_2 equivalent to M_1.

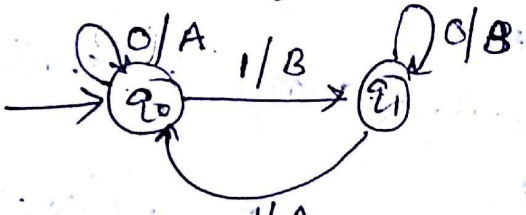
Proof: Let $M_2 = (Q \times \Delta, \Sigma, \Delta, \delta', \lambda', [q_0, b_0])$, where b_0 is an arbitrarily selected member of Δ. That is, the states of M_2 are pairs $[q, b]$ consisting of state of M_1 and an o/p symbol. Define $\delta'([q,b], a) = [\delta(q,a), \lambda(q,a)]$ and

$$\lambda'([q,b]) = b.$$

- The second component of the state $[q,b]$ of M_2 is the o/p made by M_1 on some transition into state q. Only the first components of M_2's states determine the moves made by M_2.

An easy induction on n shows that if M_1 enters states $q_0, q_1, \ldots q_n$ on input $a_1, a_2 \ldots a_n$ and emits o/ps $b_1, b_2 \ldots b_n$. Then M_2 enters states $[q_0, b_0], [q_1, b_1] \ldots [q_n, b_n]$ and emits outputs $b_0, b_1, \ldots b_n$.

(Q) Convert the following Mealy machine into equivalent Moore machine



Sol:

present state	Next state, $a=0$	o/p	Next state, $a=1$	o/p
$\rightarrow q_0$	q_0	A	q_1	B
q_1	q_1	B	q_0	A

The states in the equivalent Moore are $Q \times \Delta$ (E=A, O=A)

i.e (q_0, A) (q_0, B) (q_1, A) (q_1, B) / $\{q_0, q_1\} \times \{A, B\}$

~~$\delta'([q_0 q_1], \cdot$~~ $\{q_0, q_1\} \times \{A, B\}$

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

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

$\Rightarrow \delta'([q_0, A], 1) = [\delta(q_0, 1), \lambda(q_0, 1)]$ $\lambda'([q_0, A]) = A$

$= (q_1, B)$

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

$= [[q_0, A]$

$\rightarrow \delta'[(q_0, B], 1) = [\delta([q_0, A], \lambda(q_0, 1)]$ $\lambda'([q_0, B]) = B$

$= [q_1, B]$

$\rightarrow \delta'([q_1, A], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$ $\lambda'(q_1, A) = A$

$= (q_1, B)$

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

$= ([q_0, A]$

$\delta([q_1, B], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$ $\lambda'([q_1, B]) = B$

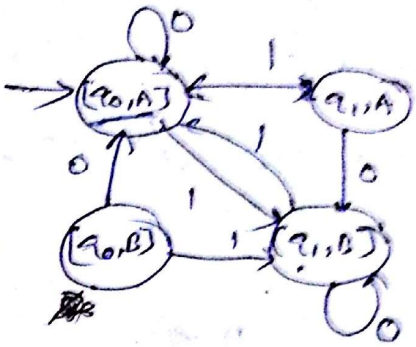
$= [q_1, B]$

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

$= [[q_0, A]]$

Moore machine

present state	Next state q=0	Next state q=1	O/P
→ [q0,A]	[q0,A]	[q1,B]	A
[q0,B]	[q0,A]	[q1,B]	B
[q1,A]	[q1,B]	[q0,A]	A
[q1,B]	[q1,B]	[q0,A]	B



q0/A

I/P: 00110

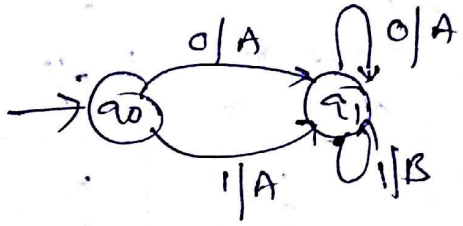
(Mealy) O/P: AABAA

$[q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_0]$

A A B A A

(Moore) O/P: A AABAA

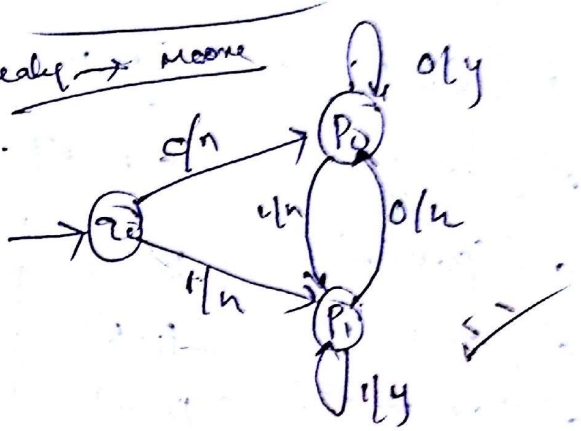
(Q) Mealy → Moore



Moore to Mealy

present state	Next state q=0	Next state q=1	O/P
→ q0	q0	q1	1
q1	q0	q2	0
q2	q2	q2	1
q3	q1	q1	0

Mealy → Moore



present state	Next state q=0 state, O/P		Next state q=1 state, O/P	
→ q0	P0	n	P1	n
P0	P0	y	P1	n
P1	P0	n	P1	y

for Moore machine

$\delta'([q,k],a) = [\delta(q,a), \lambda(q,a)]$ and $\lambda'([q,k]) = b$

① → $\delta'([q_0,n],0) = [\delta(q_0,0), \lambda(q_0,0)]$ $\lambda'([q_0,n]) = n$

$= [p_0,n]$

$\delta'([q_0,n],1) = [p_1,n]$ $\lambda'[(q_0,n)] = n$

$\delta'([p_0,n],0) = [p_0,y]$ $\lambda'[(p_0,y)] = n$

$\delta'([p_0,n],1) = [p_1,n]$ $\lambda'[(p_0,n)] = n$

$\delta'([p_1,n],0) = [p_0,n]$ $\lambda'[[p_1,n]] = n$

$\delta'[[p_1,n],1) = [p_1,y]$ $\lambda'([p_1,n] = n$

$\delta'([q_0,y],0) = [p_0,n]$ $\lambda'[[q_0,y]] = y$

$\delta'([q_0,y],1) = [p_1,n]$ $\lambda'([q_0,y] = y$

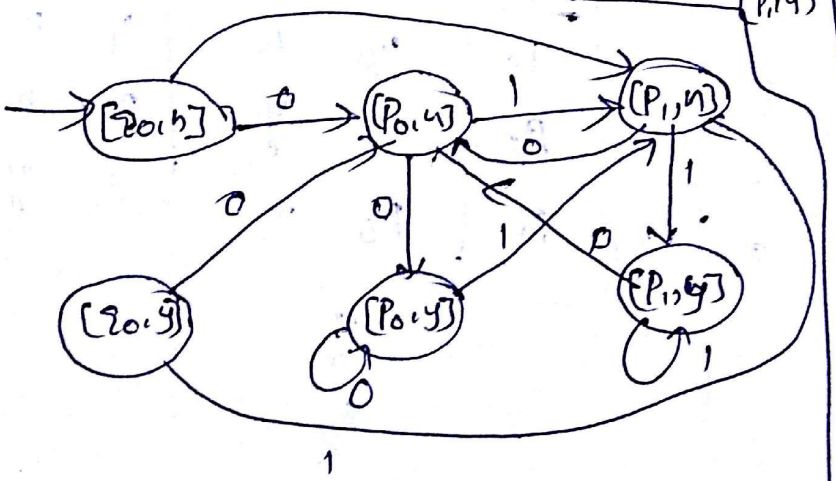
$\delta'([p_0,y],0) = [p_0,y]$

$\delta'([p_0,y],1) = [p_1,n]$

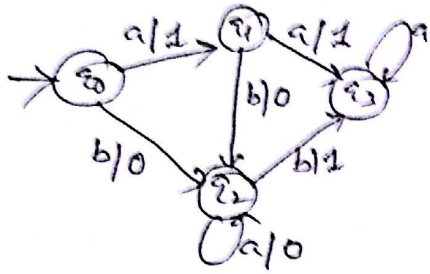
$\delta'([p_1,y],0) = [p_0,n]$

$\delta'([p_1,y],1) = [p_1,y]$

present state	Next state $q=0$	$q=1$	o/p
→ $[q_0,n]$	$[p_0,n]$	$[p_1,n]$	n
$[q_0,y]$	$[p_1,n]$	$[p_1,n]$	y
$[p_0,n]$	$[p_0,y]$	$[p_1,n]$	n
$[p_0,y]$	$[p_0,y]$	$[p_1,n]$	y
$[p_1,n]$	$[p_0,n]$	$[p_1,y]$	n
$[p_1,y]$	$[q_0,n]$	$[p_1,y]$	y



(4) Construct the Moore machine for the given Mealy machine. (2)



Present state	Input = a		Input = b	
	Next state	o/p	Next state	o/p
q_0	q_1	1	q_2	0
q_1	q_3	1	q_2	0
q_2	q_2	0	q_3	1
q_3	q_3	1	q_3	1

Sol:

The states in Moore machine are

$Q \times \Delta = \{q_0, q_1, q_2, q_3\} \times \{0,1\}$

$= [q_0,0], [q_0,1], [q_1,0], [q_1,1], [q_2,0], [q_2,1], [q_3,0], [q_3,1]$

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

$\delta'([q_0,0],b) = [q_2,0]$ $\lambda'([q_0,0]) = 0$

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

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

ab
1 0

$\delta'([q_1,0],a) = [q_3,1]$ $\lambda'(q_1,0) = 0$

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

$\delta'([q_1,1],a) = [q_3,1]$ $\lambda'(q_1,1) = 1$

$\delta'([q_1,1],b) = [q_2,0]$

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

$\delta'([q_2,0],b) = [q_3,1]$

$\delta'([q_2,1],a) = [q_2,0]$ $\lambda'[q_2,1] = 1$

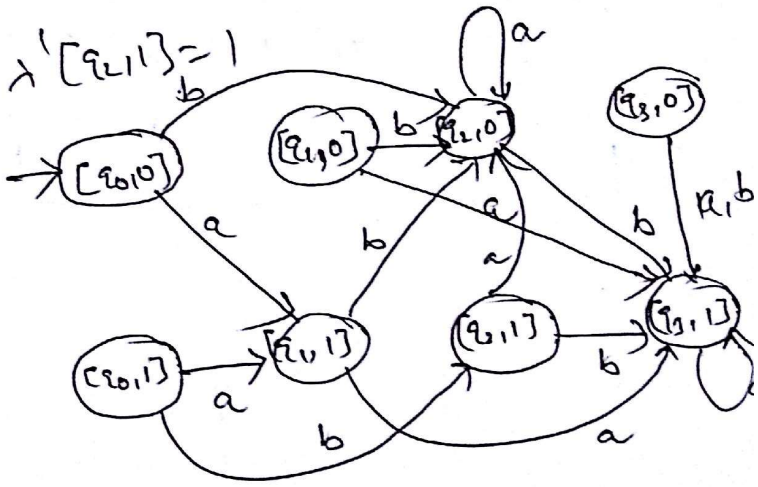
$\delta'([q_2,1],b) = [q_3,1]$

$\delta'([q_3,0],a) = [q_3,1]$

$\delta'([q_3,0],b) = [q_3,1]$

$\delta'([q_3,1],a) = [q_3,1]$

$\delta'([q_3,1],b) = [q_3,1]$



Moore machine

Equivalence of two finite automata :-

→ Two finite automata over Σ are equivalent if they accept the same set of strings over Σ. When two FAs are not equivalent, there is some string w over Σ satisfying the following :

one automaton reaches a final state on application of w, where as the other automaton reaches a nonfinal state.

Comparison method :-

Let M and M' be two FAs over Σ. We construct comparison table consisting of $n+1$ columns, where n is the no of i/p symbols

→ The comparison table is constructed by starting with the pair of initial vertices q_{in}, q'_{in} of M and M' in the first column. The first elements in the subsequent columns are (q_a, q'_a), where q_a and q'_a are reachable by a-paths from q_{in} and q'_{in}. We repeat construction by considering the pairs in the second and subsequent columns which are not in the first column.

The row-wise construction is repeated. There are two cases:

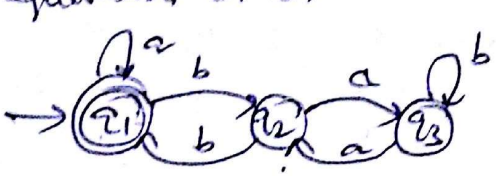
case 1 : If we reach a pair (q, q') such that q is a final state of M, and q' is a nonfinal state of M' (or vice versa), we terminate the construction and conclude that **M and M' are not equivalent.**

case 2 : Here the construction is terminated when no new element appears in the second and subsequent columns which are not in the first column. In this case we conclude that **M and M' are equivalent.**

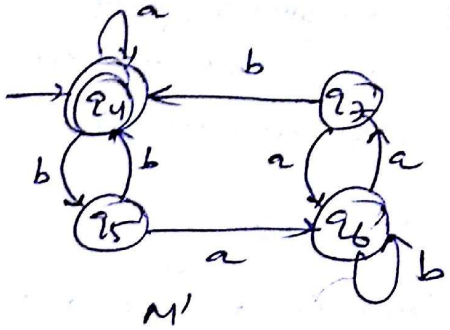
(7)

Ex:

(a) Find out whether the following two DFAs M and M' are equivalent or not.



M



M'

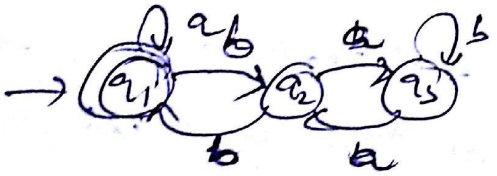
Sol: (q_1, q_4) Initial states

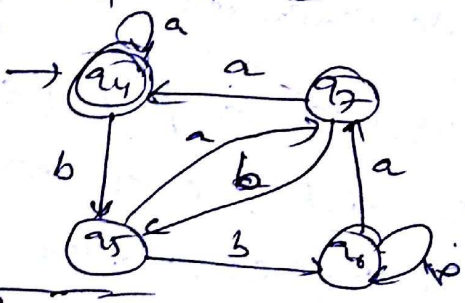
Comparison table

(q, q')	(q_a, q_a')	(q_b, q_b')
{Initial states of M and M'} (q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_5)	(q_3, q_6)	(q_1, q_4)
(q_3, q_6)	(q_2, q_5)	(q_3, q_6)
(q_2, q_3)	(q_3, q_6)	(q_1, q_4)

∴ we do not get a pair (q, q'), where q is a final state and q' is a non-final state at every row, we proceed until all the elements in the second and third columns are also in the first column. Therefore, M and M' are equivalent.

(b) Show that the automata M_1 and M_2 are not equivalent



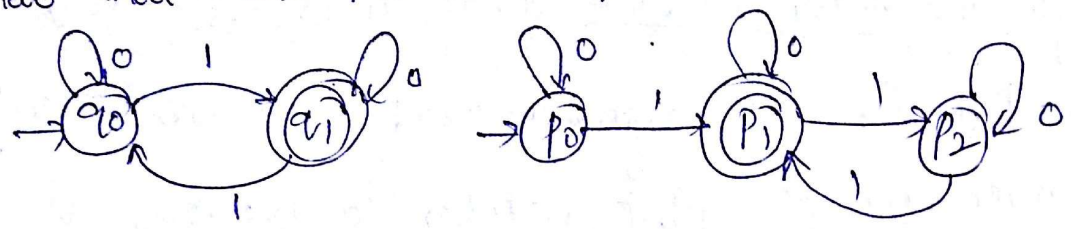


Comparison table

(q, q')	(q_a, q_a')	(q_b, q_b')
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_5)	(q_3, q_7)	(q_1, q_5)
(q_1, q_5)		

∴ Two FAs are not equivalent.

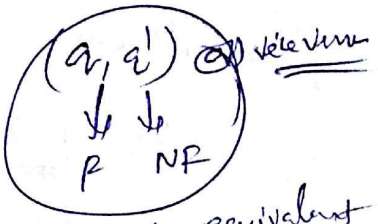
Q) Show that the FA are equivalent as shown in figure



① ②

comparison table

(q, q')	(q_0, q_0') Input	(q_1, q_1')
(q_0, P_0)	(q_0, P_0)	(q_1, P_1)
(q_1, P_1)	(q_1, P_1)	(q_0, P_2)
(q_0, P_2)	(q_0, P_2)	(q_1, P_1)



Not equivalent

∴ we donot get a pair (q, q'), where 'q' is a final state and q' is a non-final state.

So, the above two FA's are equivalent.

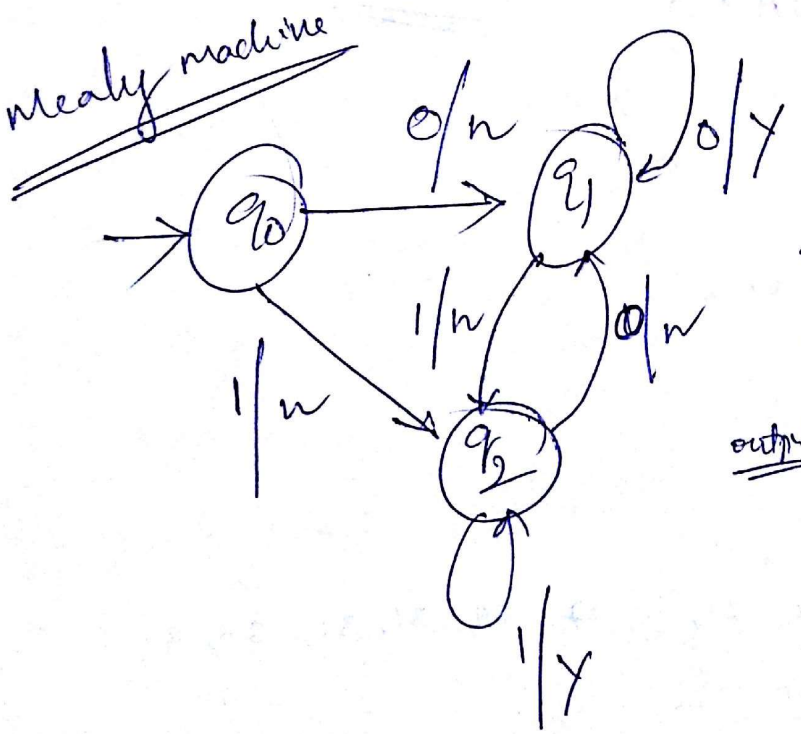
(Q) Design a Mealy machine that uses its state to remember the last symbol read and emits output 'y' whenever current input matches to previous one, and emits 'n' otherwise.

Sol. Assume input alphabet $\Sigma = \{0,1\}$

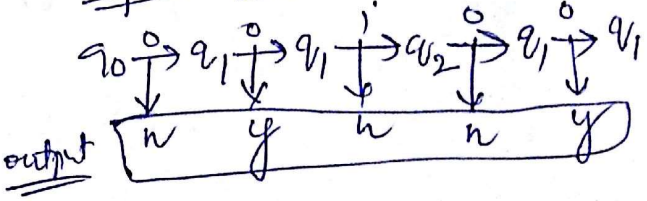
If 00 is input then 'y' is the output
11 " " 'y'
01 " " 'n'
10 " " n

0 → n
1 → n

Mealy machine



Input : 00100



Minimization of FSM :-

Indistinguishable states :

Two states 'p' and 'q' of a DFA are called indistinguishable if $\delta(p,w) \in F$ implies $\delta(q,w) \in F$ and $\delta(p,w) \notin F$ implies $\delta(q,w) \notin F$ for all $w \in \Sigma^*$.

Distinguishable states :

There exists some string $w \in \Sigma^*$ such that $\delta(p,w) \in F$ and $\delta(q,w) \notin F$ (or) vice versa.

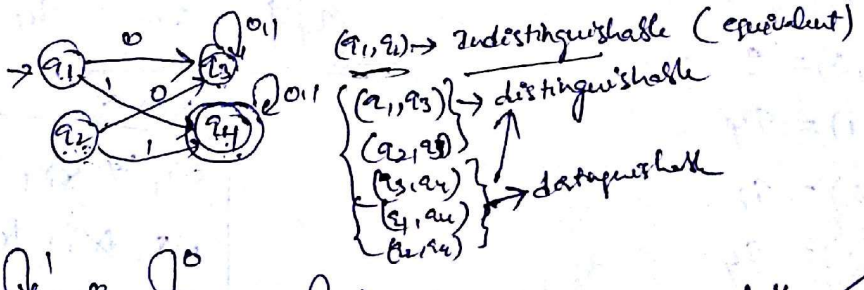
Equivalence :

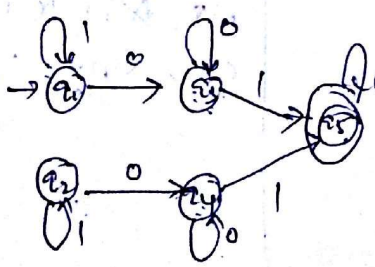
Two states q_1 and q_2 are equivalent $(q_1 \equiv q_2)$ if both $\delta(q_1,x)$ and $\delta(q_2,x)$ are final states (or) both of them are non-final states for all $x \in \Sigma^*$.

K-Equivalence :-

Two states q_1 and q_2 are k-equivalent $(k \geq 0)$ if both $\delta(q_1,x)$ and $\delta(q_2,x)$ are final states (or) both non-final states for all strings of length k (or) less.

Any two final states are 0-equivalent and any non-final states are also 0-equivalent.





(q_1,q_2) are indistinguishable ✓

~~[Step1: Remove all inaccessible states. [If any state is not reachable from starting state that state is inaccessible]]~~

Step 1: Remove all inaccessible states [If any state is not reachable from starting state]

Step 2: Mark all pairs which marking combinations with final state as distinguishable.

[Consider pair (p,q), if $p \in F$ and $q \notin F$ (or) vice versa then mark (p,q) as distinguishable].

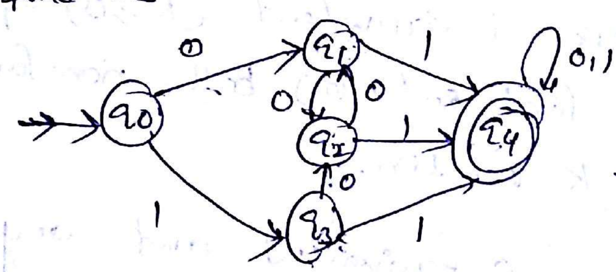
Step 3: Repeat this step until no previously unmarked pairs are marked.

[For all pairs (p,q) and all $a \in \Sigma$, compute $\delta(p,a) = r$ and $\delta(q,a) = s$, if the pair (r,s) is marked as distinguishable, then mark (p,q) also as distinguishable]

After applying above steps for all the pairs remaining pairs are indistinguishable. [$\delta(p,a) \in F$ implies $\delta(q,a) \in F$]

→ All indistinguishable pairs are merged and used with single state

Q) Find the minimum state automata for the following DFA



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

Sol:

$\delta(q_0,0) = q_1$

$\delta(q_0,1) = q_3$

$\delta(q_1,0) = q_2$

$\delta(q_1,1) = q_4$

$\delta(q_2,0) = q_1$

$\delta(q_2,1) = q_4$

$\delta(q_3,0) = q_2$

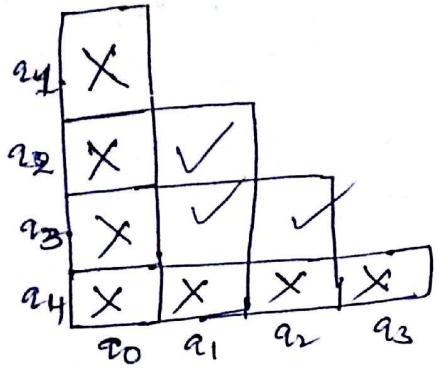
$\delta(q_3,1) = q_4$

$\delta(q_4,0) = q_4$

$\delta(q_4,1) = q_4$

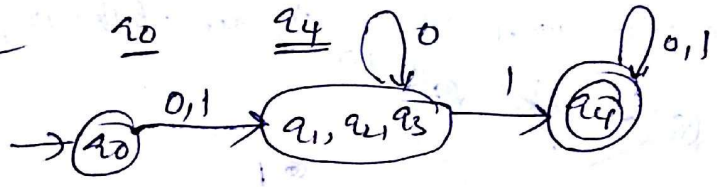
10/02/08

65, ~~88~~ 88, 89, 96, ~~96~~,
A8, A9, B0, B1, B2, B9,
CO, SA4, le92

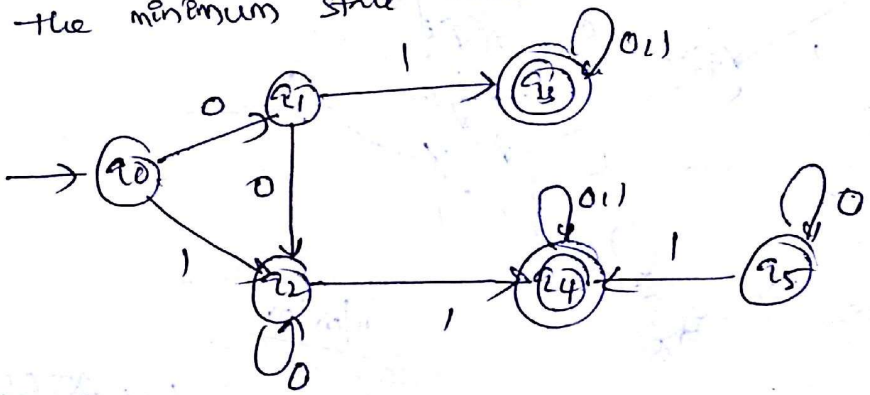


$(q_0, q_1) \rightarrow ① \rightarrow (q_1, q_4)$

States $\begin{cases} (q_1, q_2) & (q_2, q_3) & (q_1, q_3) \Rightarrow (q_1, q_2, q_3) \\ q_0 & q_4 \end{cases}$



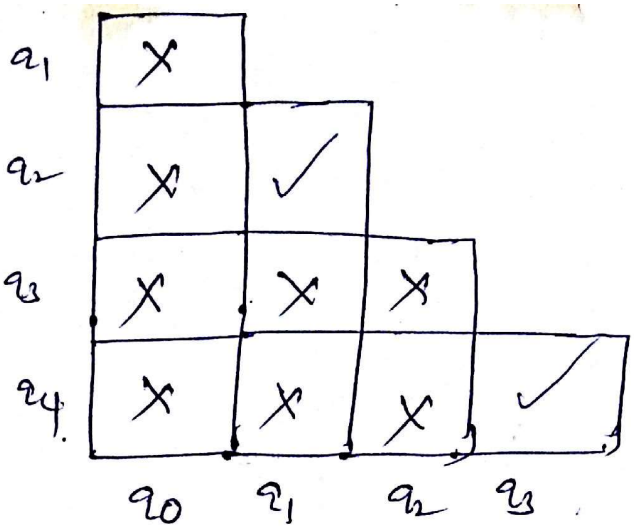
(4) Find the minimum state automata for the following DFA

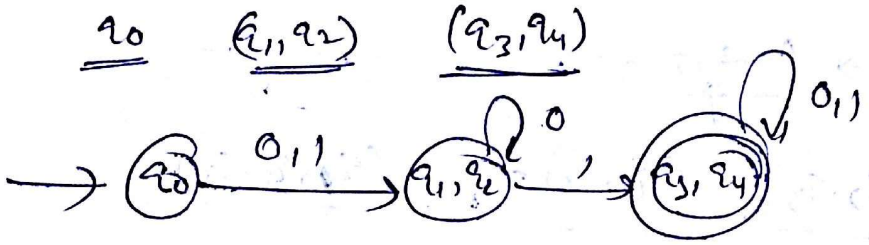


Sol:

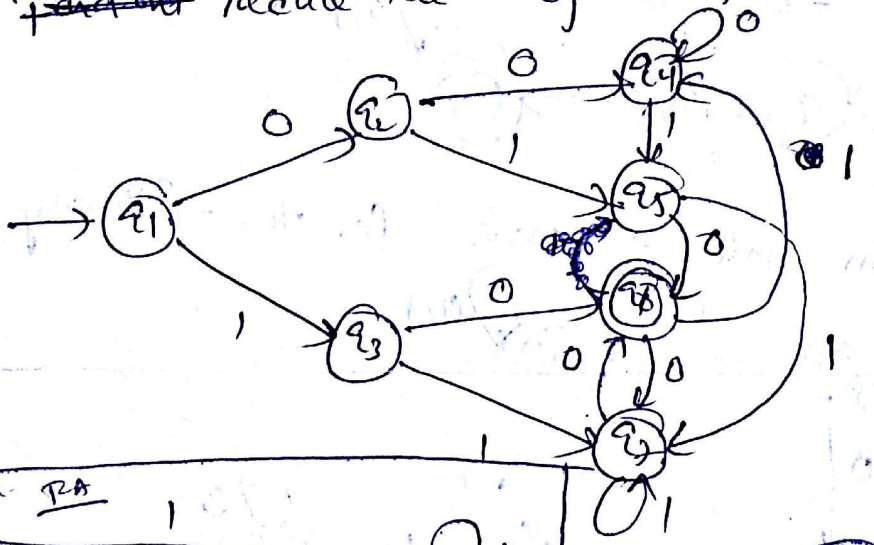
States	Inputs 0	1
q_0	q_1	q_2
q_1	q_2	q_3
q_2	q_2	q_4
q_3	q_3	q_3
q_4	q_4	q_4
q_5	q_5	q_0

Remove q_5 (inaccessible state) (13)

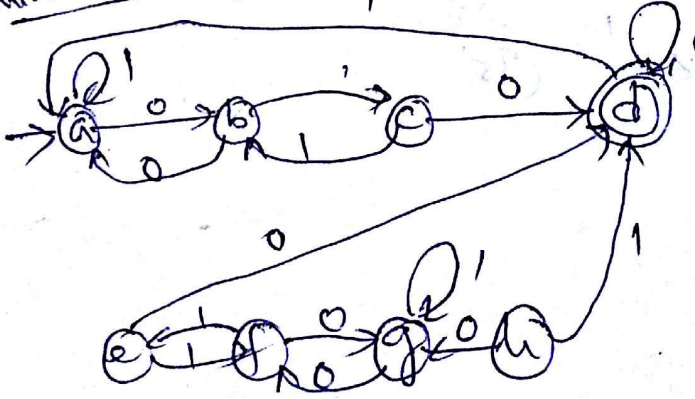




(9) ~~Find out~~ Reduce the no of states in the following DFA

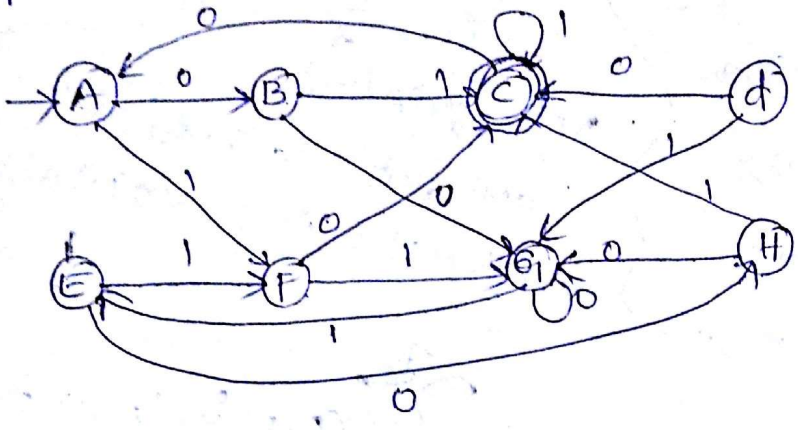


Minimize the DFA



11/01/01

(8) Find the minimum state automata for the following FA.



B	X						
C	X	X					
D	X	X	X				
E	✓	X	X	X			
F	X	X	X	✓	X		
G1	X	X	X	X	X	X	
H	X	✓	X	X	X	X	X
	A	B	C	D	E	F	G1

(NF, F) ⇒ (A,C), (B,C), (D,C), (E,C), (F,C), (G1,C), (H,C)

Not equivalent.

Again start with (A,B) Mark the pairs as 'X'

(A,B) $\xrightarrow{1}$ (F,C) ⇒ X

NF F

(A,D) $\xrightarrow{0}$ (B,C) ⇒ X

(A,E) $\xrightarrow{0}$ (B,H)
(A,E) $\xrightarrow{1}$ (F,F)

we should know about (B,H)

(A,F) $\xrightarrow{0}$ (B,C) ⇒ X

(A,G1) $\xrightarrow{0}$ (B,G1)
(A,G1) $\xrightarrow{1}$ (F,E)

we should know about (B,G1) & (F,E)

$(A,H) \xrightarrow{1} (F,C) \Rightarrow \times$

$(B,D) \xrightarrow{1} (C,G) \Rightarrow \times$

$(B,E) \xrightarrow{1} (C,F) \Rightarrow \times$

$(B,F) \xrightarrow{0} (G,C) \Rightarrow \times$

$(B,G) \xrightarrow{1} (C,E) \Rightarrow \times$

Not equivalent.

ONE state is a Final state

Another state is a Non Final state

$(B,H) \xrightarrow{0} (G,G)$

$(B,H) \xrightarrow{1} (C,C)$

equivalent ✓

$(D,E) \xrightarrow{0} (C,H) \Rightarrow \times$

$(D,F) \xrightarrow{0} (C,C)$

$(D,F) \xrightarrow{1} (G,G)$

equivalent ✓

$(D,G) \xrightarrow{0} (C,G) \rightarrow \times$

$(D,H) \xrightarrow{0} (C,G) \Rightarrow \times$

$(E,F) \xrightarrow{0} (H,C) \Rightarrow \times$

$(E,G) \xrightarrow{0} (H,G) \Rightarrow \times$

$(E,H) \xrightarrow{1} (F,C) \Rightarrow \times$

$(F,G) \xrightarrow{0} (C,G) \Rightarrow \times$

$(F,H) \xrightarrow{0} (C,G) \Rightarrow \times$

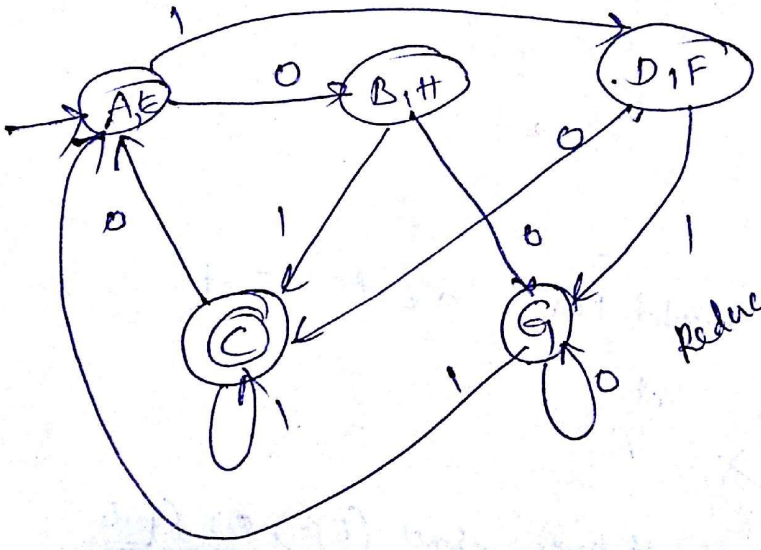
$(G,H) \xrightarrow{1} (E,C) \Rightarrow \times$

Since (B,H) are equivalent, (A,E) are equivalent

since (B,G) are Not equivalent (A,G) are Not equivalent

The states in the minimized FA are

(A,E), (B,H), (D,F), C, G



Reduced F.A with '5' states

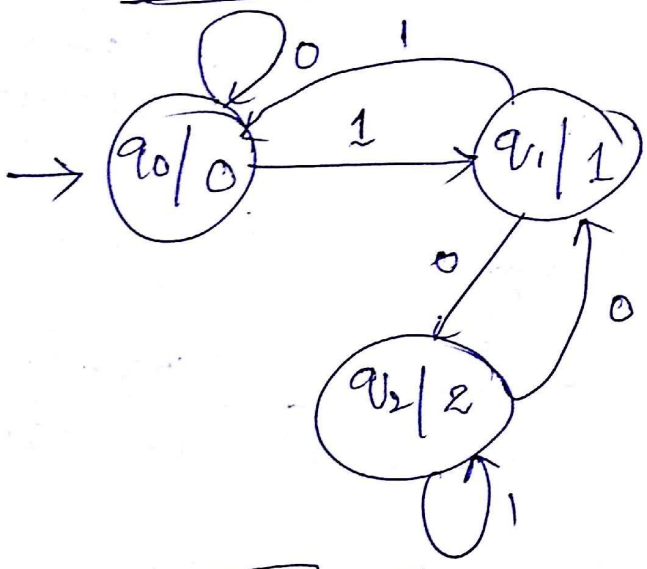
(8) Construct a Moore machine to determine the residue mod 3 for each binary string treated as a binary integer. Inputs $= \Sigma = \{0,1\}$

Sol: Here we are getting three outputs i.e 0, 1 and 2. [for any integer only one output]

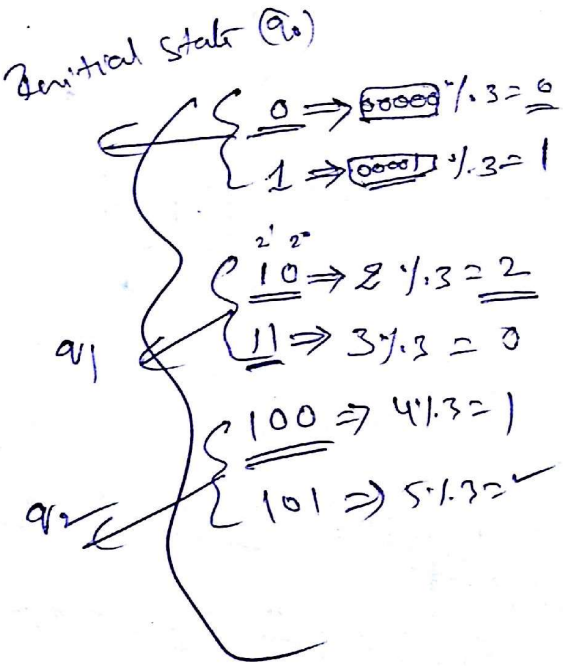
$\Delta = \{0,1,2\}$

So, In moore machine output depends on the present state, we can take three states.

q_0, q_1, and q_2.



0 % 3 = 0
1 % 3 = 1
2 % 3 = 2
3 % 3 = 0
4 % 3 = 1
5 % 3 = 2
6 % 3 = 0
⋮ ⋮



Input: 00101 = 5

$q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_2$

$q_2 \rightarrow$ 2 → output

5 % 3 = 2 → output

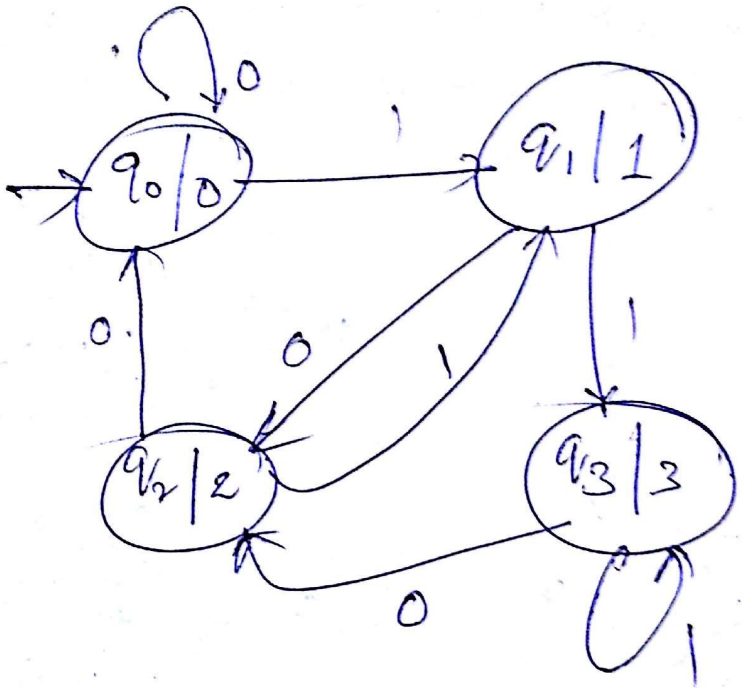
(At the last state we will get the output for any input string)

(9) Construct a Moore Machine to determine the residue mod 4 for each binary string treated as a binary integer.

0%4 = 0
1%4 = 1
2%4 = 2
3%4 = 3
4%4 = 0
5%4 = 1
6%4 = 2
7%4 = 3
8%4 = 0

Sol: Input alphabet $\Sigma = \{0,1\}$

output alphabet $\Delta = \{0,1,2,3\}$



q_0 { 0 ⇒ 0%4 = 0, 1 ⇒ 1%4 = 1

q_1 { 10 ⇒ 2%4 = 2, 11 ⇒ 3%4 = 3

q_2 { 100 ⇒ 4%4 = 0, 101 ⇒ 5%4 = 1

q_3 { 110 ⇒ 6%4 = 2, 111 ⇒ 7%4 = 3

Input: 00110 ⇒ 6

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

⇓

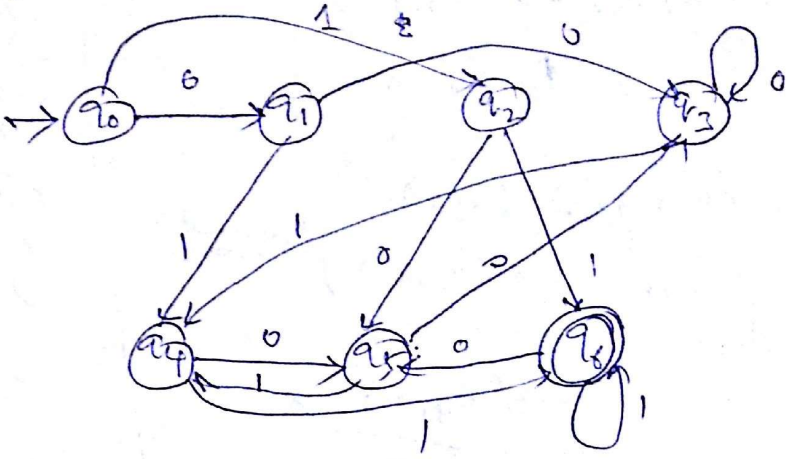
(2) output

6%4 = 2 → output

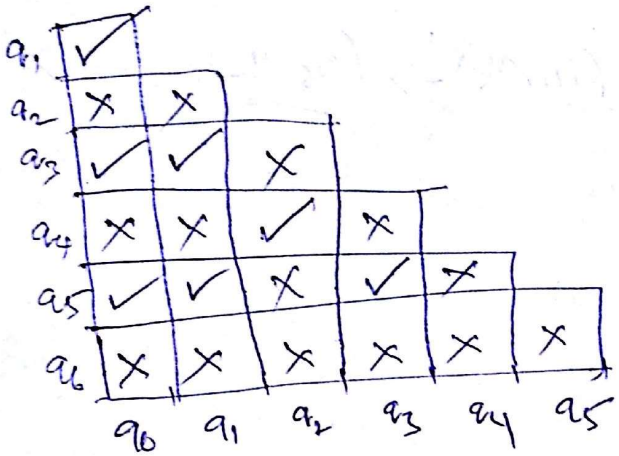
27/1

Q) Minimize the FA given below and show both given and reduced are equivalent.

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



Sol:



- $(\underbrace{q_0, q_1, q_2, q_3, q_4, q_5}_{\text{N.F}}, \underbrace{q_6}_{\text{F}})$ → Not equivalent

① $\{(q_0, q_1) \xrightarrow{0} (q_1, q_3)\}$
$\{(q_0, q_1) \xrightarrow{1} (q_2, q_4)\}$

② $\{(q_0, q_2) \xrightarrow{0} (q_1, q_5)\}$
$\{(q_0, q_2) \xrightarrow{1} (q_2, q_6)\}$ ⇒ (q_0, q_2) ✗
NF F

$\{(q_0, q_3) \xrightarrow{0} (q_1, q_3)\}$
$\{(q_0, q_3) \xrightarrow{1} (q_2, q_4)\}$

$(q_0, q_4) \xrightarrow{1} (q_2, q_6)$ ⇒ (q_0, q_4) are not equivalent.
NF F

$(q_0, q_5) \xrightarrow{0} (q_1, q_3)$ ✓
$(q_0, q_5) \xrightarrow{1} (q_2, q_4)$

$(q_1, q_2) \xrightarrow{0/1} (q_4, q_6)$ ×
NT P

$(q_1, q_3) \xrightarrow{0} (q_3, q_3)$ equivalent
$(q_1, q_3) \xrightarrow{1} (q_4, q_4)$

$(q_1, q_4) \xrightarrow{1} (q_4, q_6)$ ×

$(q_1, q_5) \xrightarrow{0} (q_3, q_4)$ equivalent
$(q_1, q_5) \xrightarrow{1} (q_4, q_4)$

$(q_2, q_3) \xrightarrow{1} (q_6, q_4)$ ×
F NP

$(q_2, q_4) \xrightarrow{0} (q_5, q_5)$ equivalent
$(q_2, q_4) \xrightarrow{1} (q_6, q_6)$

$(q_2, q_5) \xrightarrow{0} (q_5, q_3)$
$(q_2, q_5) \xrightarrow{1} (q_6, q_4) \rightarrow$ ×

$(q_3, q_4) \xrightarrow{1} (q_3, q_6)$ ×

$(q_3, q_5) \xrightarrow{0} (q_3, q_3)$ equivalent
$(q_3, q_5) \xrightarrow{1} (q_4, q_4)$

$(q_4, q_5) \xrightarrow{1} (q_6, q_4) \rightarrow$ ×

Here (q_0, q_4) (q_0, q_3)

Since (q_1, q_3) and (q_2, q_4) are equivalent

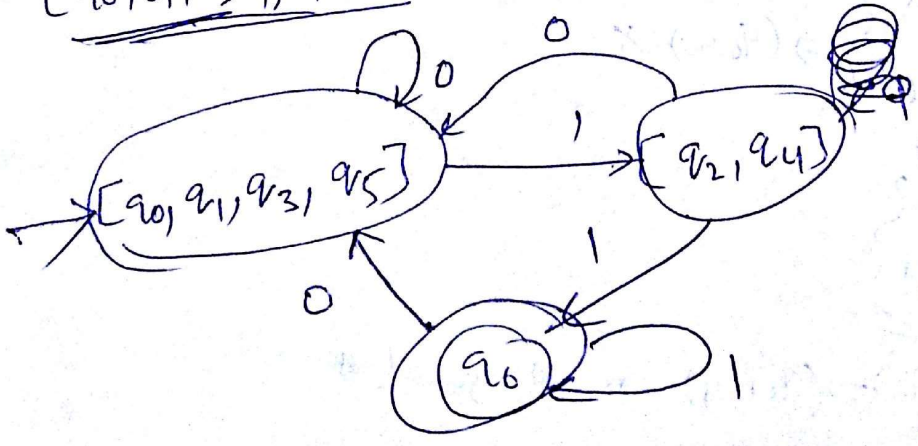
(q_0, q_1) equivalent
(q_0, q_3) equivalent

Now, the states in reduced FA are

$[q_0, q_3]$, $[q_0, q_3]$, $[q_0, q_5]$ $[q_1, q_3]$ $[q_1, q_5]$ $[q_2, q_4]$ $[q_3, q_5]$

$[q_0 q_1 q_3 q_5]$ $[q_2, q_4]$ q_6

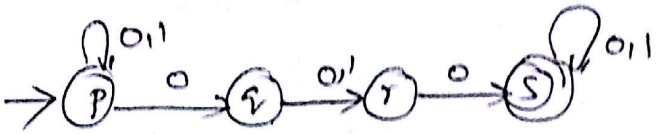
F.A



Tutorial-I

Q) Construct DFA equivalent to NFA $M = (\{p,q,r,s\}, \{0,1\}, \delta, p, \{s\})$ and δ is given by

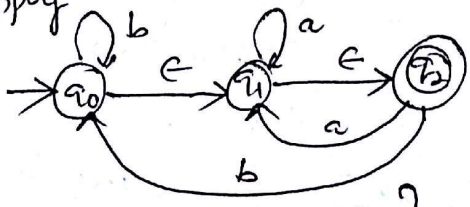
States	Inputs 0	1
→p	p,q	p
q	r	r
r	s	—
(s)	s	s



Sol:

States	Inputs 0	1
→ [p]	[p,q]	[p]
[p,q]	[p,q,r]	[p,r]
[p,q,r]	[p,q,r,s]	[p,r]
[p,r]	[p,q,s]	[p]
[p,q,r,s]	[p,q,r,s]	[p,r,s]
[p,q,s]	[p,q,s]	[p,r,s]
[p,r,s]	[p,q,s]	[p,s]
[p,s]	[p,q,s]	[p,s]

(Q) Explain the procedure to convert NFA-ε moves to ordinary NFA, and apply the same for the following diagram.



Sol:

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

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

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

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

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

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

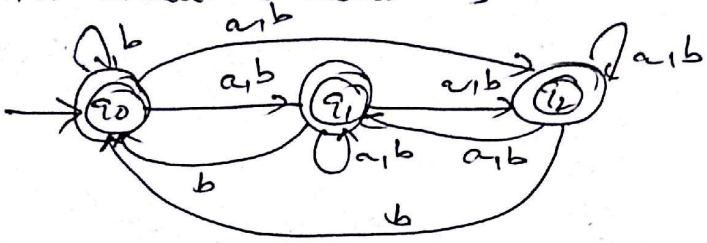
$\hat{\delta}(q_1, a) = \epsilon\text{-closure}(\delta(\{q_1, q_2\}, a)) = \epsilon\text{-closure}(q_1) = \{q_1, q_2\}$

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

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

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

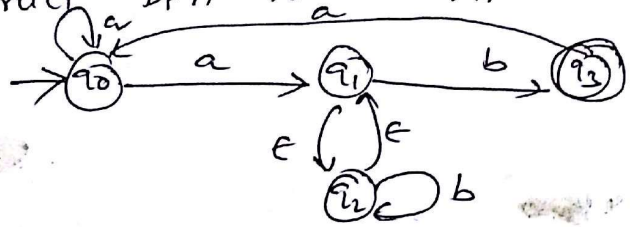
NFA without ε-moves is



q_0, q_1, q_2 are final states

ε-closure (q_0), (q_1), (q_2) } contains final state

(a) construct DFA for the NFA-ε moves shown below:



NFA with ε-moves
⇓
NFA without ε-moves
⇓
DFA

Sol:

ε-CLOSURE $(q_0) = \{q_0\}$

ε-CLOSURE $(q_1) = \{q_1, q_2\}$

ε-CLOSURE $(q_2) = \{q_2, q_1\}$

$\Rightarrow$ ε-CLOSURE $(q_3) = \{q_3\}$

$\hat{\delta}(q_0, a) =$ ε-CLOSURE $(\delta(q_0, a)) =$ ε-CLOSURE $(q_0, q_1) = \{q_0, q_1, q_2\}$

$\hat{\delta}(q_0, b) =$ " $(\delta(\{q_0, q_1, q_2\}, b)) =$ " $(\{q_2, q_3\}) =$ $\{q_1, q_2, q_3\}$

$\hat{\delta}(q_0, a) =$ " $(\delta(\{q_1, q_2, q_3\}, a)) =$ " $(\phi) = \phi$

$\hat{\delta}(q_1, b) =$ " $(\delta(\{q_1, q_2\}, b)) =$ " $(q_2, q_3) = \{q_1, q_2, q_3\}$

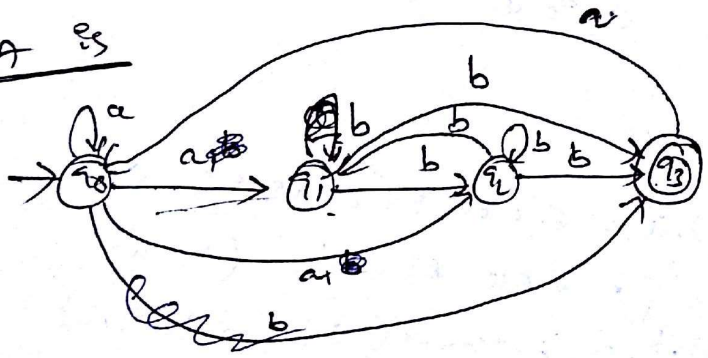
$\hat{\delta}(q_2, a) =$ " $(\delta(\{q_1, q_2\}, a) =$ " $(\phi) = \phi$

$\hat{\delta}(q_2, b) =$ " $(\delta(\{q_1, q_2\}, b] =$ " $(q_2, q_3) = \{q_1, q_2, q_3\}$

$\hat{\delta}(q_3, a) =$ " $(\delta(q_3, a) =$ " $(q_0) = \{q_0\}$

$\hat{\delta}(q_3, b) =$ " $(\delta(q_3, b)) =$ " $(\phi) = \phi$

NFA is



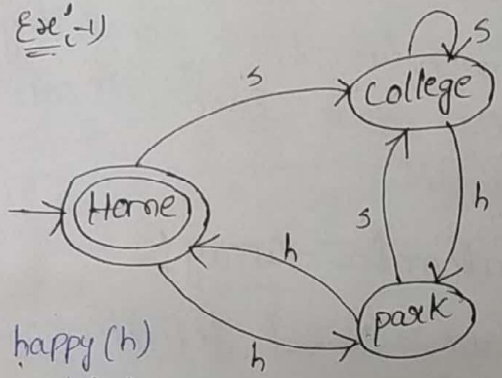
DFA is

State	Inputs a	b
$\rightarrow [q_0]$	$[q_0, q_1, q_2]$	$[q_1, q_2, q_3]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_1, q_2, q_3]$
$[q_1, q_2, q_3]$		

UNIT - I

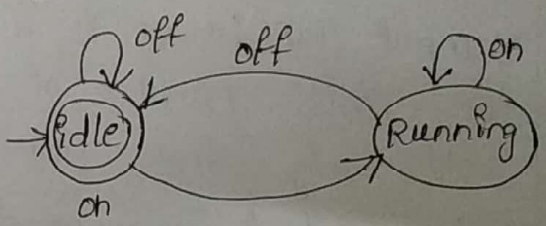
* Introduction to finite Automata:- An Automata with a finite no. of states is called finite Automata
- the finite Automata (or) finite state machine is an abstract machine that has five tuples or elements
- It has a set of states and rules for moving from one state to another. but it depends upon the applied input symbol.

Ex:-1)

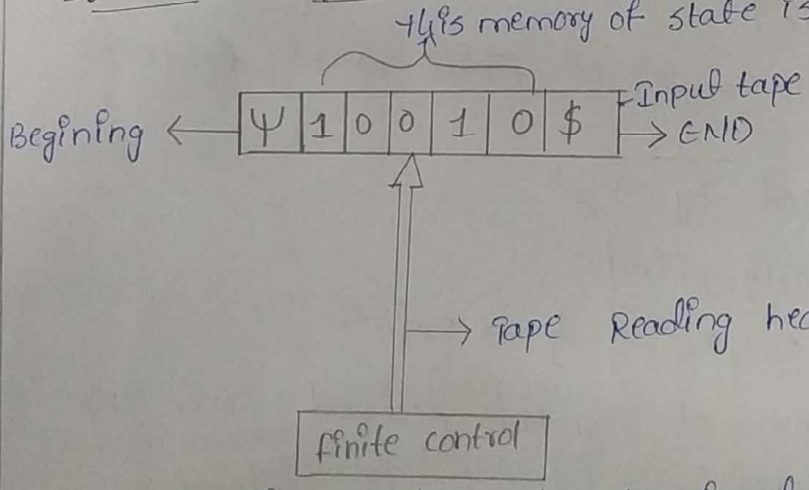


happy (h)
sad (s)

ex:-2)



* Structural Representation of finite Automata:-



→ Tape Reading head reads the input tape from left to Right.
→ "$" represents End of the string
→ "Ψ" represents at begining
→ It has 3 elements

1) finite controller
2) Tape Reader
3) Input Tape

1) finite controller:- It takes decision about which input symbol to read next after receiving a particular input

2) Tape Reader:- It performs the read operation cell-by-cell and reads only one input symbol (from each cell) at a given instance

3) Input Tape:- It is a sequential tape consisting of finite number of cells where each cell can hold only single input symbol.

* The central concepts of Automata theory:-

Symbol:- symbol does not have formal definition like dot (.) in mathematics

Ex:- A, B, 3, 9

Alphabet:- It is a set of finite no. of alphabets (or) symbols

→ It is denoted by "Σ"

Ex:- $\Sigma = \{0,1\}$ → finite

$\Sigma = \{0,1,2,---\}$ → Infinite

String:- collection of symbols decived from the single alphabet

→ It is denoted by (w/s)

Ex:- $w = 0110$

$w = ABBC$

Length of string:- the sum of occurance of symbols in a string is called "length of string"

→ It is denoted by (|w|)

Exi:- 1) w=0110 ex:-2) w=ϵ

|w| =4 |w|=0

concatenation of string:- the addition of 2 strings must be added is called their concatenation

Exi:- |w1| =n

|w2| =m

|w1,w2| =n+m

Null string:- It is denoted by (ϵ) (or) empty string

→ that means no symbols in a string

length of null string |ϵ| =0

prefise:- leading symbols of the string

→ Any length of string from left hand side (LHS) without skipping symbols in middle

Exi:- 1) w=make

|w| =4

prefixes:-1) ϵ

m

m a

m a k

m a k e

ex:-2) |w| =n

possible prefixes n+1

suffise:- Trailing symbols of string

→ Any length of string from right hand side (RHS) without skipping symbols in middle

Exi:-1) w =make

|w|=4

Suffix:- ε
e
ke
ake
make

Exe:-2) |w|=n
possible suffixes are n+1

substring:- It is a string or part of the string obtained by removing the suffix or a prefix or both from the given string

→ any length of string from any where without skipping symbols in middle

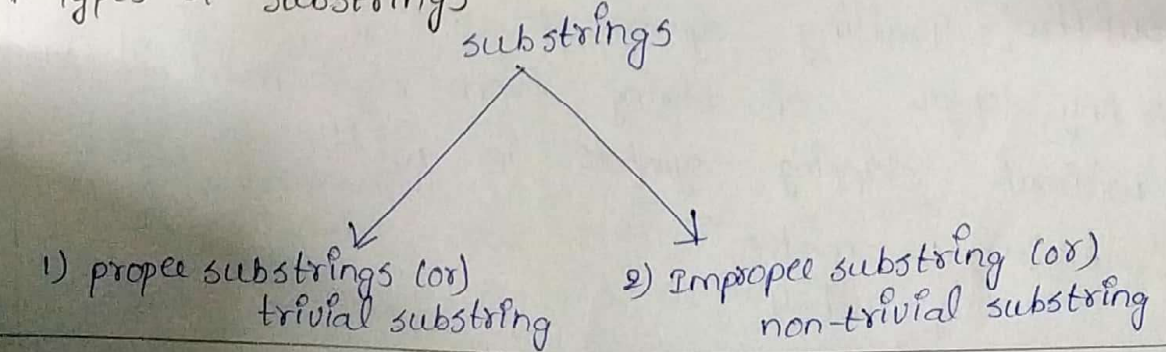
Exe:- w=make
|w|=4

substrings are (ε, m, a, k, e, ma, ak, ke, mak, ake, make)

substring are:- ε → 1
m, a, k, e → 4
ma, ak, ke → 3
mak, ake → 2
make → 1
} Σn

If |w|=n
substring = Σn+1

* Types of substrings



Ex:- w = make

{ ϵ, m, a, k, e, ma, mak, ak, ke, ake, make }

→ ϵ and make are improper substrings

→ Remaining strings are proper substrings

* language:- It is a collection of strings

→ It may be finite / Infinite

→ It is denoted by (L)

Ex:- L = { 0, 10, 110, 11 } ⇒ |L| = 4

L = { 0, 1, 00, 01, 10, 11, 000 ---} → infinite

L = { ϵ, 0, 11 } ⇒ |L| = 3

L = { ϵ } ⇒ |L| = 1

→ string is empty but not language

L = { } ⇒ |L| = 0 ⇒ ϕ

$L = \{\epsilon\} \neq \phi$ → empty language

↓ Empty string

* Kleene closure (*):-

Σ = {0, 1}

$L^* = \Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, ---\}$

* positive closure (+) / powers of Σ:-

Σ = {0, 1}

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

$\epsilon^+ = \{0, 1, 00, 01, 10, 11, ----\}$

$\Sigma^0 = \{\epsilon\}$ = set of all strings of length '0'

$\Sigma^1 = \{0, 1\}$ = set of all strings of length '1'

$\Sigma^2 = \{00, 01, 10, 11\}$

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots \cup \Sigma^n$ $\quad \Sigma^* = \{\epsilon\} \cup \{0,1\} \cup \{00,01,10,11,\dots\}$

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \dots \cup \Sigma^n$

(or)

$L^* = L_0 \cup L_1 \cup L_2 \cup \dots \cup L_n$

$L^+ = L_1 \cup L_2 \cup \dots \cup L_n$

* cardinality :- no. of elements in a set

$|\Sigma^n| = 2^n$

* Representation of FA (finite Automata) :-

A finite Automata is denoted with 5 tuple notation

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

Q = set of states

Σ = set of symbols (or) Input alphabet

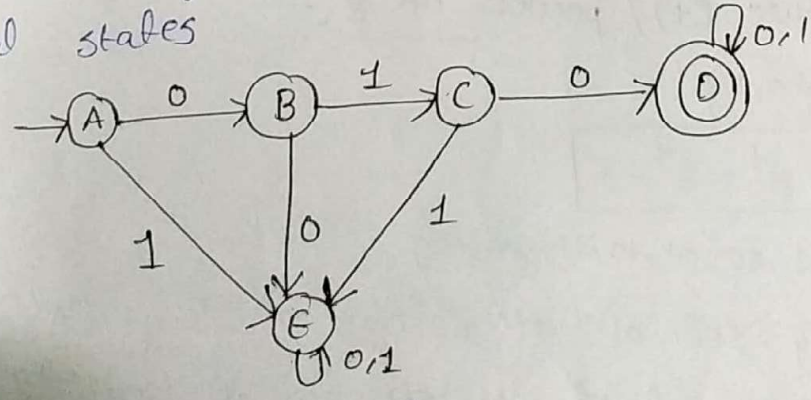
δ = transition function

q_s = starting state

F = set of final states (or) accepting states

→ A FA may have '0' (or) 1 (or) more no. of final states

Ex :-



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

$Q = \{A, B, C, D, E\}$

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

$q_s = A$

$F = \{D\}$

→ A FA can be represented in 3 ways

1) Transition Diagram
2) Transition table
3) Instantaneous Description (ID)

Transition Table :-

δ	0	1
→A	B	E
B	E	C
C	D	E
*D	D	D
E	E	E

Instantaneous description (ID) :-

δ (current state, input symbol) = { next state }

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

$\delta(A, 1) = \{E\}$

$\delta(B, 0) = \{E\}$

$\delta(B, 1) = \{C\}$

$\delta(C, 0) = \{D\}$

$\delta(C, 1) = \{E\}$

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

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

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

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

→ Every FA generates a language

→ the language generated by the FA is nothing but the set of strings accepted by the automata

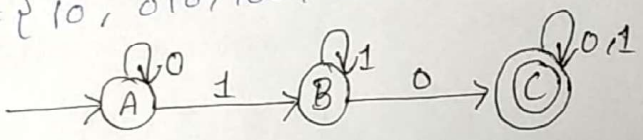
problems:-

1Φ) construct a FA for a formal language over an alphabet {0,1} in which every string contains substring of "10"

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

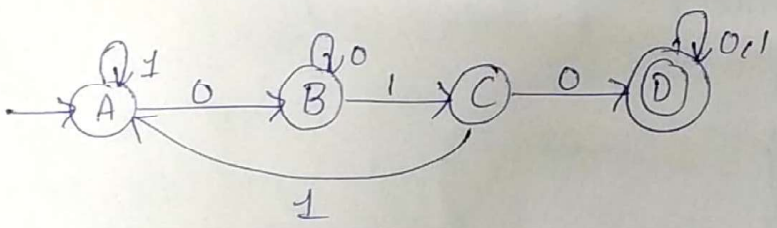
$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111,$
$0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001,$
$1010, 1011, 1100 - - - -\}$

$L = \{10, 010, 100, 101, 110, 0010, 0100, 0101 - - - -\}$



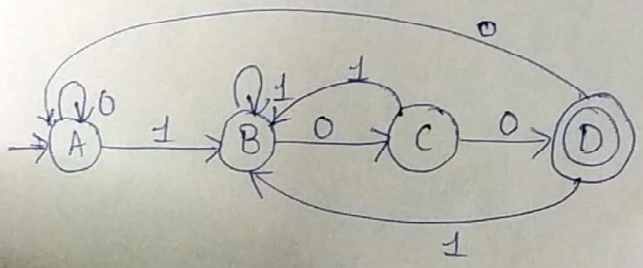
2Φ) substring of "010"?

sol: $L = \{010, 0010, 1010, 0100, 0101, - - - -\}$



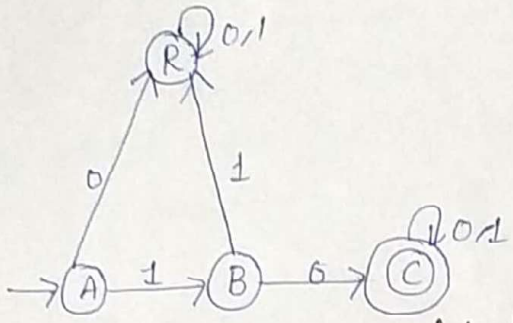
3Φ) ends with "100"

sol: $L = \{100, 0100, 1100, - - -\}$



4Φ) start with "10"

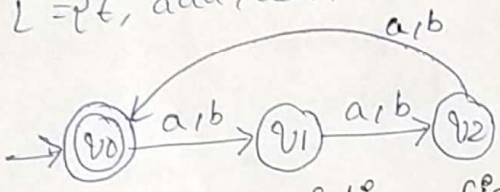
sol: $L = \{10, 100, 101, 1000, 1001, - - - - -\}$



50) Construct FA over an alphabet {a,b} in which length of the string is divisible by 3

Sol:- $L = \{|w| \% 3 == 0\}$

$L = \{\epsilon, aaa, aab, aba, abb, baa, bab, bba, bbb\}$



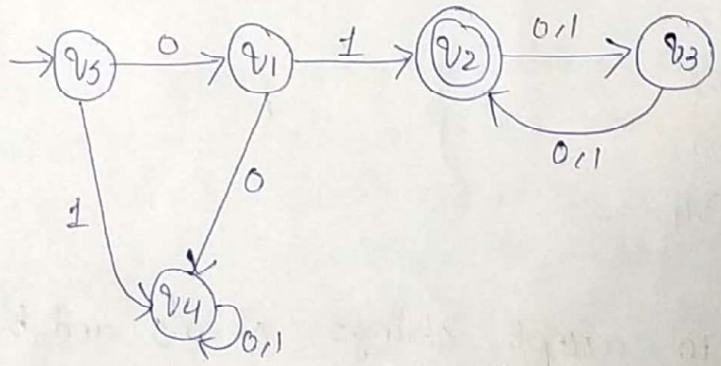
* DFA (Deterministic finite Automata) :-

- Deterministic refers to the uniqueness of the computation.
- In DFA the Input character goes to one state only
- DFA doesn't accept the null move that means the DFA cannot change state without any input character

→ In this we have 5 tuples

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

Ex:-



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

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

q_5 = Initial state $\in Q$

$f = q_5 \in Q$

$\delta = Q \times \Sigma \to Q$

Transition table:-

δ	0	1
$\to q_5$	q_1	q_4
q_1	q_4	q_2
$* q_2$	q_3	q_3
q_3	q_2	q_2
q_4	q_4	q_4

convert transition table to state diagram:-

$\delta(q_5, 0) = q_1$

$\delta(q_5, 1) = q_4$

$\delta(q_1, 0) = q_4$

$\delta(q_1, 1) = q_2$

$\delta(q_2, 0) = q_3$

$\delta(q_2, 1) = q_3$

$\delta(q_3, 0) = q_2$

$\delta(q_3, 1) = q_2$

$\delta(q_4, 0) = q_4$

$\delta(q_4, 1) = q_4$

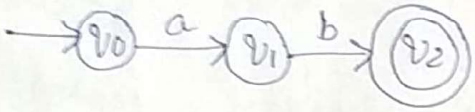
problems:-

1Q) obtain a DFA to accept strings of a's and b's starting with the string ab

sol:- It is clear that the string should start with 'ab'

and so, the minimum string that can be accepted by the machine is ab

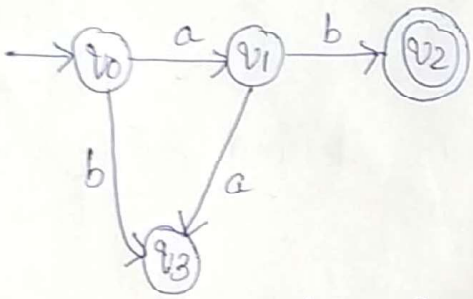
→ To accept the string ab, we need three states and the machine can be written as.



- 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 rejecting / dead state q_3 from q_0

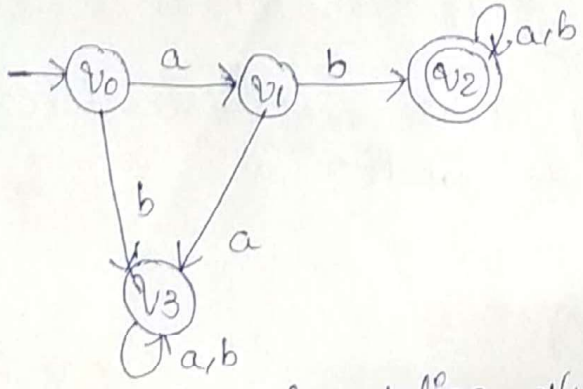
→ since the second char shouldn't be 'a' the input symbol 'a' is pointed to dead state q_3 from q_1



→ 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 and b's can be accepted and so remain in state q_2 only

→ the complete machine to accept the string of a's and b's starting with the string ab is shown in fig.



→ In the set notation, the language accepted by DFA can be represented as

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

∴ $M=(Q, \Sigma, \delta, q_0, f)$ where

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

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

q_0 = Initial state

$f = \{q_2\}$

→ Transition function δ is defined as

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

20) Draw a DFA to accept string of 0's and 1's ending with the string "011"

Sol: given $\Sigma=\{0,1\}$

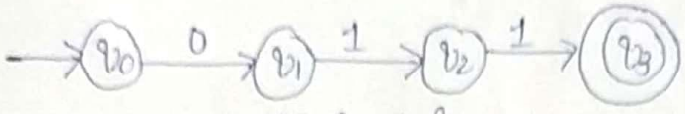
L = All the strings ending with string 011

so,

L = {011, 0000---011, 1111---011, 01000101 1---}

→ The basic string is 011, so draw FA for this using

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



Here q_0 – initial state

q_3 – final state

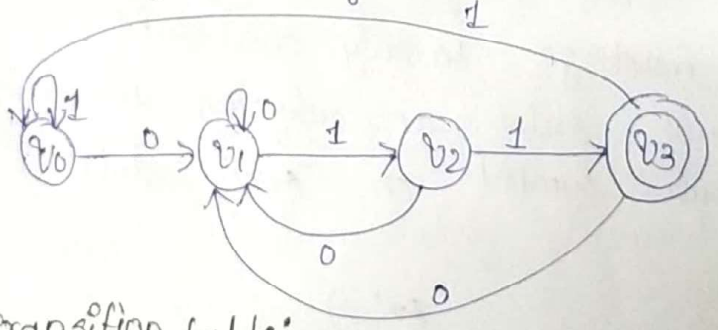
→ since the starting symbol can be anything the input symbol '1' of q_0 is self looped

→ since the ending must be 011 the input '0' of q_1 is self looped

→ similarly the input '0' of q_2 is pointed to q_1 so that it can end as 011

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

→ The DFA that accepts string ending with 011 is as shown fig:-



Transition table:-

states	symbols	
	0	1
q_0	q_0	q_0
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_1	q_0

* Acceptance of string :- A string 'w' over an alphabet 'Σ' is said to be accepted by an Automata 'm' $(Q, \Sigma, \delta, q_0, F)$ if and only if transition of $\delta(q_0, w) \in F$

Ex:-1) w=0101

$\hat{\delta}(A, 0101)$

$\hat{\delta}(B, 101)$

$\hat{\delta}(C, 01)$

$\hat{\delta}(D, 1) \Rightarrow \{D\} \in F$

so it is Accepted

w=011

$\delta(A, 011)$

$\delta(B, 11)$

$\delta(C, 1)$

$\{\epsilon\} \notin F$

Not accepted

L={ starts with "010" }

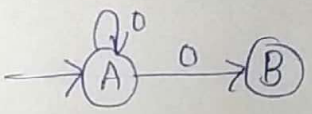
L={010, 0100, 0101, 01000, 01001, 01010, ----}

* complete system :- A FA is said to be complete if and only if it satisfies the following rules.

Rule-1 :- on any state if you read any input symbol then it must go to only one state

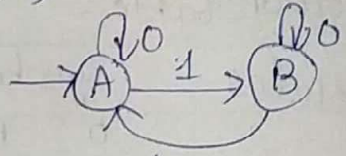
Rule-2 :- there must exist an outgoing transition for every input symbol on every state.

Ex:-1)



Not complete system (x)

Ex:-2)



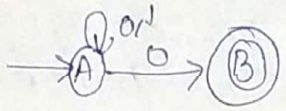
complete system (✓)

→ Based on complete system we are dividing the Automata as 2 types

1) DFA 2) NFA

* Non-deterministic finite Automata (NFA) :-

• A finite Automata (FA) is said to be non-determin-istic, if there is more than one possible transition from one state on the same input symbol

Ex:- 

→ A NFA is also set of five tuples and represented as, $M' = (Q, \Sigma, \delta, q_0, f)$

Q = set of all states

Σ = input symbols

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

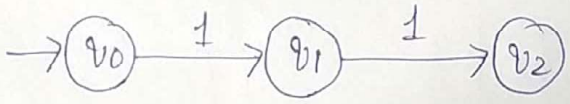
q_0 = start state (or) initial state

f = set of final state

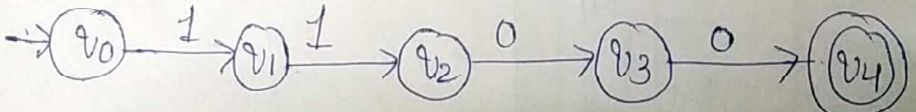
problems

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

Sol:- The FA with double '1' is as follows:

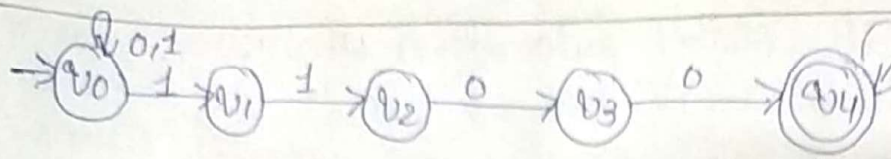


→ It should be immediately followed by double '0'



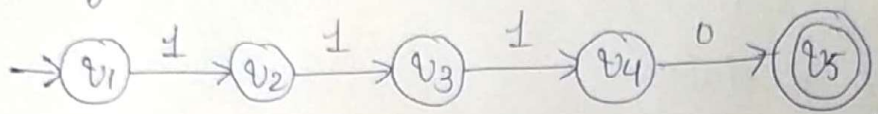
→ Now before double '1', there can be any string of '0' and '1' similarly, after double '0', there can be any string of '0' and '1'

→ Hence the NFA becomes:



2) Design a NFA in which all the string contain a substring 1110

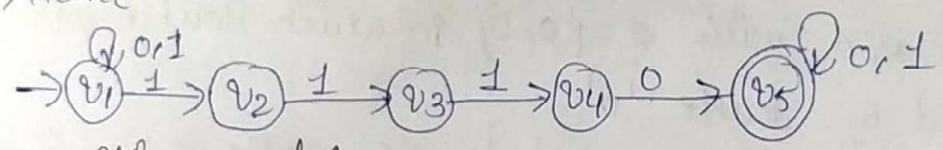
Sol: The language consists of all the string containing substring 1010. The partial transition diagram can be;



→ Now as 1010 could be the substring.

→ Hence we will add the inputs 0's and 1's so, that the substring 1010 of the language can be maintained

→ Hence the NFA becomes;



Transition table:-

present state	0	1
→ v_1	v_1	v_1, v_2
v_2	–	v_3
v_3	–	v_4
v_4	v_5	–
* v_5	v_5	v_5

→ consider a string 111010,

$$\delta(v_1, 111010) = \delta(v_1, 1100)$$
$$= \delta(v_1, 100)$$
$$= \delta(v_2, 00)$$

→ As there is no path from q_2 for input symbol 0. we can process string 111010 in another way

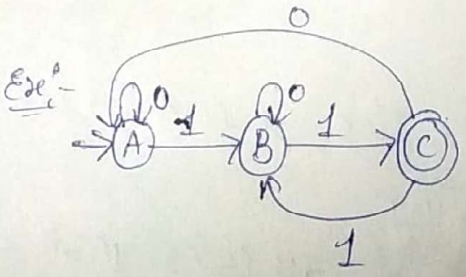
$$\begin{aligned}\delta(q1, 111010) &= \delta(q2, 11100)\\ &= \delta(q3, 1100)\\ &= \delta(q4, 00)\\ &= \delta(q5, 0)\\ &= \delta(q5, \varepsilon)\end{aligned}$$

→ As state q_5 is the accept state and we reached to the final state

Differences

DFA	NFA
1) Deterministic finite Automata	1) Non-deterministic finite Automata
2) DFA is complete system	2) It may/may not be complete
3) Every DFA is NFA	3) Every NFA may/may not be DFA. But NFA can be converted to DFA
4) DFA is as powerful as NFA	4) NFA is as powerful as DFA
5) for every input string there exist only one path	5) for every input string there exist 0 (or) 1 (or) more no. of paths

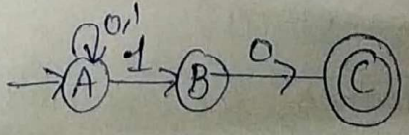
Ex:-

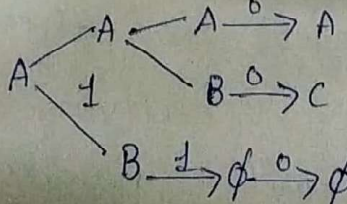


$A \xrightarrow{1} B \xrightarrow{1} B \xrightarrow{0} C$

Accepted

Ex:-





DFA	NFA
6) DFA is more efficient	6) NFA is comparatively less efficient
7) $\delta: Q \times \Sigma \to Q$	7) $\delta: Q \times \Sigma \to 2^Q$
8) Here we specify what is required to us & what is not 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 a multiple little machines computing at the same time

*Applications of finite Automata:-

The Applications of finite Automata are:-

1) A finite automata is highly useful to design lexical Analyzers
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 (Transducer)
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 / n/w protocol)

* NFA for Text search:-
→ Non-deterministic finite automata are mostly used for performing text search from group of words usually called as keywords. To determine the occurance of any keyword with in a text
→ for this reason a non-deterministic finite Automata is designed. This Automata is designed in such a way that it send signals upon seeing a keyword in the 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
The NFA to search text is designed as follows:-
1) The FA consist a start state with a transition to every input symbol
2) There exit K states for the keyword $m_1, m_2 \dots m_k$
3) A transition from the start state of v_1 on symbol m_1 will exit wherein a transition from v_1 to v_2 exist on symbol m_2 and on forth v_K state represents that it is an accepting state and signifies, it has found all the keywords $m_1, m_2 \dots m_k$.

* Languages Accepted by finite Automata with ε-transitions
• An NFA with ε-transition defined by $M=(Q, \Sigma, \delta, v_0, f)$ accepts set of strings only if it reaches the final state (v_f) that is, any string
→ that starts from initial state (v_0) and ends at final state (v_f) is said to be acceptable by FA

→ Let L be a language and 'w' be the string of language, then the acceptance of language by NFA is given by,

$L(M) = \{ w / w \in \Sigma^* \text{ and transition } (\delta) \text{ on } 'w' \text{ starts from } q_0 \text{ and ends at } q_F \}$

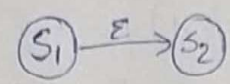
* significance of ϵ:-

when there is a requirement of moving from one state to another state without reading any input symbol

→ then we specify the transition with 'ϵ'

* conversion from NFA with ε to NFA without ε

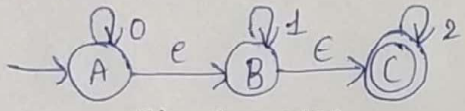
steps:-



1) find all edges starting from S_2
2) Duplicate all edges to S_1 without changing edge labels
3) if S_1 is initial state, make S_2 initial
4) if S_2 is final state make S_1 final
5) Remove dead state

Note:- If we have multiple 'ε' moves then eliminate the move which doesn't have 'ε' moves further then eliminate the other

Ex:-



NFA - ε moves

δ	0	1	2	ε
A	A	ϕ	ϕ	B
B	ϕ	B	ϕ	ε
C	ϕ	ϕ	ϕ	ϕ

ε-closure (A): {A, B, C}

ε-closure (B): {B, C}

ε-closure (C): {C}

$\delta(A,0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(A), 0))$

$= \varepsilon\text{-closure}(\delta(\{A,B,C\}, 0))$

$= \varepsilon\text{-closure}(\delta(A,0) \cup \delta(B,0) \cup \delta(C,0))$

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

$= \varepsilon\text{-closure}(A)$

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

$\hat{\delta}(A,1) = \epsilon\text{-closure}(\delta(\{A,B,C\},1))$

$= \epsilon\text{-closure}(\delta(A,1) \cup \delta(B,1) \cup \delta(C,1))$

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

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

$= \{B, C\}$

$\hat{\delta}(A,2) = \epsilon\text{-closure}(\delta(\{A,B,C\},2))$

$= \epsilon\text{-closure}(\delta(A,2) \cup \delta(B,2) \cup \delta(C,2))$

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

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

$= \{C\}$

$\hat{\delta}(B,0) = \epsilon\text{-closure}(\delta(\{B,C\},0))$

$= \epsilon\text{-closure}(\delta(B,0) \cup \delta(C,0))$

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

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

$= \{\phi\}$

$\hat{\delta}(B,1) = \epsilon\text{-closure}(\delta(\{B,C\},1))$

$= \epsilon\text{-closure}(\delta(B,1) \cup \delta(C,1))$

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

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

$= \{B, C\}$

$\hat{\delta}(B,2) = \epsilon\text{-closure}(\delta(\{B,C\},2))$

$= \epsilon\text{-closure}(\delta(B,2) \cup \delta(C,2))$

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

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

$= \{C\}$

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

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

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

$= \{\phi\}$

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

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

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

$= \{\phi\}$

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

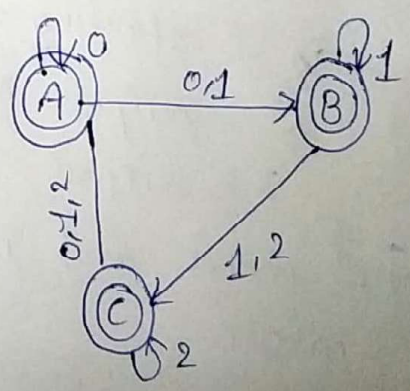
$= \epsilon\text{-closure}(\delta(C,2))$

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

$= \{C\}$

NFA without ϵ

$\hat{\delta}$	0	1	2
→A	{A,B,C}	{B,C}	{C}
B	{ϕ}	{B,C}	{C}
C	{ϕ}	{ϕ}	{C}



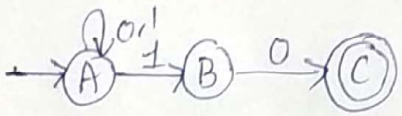
→ Here all are final states because the final state 'C' contains in all ϵ-closures of A, B, C

* How to convert NFA to DFA :-

steps :-

1) Initially $Q' = \phi$
2) Add q_0 of NFA to Q'. Then find the transitions from this start state
3) In 'Q' find the possible set of states for each input symbol. If this set of states is not in 'Q' then add it to 'Q'
4) In DFA, the final state will be all the states which contain f.

Ex :- 1)

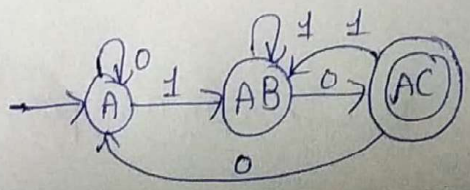


Transition table NFA

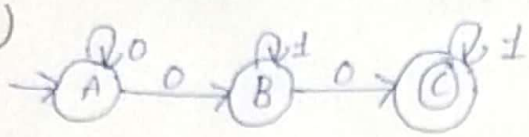
δ	1	0
→ A	{A,B}	{A}
B	ϕ	{C}
C	ϕ	ϕ

DFA

δ	0	1
→ [A]	[A]	[A,B]
[AB]	[AC]	[AB]
* [AC]	[A]	[AB]



Ex:- 2)

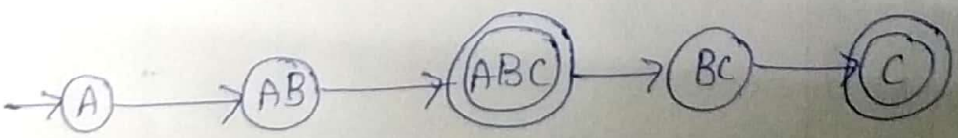


Transition Table:- NFA:-

S	0	1
→A	A,B	ϕ
B	C	B
C	ϕ	C

DfA

S	0	1
[A]	[A,B]	ϕ
[AB]	[ABC]	[B]
[ϕ]	[ϕ]	[ϕ]
[ABC]	[ABC]	[BC]
[BC]	[C]	[BC]
[B]	[C]	[B]
[C]	[ϕ]	[C]



→ If NFA contains 'n' states then maximum possible states in DFA is 2^n

FA without output :- when there is a requirement of more than one output 2 responses then we go with FA with 2 o/p's

2) types :-

1) moore machine
2) mealy machine

1) moore machine :- In moore output is associated with states

→ Both moore and mealy machines are denoted with 6 tuple

→ the output symbol at a given time dependends only on the present state of the machine

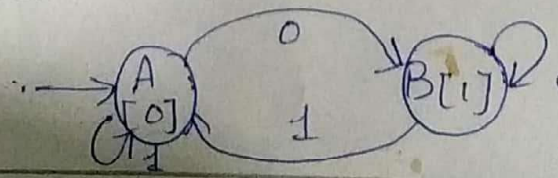
$M_0 = \{ Q, \Sigma, \delta, q_0, \Delta, \lambda \}$

- Q = finite set of states
- Σ = input symbol alphabets
- δ = Transition state $[Q \times \Sigma \to Q]$
- q_0 = starting state.
- Δ = output Alpha $\{0,1\}$
- λ = output transition $[Q \times \Sigma \to Q \times \Delta]$

→ Both moore and mealy machines are deterministic nature

→ In moore machine if the length of the string is 'n', then it gives you 'n+1' as output. where as in mealy it gives 'n' as output

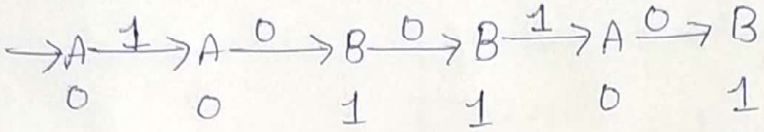
Ex :- moore machine for 1's compliment.



Transition table for moore

	0	1	λ
→A	B	A	0
B	B	A	1

string Acceptance :- 10010



→ while converting moore to mealy there is no change in no. of states and no. of transitions

2) mealy machine :-

→ In mealy output is associated with transitions

→ A 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 described by 6 tuples

$m\ell = \{Q, \Sigma, \Delta, \delta, \lambda, q_s\}$

$Q \rightarrow$ finite set of elements

$\Sigma \rightarrow$ input alphabet

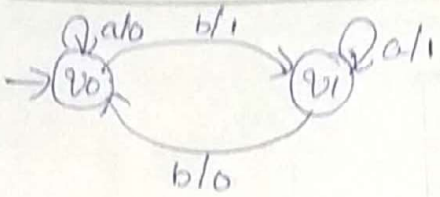
$\delta \rightarrow$ transition function $[Q \times \Sigma \rightarrow Q]$

$\lambda \rightarrow$ output transition function $[Q \times \Sigma \rightarrow \Delta]$

$\Delta \rightarrow$ output alphabet

$q_s \rightarrow$ starting state

Ex:-



$Q = \{q_0, q_1\}$

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

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

$q_0 = q_0$

$\delta = Q \times \Sigma \to Q$

$\lambda = Q \times \Sigma \to \Delta$

Transition table.

	a	b
q_0	q_0	q_1
q_1	q_1	q_0

$q_0 \times a \to 0$

$q_0 \times b \to 1$

$q_1 \times a \to 1$

$q_1 \times b \to 0$

→ If input string is of 'n' length the output also will be 'n'

$n \to n$

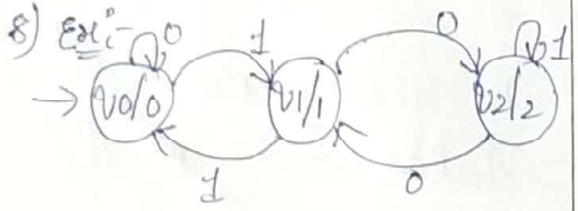
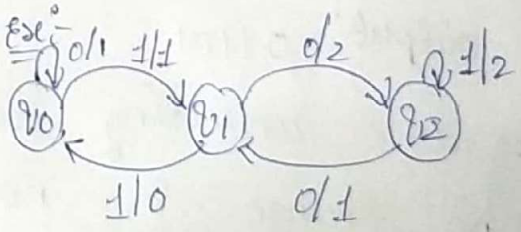
Ex:- abba input

0100 output

Overall table:-

state	next state			
	Input = a		Input = b	
	state	o/p	state	o/p
q_0	q_0	0	q_1	1
q_1	q_1	1	q_0	0

*

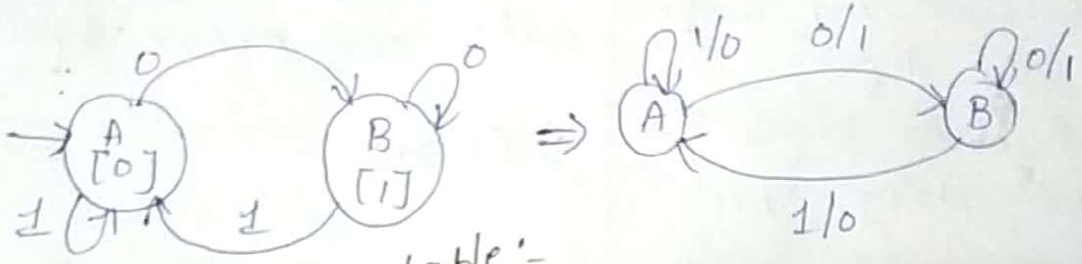
moore machine	mealy machine
1) output only depends on present state and Indepe-ndent of input	1) output depends on present state and present input
2) Input string length 'n' → output string 'n+1'	2) (n→n)
3) If we convert mealy to moore then the states may be increased	3) If moore to mealy then no change is no. of states
4) They react slower to inputs	4) They react faster to inputs
5) synchronous output and state generation	5) Asynchronous output generation
6) Easy to design	6) It is difficult to design
7) $\lambda : Q \to \Delta$	7) $\lambda : Q \times \Sigma \to \Delta$
8) Exi- 	8) Exi- 

→ If we convert mealy to moore then the states may be increased.

→ If moore to mealy then no change in no. of states.

* moore to mealy machine:-

Ex:- Transition state diagram



moore transition table:-

mealy:-

	0	λ	1	λ
→A	B	1	A	0
B	B	1	A	0

Input:- 10010

$A \xrightarrow{1} A \xrightarrow{0} B \xrightarrow{0} B \xrightarrow{1} A \xrightarrow{0} B$

0 1 1 0 1

output:- 01101

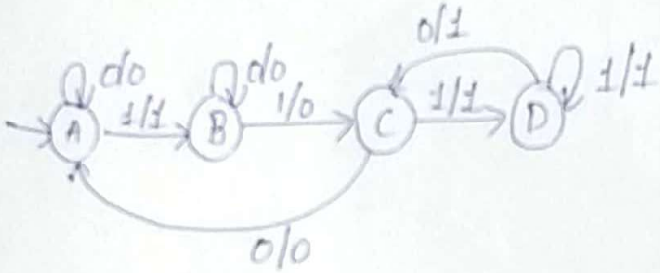
→ while converting moore to mealy there is no change in no. of states and no. of transitions

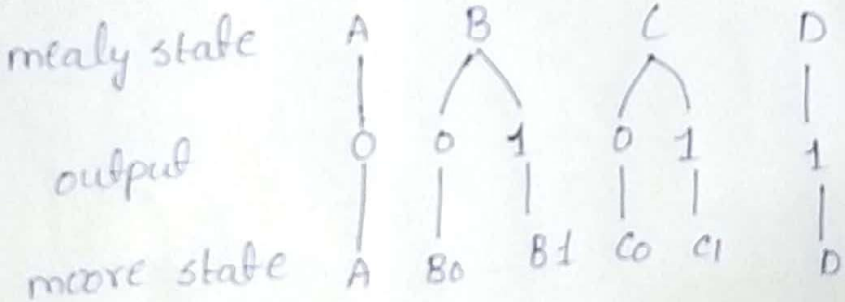
* conversion of mealy to moore :-

→ while converting mealy to moore there may be change in no. of states

mealy

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





moore

	0	1	1
→A	A	B1	0
B0	B0	C0	0
B1	B0	C0	1
C0	A	D	0
C1	A	D	1
D	C1	D	1