



# UNIT - I

## **Introduction to Finite Automata**

Structural Representations, Automata and Complexity, the Central Concepts of Automata Theory – Alphabets, Strings, Languages, Problems.

## **Nondeterministic Finite Automata**

Formal Definition, an application, Text Search, Finite Automata with Epsilon-Transitions.

## **Deterministic Finite Automata**

Definition of DFA, How A DFA Process Strings, The language of DFA, Conversion of NFA with  $\epsilon$ -transitions to NFA without  $\epsilon$ -transitions. Conversion of NFA to DFA, Moore and Melay machines

## UNIT - I

### INTRODUCTION TO FINITE AUTOMATA

#### FINITE AUTOMATA [FA]

Finite Automata is an abstract computing device. It is a mathematical model of a system with discrete inputs, outputs, ~~states~~ and set of transitions from state to state that occurs on input symbols from alphabet  $\Sigma$ .

#### Its Representations:

- Graphical (Transition Diagram or Transition table)
- Tabular (Transition Table)
- Mathematical (Transition function or Mapping function)

#### Formal Definition of Finite Automata:-

A finite automata is a 5-tuples; they are

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

$Q$  - States

$\delta$  -  $Q \times \Sigma \rightarrow Q$

Where

$Q$ : is a finite set called the states

$\Sigma$ : is a finite set called the alphabets

Transition function  $\delta: Q \times \Sigma \rightarrow Q$  is the transition function

$q_0 \in Q$  is the start state also called initial state

$F \subseteq Q$  is the set of accept states, also called final state.

#### STRUCTURAL REPRESENTATIONS:-

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

$$E \Rightarrow B + E$$

Regular Expressions: also denote the structure of data especially text strings.

$[A-Z][a-z]^* [ ] [A-Z][A-Z]^*$

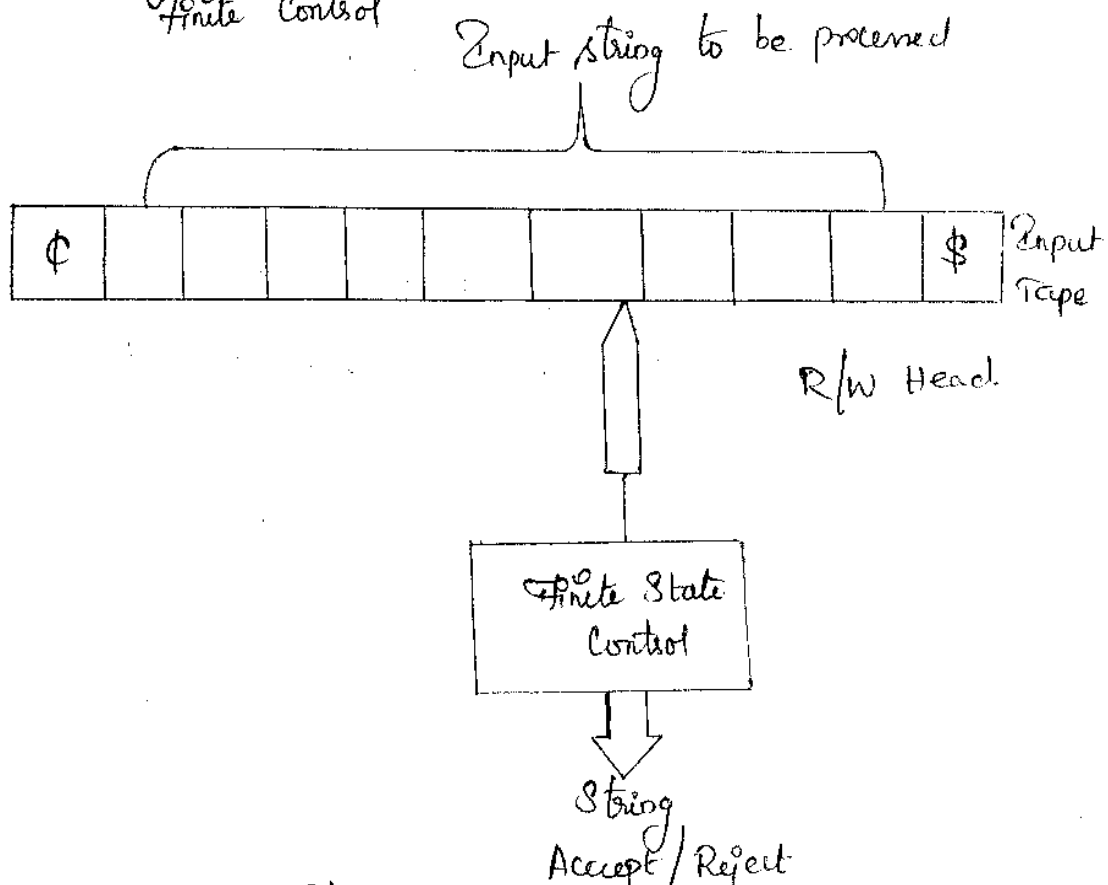
Complex Expressions:-

$^* [A-Z][a-z]^* ([ ] [A-Z][a-z]^* )^* [ ] [A-Z][A-Z]^*$

An automaton with a finite number of states is called finite automaton or finite state machine.

The block diagram of finite automaton consists of following three major components.

Input type  
Read/write head  
Finite control



Block diagram of Finite Automata

Input Tape:

The input tape is divided into squares, each square contains a single symbol from input alphabet  $\Sigma$

The end squares of each tape contain end markers  $\$$  at left <sup>end</sup> and  $\$$  at right end.

~~The end~~ Absence of end markers indicates that tape is of infinite length.

The left-to-right sequence of symbols between end markers is the input string to be processed.

### Read/Write Head:

The R/W head examines only one square at a time and can move one square either to the left or the right.

For further analysis, we restrict the movement of R/W head only to the right side.

### Finite State Control:

The finite state control is responsible for controlling total functioning of finite automata machine.

It will decide the which input symbol is read <sup>next</sup> and where to move either to the left or right.

The input to the finite control will be usually

Input symbol from input tape  
Present state of machine.

The output may be  
Movement of R/W head along the tape to the next square or to null move.

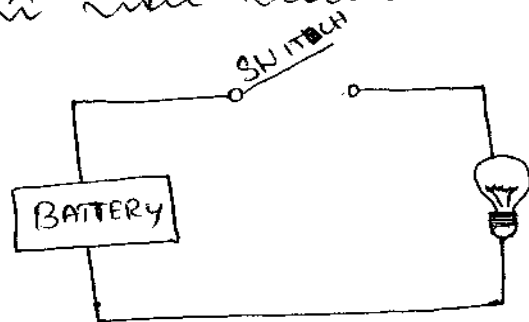
The next state/new state of FSM.

### AUTOMATA AND COMPLEXITY:

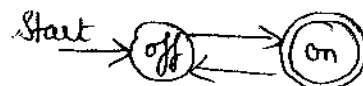
What can a computer do at all? This study is called "decidability" and the problems that can be solved by computer are called "decidable".

What can a computer do efficiently? This study is called "tractability" and the problems that can be solved by a computer using no more time than some slowly growing function of the size of the input are called "tractable".

Example: A Simple Computer



Input: Switch  
Output: light bulb  
Actions: flip switch  
States: On, Off



Finite automata: Devices with a finite amount of memory.  
Used to model "small" computers

Push-down automata: Devices with infinite memory that can be accessed in a restricted way.  
Used to model Parsers etc.

Turing machines: Devices with infinite memory.  
Used to model any computer.

time-bounded Turing Machines: Infinite memory, but bounded running time.  
Used to model any computer program that runs in a "reasonable" amount of time.

Automata Theory:- Is the study of abstract computational devices.

Automaton = an abstract computing device.

- \* Abstract devices are (simplified) models of real computations.
- \* Computations happens everywhere: on your laptop, on your cell phones etc,...

\* Why do we need abstract models?

Computability Vs Complexity.

Formal Languages: The Chomsky Hierarchy

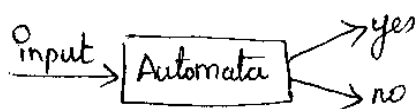
<u>Grammar</u>	<u>Language</u>	<u>Machine</u>
Unrestricted Grammar	RE Sets	TM
CSG	CSL	LBA
CFG	CFL	PDA
RG	RL	FA

Automatons are abstract models of machines that perform computations on an input by moving through a series of states or configurations.

At each state of the computation, a transition function determines the next configuration on the basis of a finite portion of the present configuration.

As a result, Once the computation reaches an accepting configuration, it accepts the input.

Example: Declaration statement in C language like `int a,b,c;`



Binary string ends with 0

101011010 → accepted

101000101 → rejected

Computability is a study of problems which can be solved by computers called decidable problems.

Decidability is the main topic in the study of computability

Computational Complexity is a study of :  
tractable problems solvable with slowly growing functions  
(like polynomial) of input size.

intractable problems solvable with fast growing functions  
(like exponential).

→ Intractability is the main topic of Computational Complexity.

### THE CENTRAL CONCEPTS OF AUTOMATA THEORY:-

Any formal language can be constructed by the  
basic concepts of automata theory.

Basic concepts of building blocks of automata  
theory.

#### Basic Concepts:-

- \* Symbol
- \* Alphabet
- \* Strings
- \* Languages
- \* Problem

Symbol :- Symbol is an object (or) a thing

Ex:  $a, b, c, d, \dots$   
 $0, 1, 2, 3, \dots$   
 $\#, *, +, -, @, \dots$

Symbols are used to form a string.

Alphabet :- It is a non-empty and finite set of symbols

- \* It is denoted by  $\Sigma$ . (or) conventional notation -  $\Sigma$
- \* The term "symbol" is usually undefined.

Example:

- \* Binary alphabet  $\Sigma = \{0, 1\}$
- \* English alphabet  $\Sigma = \{a, b, \dots, z\} \dots$

### Strings:

A string (or word) is a finite sequence of symbols from an alphabet.

Example:

1011 is a string from the binary alphabet  $\Sigma = \{0, 1\}$ .

Empty String:  $\epsilon$

A string with zero occurrences of symbols.

Length of string: String  $w$  length  $|w|$

The number of positions for symbols in  $w$

Example:

$|01111| = 5, |\epsilon| = 0, \dots$

Power of a symbol  $a$ :

The  $k^{\text{th}}$  power  $a^k$  of  $a$  is the result of concatenating  $a$  for  $k$  times, i.e.,  $a^k = aa \dots a$  ( $k$  times)

Power of an alphabet  $\Sigma$ :

The  $k^{\text{th}}$  power of  $\Sigma$ ,  $\Sigma^k$  is a set of all strings of length  $k$ .

Examples:

given  $\Sigma = \{0, 1\}$ , we have

$\Sigma^0 = \{\epsilon\}, \Sigma^2 = \{00, 01, 10, 11\}$

Power of a string  $x$  (supplemental):

Defined by Concatenation

$x^p = xxx \dots x$  ( $x$  concatenated  $p$  times)

Defined by Recursion

$x^0 = \epsilon$  (by definition); and

$x^p = xx^{p-1}$

Ex:  $(10)^0 = \epsilon, (011)^2 = 011011$

Note: for a symbol  $a$ , we define  $a^0 = \epsilon$  (i.e., in this case we

regard  $a$  as a one-symbol string)

Set of all strings over  $\Sigma$  - denoted as  $\Sigma^*$

It is not difficult to know that  $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$

$\Sigma^+$  = the set of nonempty strings from  $\Sigma = \Sigma^* - \{\epsilon\}$

$\therefore$ , we have

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots$

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



Concatenation of two strings  $x$  and  $y$  ---  $xy$

Examples:

If  $x = 01101$ ,  $y = 110$ , then  $xy = 01101110$ ,  $xx = x^2 = 0110101101$ , ...  
 $\epsilon$  is the identity for concatenation since  $\epsilon w = w\epsilon = w$ .

### Languages:-

A language is a set of strings all chosen from some  $\Sigma^*$ .

In other words, if  $\Sigma$  is an alphabet, and  $L \subseteq \Sigma^*$ , then  $L$  is a language over  $\Sigma$ .

Examples:

The set of all legal English words is a language.

$\therefore$  The set of all letters.

A legal program of C is a language.

$\therefore$  A subset of the ASCII characters.

More examples of languages...

The set of all strings of  $n$  0's followed by  $n$  1's

for  $n \geq 0$ :  $\{\epsilon, 01, 0011, 000111, \dots\}$

$\Sigma^*$  is an infinite language for any alphabet  $\Sigma$ .

$\emptyset$  denotes the empty language (not the empty string  $\epsilon$ )

which is a language over any alphabet.

$\{\epsilon\}$  is a language over any alphabet (consisting of only one string, the empty string  $\epsilon$ ).

Ways to describe languages...

Description by exhaustive listing...

$L_1 = \{a, ab, abc\}$  (finite languages; listed one by one)

$L_2 = \{a, ab, abb, abbb, \dots\}$  (infinite language; listed partially)

$L_3 = L(ab^*)$  (infinite language; expressed by a regular expression)

Description by generic elements...

$L_4 = \{x\}$   $x$  is over  $V = \{a, b\}$ , begins with  $a$ , followed by any number of  $b$ , possibly none

Note:  $L_4 = L_3 = L_2$

Description by integer parameters ---

$$L_5 = \{ab^n \mid n \geq 0\}$$

$$\text{Note: } L_5 = L_4 = L_3 = L_2$$

### Operations on Languages (Supplemental)

Languages are sets and operations of sets may be applied to them:

$$\text{union} \rightarrow A \cup B = \{a \mid a \in A \text{ or } a \in B\}$$

$$\text{intersection} \rightarrow A \cap B = \{a \mid a \in A \text{ and } a \in B\}$$

$$\text{difference} \rightarrow A - B = \{a \mid a \in A \text{ and } a \notin B\}$$

$$\text{Product} \rightarrow A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$$

$$\text{Complement} \rightarrow \bar{A} = \{a \mid a \in U \text{ and } a \notin A\}$$

$$\text{power set} \rightarrow 2^A = \{B \mid B \subset A\}$$

Note:  $U$  above is the universal set, just like  $\Sigma^*$  which is the closure of an alphabet ~~& def~~

### More Operations on Languages (Supplemental)

Concatenation of two languages  $L_1$  and  $L_2$  ---

$$L_1 L_2 = \{x_1 x_2 \mid x_1 \in L_1 \text{ and } x_2 \in L_2\}$$

Power of a language  $L$  ---

$$\text{Defined directly } L^k = \{x_1 x_2 \dots x_k \mid x_1, x_2, \dots, x_k \in L\}$$

$$\text{Defined by recursion} \dots L^0 = \{\epsilon\}; \text{ and } L^i = L L^{i-1}$$

$$\text{closure of language } L \dots L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup \dots$$

$$\text{positive closure of a language } L \dots L^+ = L^1 \cup L^2 \cup \dots$$

$$\text{Note: } L^* - L^0 = L^+ = L^* - \{\epsilon\}$$

### Problems:

A problem in automata theory  $\rightarrow$  deciding whether a given string is a member of some particular language.

E.g., if  $\Sigma$  is an alphabet and  $L$  is a language over  $\Sigma$ , the problem  $L$  is: Given a string  $w$  in  $\Sigma^*$ , decide if  $w \in L$  or not.

The solution will be studied later in the topic of decidability.

## Nondeterministic Finite Automata:-

A "nondeterministic" finite automaton (NFA) has the power to be in several states at once.

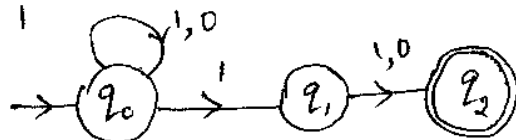
An Informal view of Nondeterministic Finite Automata

An NFA has a finite set of states, a finite set of input symbols, one start state and a set of accepting states. It also has a transition function, ( $\delta$ ).

The difference between the DFA and the NFA is in the type of  $\delta$ . For the NFA,  $\delta$  is a function that takes a state and input symbol as arguments.

Example:

An NFA accepting the set of all strings whose second last symbol is 1



## Formal Definitions:-

A nondeterministic finite automaton (NFA) is

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

where,  $Q$  is the finite set of states

$\Sigma$  is the finite set of input symbols (alphabets)

$\delta$ : the transition function is a function that takes a state in  $Q$  and an input symbol in  $\Sigma$  as arguments and returns a subset of  $Q$ .

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$$

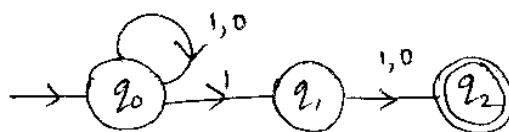
where  $P(Q)$  is the powerset of  $Q$ .

$q_0 \in Q$  initial state (or) a member of  $Q$ , is the start state.

$F$ : a subset of  $Q$ , is the set of final (or accepting) states.

$$F \subseteq Q$$

Example:

The NFA  can be specified formally as

$(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$  where the transition function  $\delta$  is given by the table.

Transition table for an NFA that accepts all strings ending in 01

$\delta$	0	1
$q_0$	$q_0$	$q_0, q_1$
$q_1$	$q_2$	$q_2$
$q_2$	$\emptyset$	$\emptyset$

The extended Transition Function:-

To extend the transition function  $\delta$  of an NFA to a function  $\hat{\delta}$  that takes a state  $q$  and a string of input symbols  $w$ , and returns the set of states that the NFA is in if it starts in state  $q$  and processes the string  $w$ .

For instance  $\hat{\delta}(q_0, 001) = \{q_0, q_2\}$ . Formally, we define  $\hat{\delta}$  for an NFA's transition function  $\delta$  by:

BASIS:  $\hat{\delta}(q, \epsilon) = \{q\}$ . i.e., without reading any input symbols, we are only in the state we began in.

INDUCTION: Suppose  $w$  is of the form  $w = xa$ , where  $a$  is the final symbol of  $w$  and  $x$  is the rest of  $w$ . Also suppose that

$\hat{\delta}(q, x) = \{p_1, p_2, \dots, p_k\}$ . Let

$$\bigcup_{i=1}^k \delta(p_i, a) = \{r_1, r_2, \dots, r_m\}$$

Then  $\hat{\delta}(q, w) = \{r_1, r_2, \dots, r_m\}$ .

We compute  $\hat{\delta}(q, w)$  by first computing  $\hat{\delta}(q, x)$  and by then following any transition from any of these states that is labeled  $a$ .

## The Language of an NFA:

An NFA accepts a string  $w$  if it is possible to make any sequence of choices of next state, while reading the characters of  $w$ , and go from the start state to any accepting state.

The other choices using the input symbols of  $w$  lead to a nonaccepting state or do not lead to any state at all (i.e., the sequence of states "dies"), does not prevent  $w$  from being accepted by the NFA as a whole.

Formally, if  $A = (Q, \Sigma, \delta, q_0, F)$  is an NFA, then

$$L(A) = \{w \mid \delta(q_0, w) \cap F \neq \emptyset\}$$

(i.e.)  $L(A)$  is the set of strings  $w$  in  $\Sigma^*$  such that  $\delta(q_0, w)$  contains at least one accepting state.

### Example:

Let us formally prove that the NFA of above figure accepts the language  $L_2 = \{w \mid w \text{ has the symbol } 1 \text{ in the second last position}\}$ . The proof is by mutual induction involving the following three statements.

$\delta^*(q_0, w)$  contains  $q_0$  for every  $w$   
 $\delta^*(q_0, w)$  contains  $q_1$  if and only if  $w$  ends in 1.  
 $\delta^*(q_0, w)$  contains  $q_2$  if and only if  $w$  has the symbol 1 in the second last position.

## Equivalence of Deterministic and Nondeterministic Finite Automata

There are many languages for which an NFA is easier to construct than a DFA, such as the language of strings that end in 01. It is a surprising fact that every language that can be described by some NFA can also be described by some DFA.

The proof that DFA's can do whatever NFA's can do involves an important "Construction" called the subset construction

because it involves constructing all subsets of the set of states of the NFA.

The subset construction starts from an NFA  $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ . Its goal is the description of a DFA  $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$  such that  $L(D) = L(N)$ .

Notice that the input alphabets of the two automata are the same and the start state of  $D$  is the set containing only the start state of  $N$ .

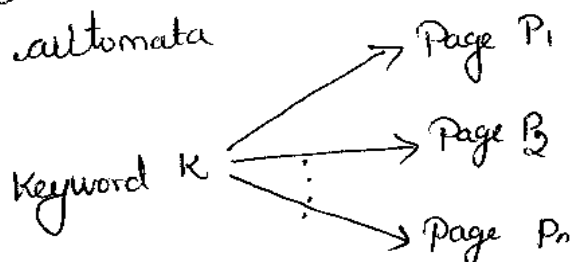
### AN APPLICATION: TEXT SEARCH

The abstract study, where we considered the "problem" of deciding whether a sequence of bits ends in 01, is actually an excellent model for several real problems that appear in applications such as web search and extraction of information from text.

Finding Strings in Text:  
 $\Rightarrow$  Searching Google for a set of words is equivalent to just finding strings in documents.

$\Rightarrow$  Techniques

- \* Using Inverted Indexes
- \* Using finite automata



### Inverted Indexing

$\Rightarrow$  Applications unsuitable to use inverted indexing:  
Document repository change rapidly

## Non-deterministic Finite Automata for Text Search:

We are given a set of keywords, which we shall call the keywords, and we want to find occurrences of any of these words.

In an application such as these, a useful way to proceed is to design a NFA, which signals, by entering an accepting state, that it has seen one of the keywords.

The text of a document is fed, one character at a time to this NFA, which then recognizes occurrences of the keywords in this text.

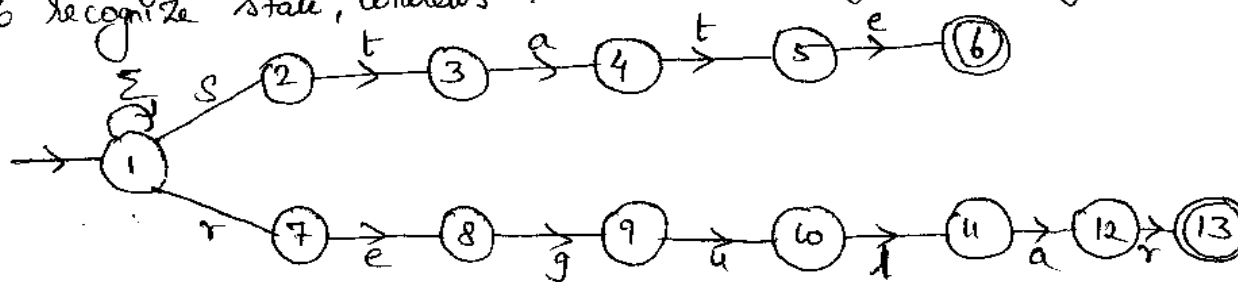
There is a simple form of NFA that recognizes a set of keywords.

1) There is a start state with a transition to itself on every input symbol.

2) For each keyword  $a_1, a_2, \dots, a_k$ , there are  $k$  states, say  $q_1, q_2, \dots, q_k$ . There is a transition from the start state to  $q_1$  on symbol  $a_1$ , a transition from  $q_1$  to  $q_2$  on symbol  $a_2$  and so on. The state  $q_k$  is an accepting state and indicates that the keyword  $a_1, a_2, \dots, a_k$  has been found.

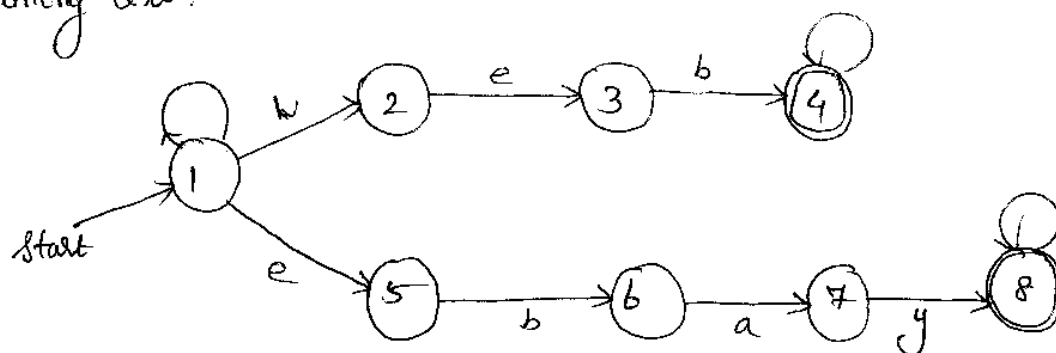
### Example:

Suppose we want to design an NFA to recognize the occurrence of words state and regular. The transition diagram for NFA, designed using the rules above, State 1 is the initial state and  $\Sigma$  is the set of all printable ASCII characters. States 2 through 6 recognize state, whereas states 7 through 13 recognize regular.





Use an NFA to search two keywords "Web" and "eBay" among text.



$\Sigma$  = set of all printable ASCII characters.

### A DFA to Recognize a set of keywords:-

The converted DFA has an equal number of states of the original NFA (an observation).

The DFA states may be constructed as (may be proved):

If  $q_0$  is start state of NFA, then  $\{q_0\}$  is start state of DFA.

If  $P$  is a state of NFA reachable (accessible) from a path  $x = a_1 a_2 \dots a_m$ , then one state of the DFA is a set of NFA states consisting of

1)  $q_0$

2)  $P$

3) every other NFA states reachable from

$q_0$  by following every subpath which is a suffix of  $x$ , like  $a_j a_{j-1} \dots a_m$ .

In the above way, all the states in the DFA may be constructed, and the all transitions derived.



## FINITE AUTOMATA WITH EPSILON TRANSITIONS

\* NFA is a finite automaton where for some cases when a single input is given to a single state, the machine goes to more than 1 states, (ie) some of the moves cannot be uniquely determined by the state and present input symbol.

\* In effect, an NFA is allowed to make a transition spontaneously, without receiving an input symbol.

### Use of $\epsilon$ -transitions :-

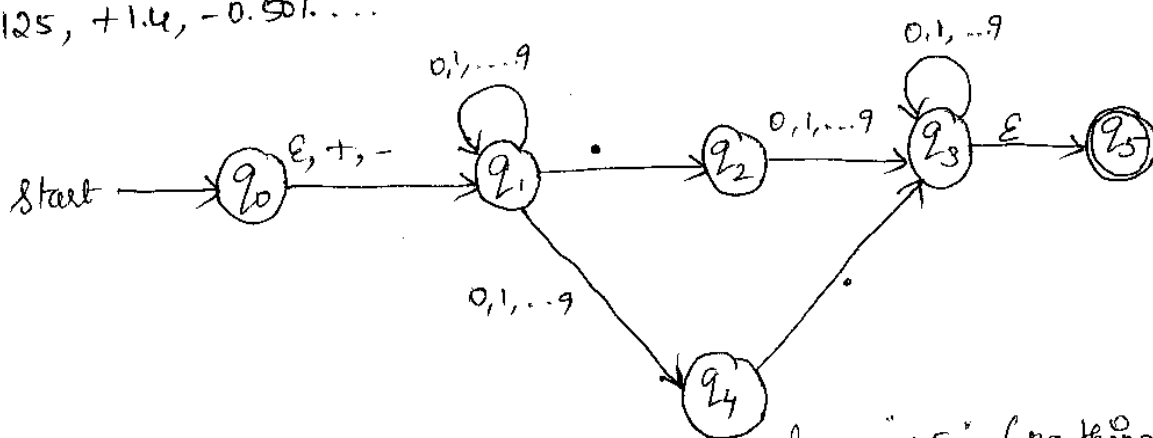
⇒ We allow the automaton to accept the empty string  $\epsilon$ .

⇒ This means that a transition is allowed to occur without reading in a symbol.

⇒ The resulting NFA is called  $\epsilon$ -NFA.

⇒ It adds "Programming (design) convenience" (more intuitive for use in designing FA's).

Example: An  $\epsilon$ -NFA accepting decimal numbers like 2.15, .125, +1.4, -0.501...



⇒ To accept a number like "+5" (nothing after the decimal point). We have to add  $q_4$ .

## Formal Notation for an E-NFA:-

Definition: An E-NFA  $A$  is denoted by  $A = (Q, \Sigma, \delta, q_0, F)$  where the transition function  $\delta$  takes as arguments.

- \* is a state in  $Q$ , and
- \* a member of  $\Sigma \cup \{\epsilon\}$

(i.e.) either an input symbol, or the symbol  $\epsilon$ ,

We require that  $\epsilon$ , the symbol for the empty string, cannot be a member of the alphabet  $\Sigma$ .

Example: The E-NFA of example 1 diagram is described as  $E = (\{q_0, q_1, \dots, q_5\}, \{., +, -, 0, 1, \dots, 9\}, \delta, q_0, \{q_5\})$

The transitions, e.g. include

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

$$- \delta(q_1, \epsilon) = \emptyset$$

The complete transition table of  $E$ .

	$\epsilon$	$+, -$	$.$	$0, 1, \dots, 9$
$q_0$	$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\emptyset$
$q_1$	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_1, q_4\}$
$q_2$	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$
$q_3$	$\{q_5\}$	$\emptyset$	$\emptyset$	$\{q_3\}$
$q_4$	$\emptyset$	$\emptyset$	$\emptyset \{q_3\}$	$\emptyset$
$q_5$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Epsilon - closures (E-closures)

⇒ We have to define the  $\epsilon$ -closure to define the extended transition function for the  $\epsilon$ -NFA.

⇒ We " $\epsilon$ -closure" a state  $q$  by following all transitions out of  $q$  that are labeled  $\epsilon$ .

⇒ Formal recursive definition of the set  $ECLOSE(q)$

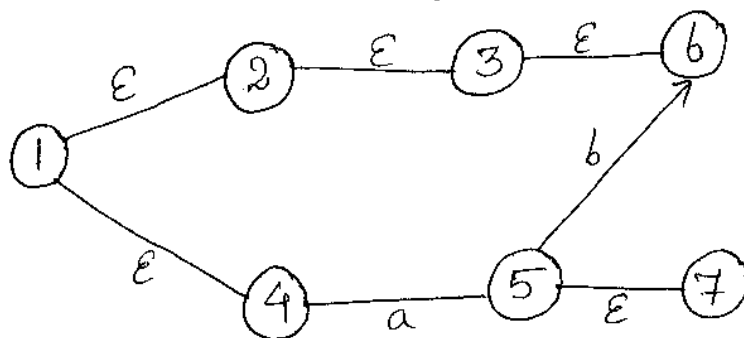
for  $q$ :

\* State  $q$  is in  $ECLOSE(q)$  (including the state itself);

\* If  $p$  is in  $ECLOSE(q)$ , then all states accessible from  $p$  through paths of  $\epsilon$ 's are also in  $ECLOSE(q)$ .

\*  $\epsilon$ -closure for a set of states  $S$ :

$$ECLOSE(S) = \bigcup_{q \in S} ECLOSE(q)$$



$\epsilon$ -transitions  
1, 2, 3, 4, 6.

$$ECLOSE(1) = \{1, 2, 3, 4, 6\}$$

$$ECLOSE(\{3, 5\}) = ECLOSE(3) \cup ECLOSE(5) = \{3, 6\} \cup$$

$$\{5, 7\} = \{3, 5, 6, 7\}$$

### Extended Transitions & Languages for $\epsilon$ -NFA's

⇒ Recursive definition of extended transition function:

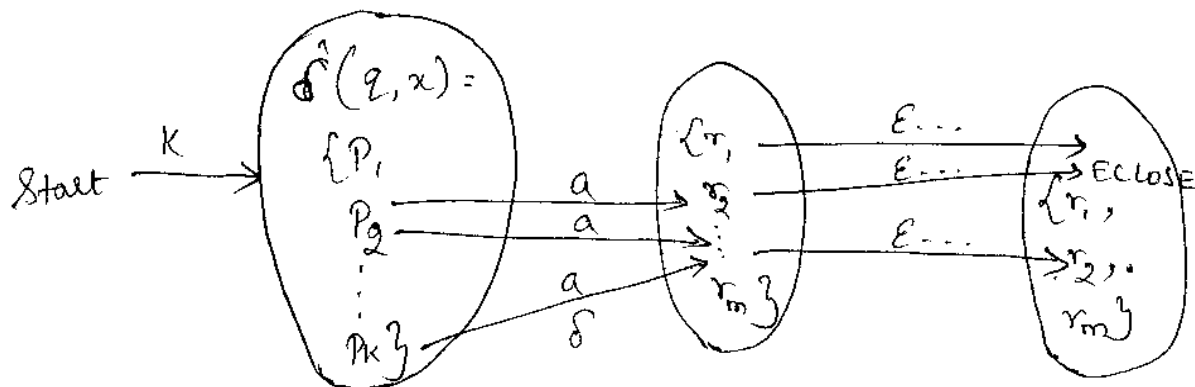
Basis:  $\hat{\delta}(q, \epsilon) = ECLOSE(q)$ .

Induction: if  $w = xa$ , then  $\hat{\delta}(q, w)$  is computed as

If  $\hat{\delta}(q, x) = \{p_1, p_2, \dots, p_k\}$  and

$$\bigcup_{i=1}^k \delta(p_i, a) = \{r_1, r_2, \dots, r_m\},$$

then  $\hat{\delta}(q, w) = \text{ECLOSE}(\{r_1, r_2, \dots, r_m\}) = \text{ECLOSE}(\bigcup_{i=1}^k \delta(p_i, a))$



### Eliminating E-transitions :-

→ The E-transition is good for design of FA, but for implementation, they have to be eliminated.

→ Given an E-NFA, we can find an equivalent DFA (a theorem).

→ Let  $E = (Q_E, \Sigma, \delta_E, q_0, f_E)$  be the given E-NFA, the equivalent DFA  $D = (Q_D, \Sigma, \delta_D, q_0, f_D)$  is constructed as follows

→  $Q_D$  is the set of subsets of  $Q_E$ , in which each accessible is an E-closed subset of  $Q_E$ , (ie) are sets  $S \subseteq Q_E$  such that  $S = \text{ECLOSE}(S)$ .

→ In other words, each E-closed set of states,  $S$ , includes those states such that any E-transition out of one of the states in  $S$  leads to a state that is also in  $S$ .

→  $q_D = \text{ECLOSE}(q_0)$  (initial state of  $D$ )

→  $F_D = \text{ECLOSE}\{S \mid S \in Q_D \text{ and } S \cap F_E \neq \emptyset\}$

→  $\delta_D(S, a)$  is computed for each  $a$  in  $\Sigma$  and each  $S$  in  $Q_D$  in the following way:

\* Let  $S = \{p_1, p_2, \dots, p_k\}$

\* Compute  $\bigcup_{i=1}^k \delta(p_i, a)$  and let this set be  $\{r_1, r_2, \dots, r_m\}$ .

\* Set  $\delta_D(S, a) = \text{ECLOSE}(\{r_1, r_2, \dots, r_m\})$   
 $= \text{ECLOSE}(\bigcup_{i=1}^k \delta(p_i, a))$

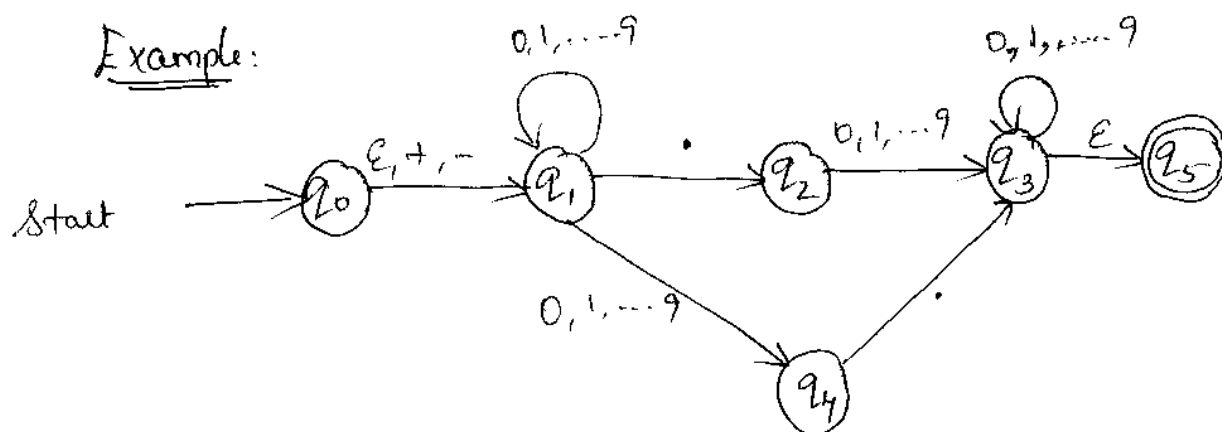
→ Technique to create accessible states in DFA  $D$ :

\* Starting from the start state  $q_0$  of E-NFA  $E$ , generate  $\text{ECLOSE}(q_0)$  as start state  $q_D$  of  $D$ ;

\* from the generated states to derive other

states.

Example:



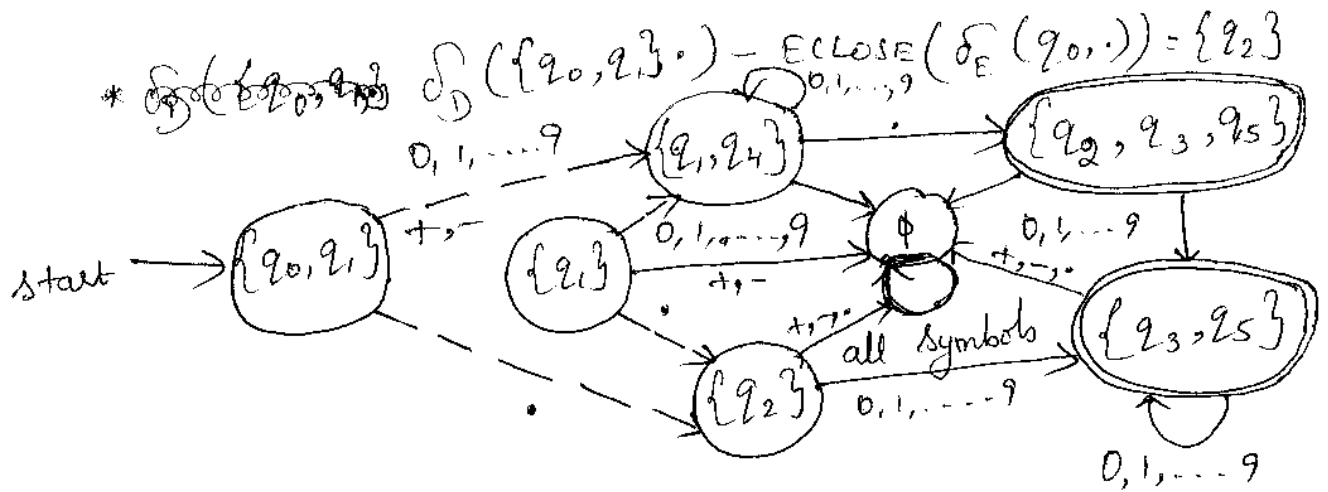
⇒ Start state  $q_D = \text{ECLOSE}(q_0) = \{q_0, q_1\}$

$\delta_D(\{q_0, q_1\}, +) = \text{ECLOSE}(\delta_E(q_0, +) \cup \delta_E(q_1, +))$

$= \text{ECLOSE}(\{q_1\} \cup \emptyset) = \text{ECLOSE}(\{q_1\}) = \{q_1\}$

\*  $\delta_D(\{q_0, q_1\}, 0) = \text{ECLOSE}(\delta_E(q_0, 0) \cup \delta_E(q_1, 0))$

$$= \text{ECLOSE}(\emptyset \cup \{q_1, q_4\}) = \text{ECLOSE}(\{q_1, q_4\}) = \{q_1, q_4\}$$



$\therefore q_5$  is the only accepting state of E  
 The accepting state of ① are those accessible states that contain  $q_5$

DETERMINISTIC FINITE AUTOMATA:

Finite Automata (or) Finite State Machine

Finite automata is also known as finite state machine.

Finite Automata

FA with Output

FA without Output

Moore Machine

Mealy Machine

DFA

NFA

E-NFA

$\Rightarrow$  There are states off and on, the automaton starts in off and tries to reach the "goal state" on  
 $\Rightarrow$  What sequences of fs lead to the goal state?

Answer:  $\{f, fff, fffff, \dots\} = \{f^n : n \text{ is odd}\}$

This is an example of a deterministic finite automaton over alphabet  $\{f\}$

The formalism of a DFA, one that is in a single state after reading any sequence of inputs. The term 'deterministic' refers to the fact that on each input there is one and only one state to which the automaton can transition from its current state.

Definition of a Deterministic Finite Automaton:-

A deterministic finite automaton consists of

- 1). A finite set of states, often denoted  $Q$
- 2). A finite set of input symbols, often denoted  $\Sigma$ .
- 3). A transition function that takes as arguments a state and an input symbol and returns a state. The transition function will commonly be denoted  $\delta$ .

If  $q$  is a state and  $a$  is an input symbol, then  $\delta(q, a)$  is that state  $p$  such that there is an arc labeled  $a$  from  $q$  to  $p$ .

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

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

where  $A$  - name of the DFA

$Q$  - set of states

$\Sigma$  - input symbols

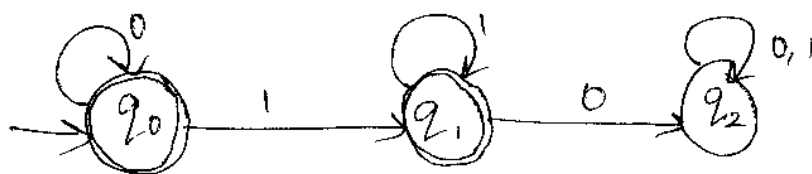
$\delta$  - transition function

$q_0$  - start state

$F$  - set of accepting states

the accepting states will be done by double loops

Example:



Alphabet  $\Sigma = \{0, 1\}$   
 Start state  $Q = \{q_0, q_1, q_2\}$   
 Initial state  $q_0$   
 Accepting states  $F = \{q_0, q_1\}$

Transition function

	inputs $\sigma$	
	0	1
States		
$q_0$	$q_0$	$q_1$
$q_1$	$q_2$	$q_1$
$q_2$	$q_2$	$q_2$

How a DFA processes strings:

→ Given an input string  $x = a_1 a_2 \dots a_n$ , if  
 $\delta(q_0, a_1) = q_1, \delta(q_1, a_2) = q_2, \dots,$   
 $\delta(q_{i-1}, a_i) = q_i, \dots, \delta(q_{n-1}, a_n) = q_n$

and  $q_n \in F$ ,

then  $x$  is "accepted"; otherwise, "rejected".

→ Every transition is deterministic

Example:

Design an FA  $A$  to accept the language

$L = \{w \mid w \text{ is of the form } x0y \text{ for some strings } x \text{ and } y \text{ consisting of 0's and 1's only}\}$

(or)  
 $L = \{01y \mid x \text{ and } y \text{ are any strings of 0's \& 1's}\}$

1's

→ strings in  $L$



## How A DFA Processes strings.

01, 11010, 100011, ...

First, its input alphabet is  $\Sigma = \{0, 1\}$ .

To decide whether 01 is a substring of the input, A needs to remember.

- 1) Has it already seen 01? If so, then it accepts every sequence of further inputs; (it will be only be in accepting states.)
- 2) Has it never seen 01, but its most recent input was 0, so if it now sees a 1, it will have seen 01 and can accept everything.
- 3) Has it never seen 01, but its last input was either nonexistent (it just started) or it last saw a 1.

A cannot accept until it first sees a 0 and then sees a 1 immediately after.

Example:

— Transitions:

$$\begin{aligned}\delta(q_0, 1) &= q_0, \delta(q_0, 0) = q_2, \delta(q_2, 1) = q_1, \\ \delta(q_2, 0) &= q_2, \delta(q_1, 0) = q_1, \delta(q_1, 1) = q_1\end{aligned}$$

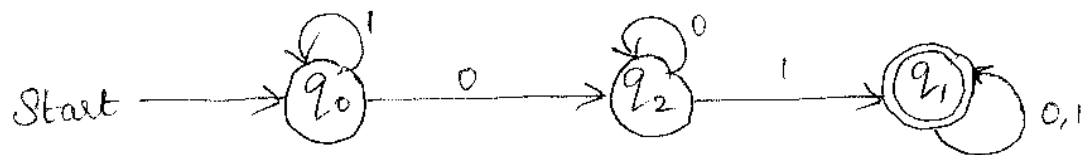
— 5-Tuple:

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\})$$

## Simple Notations for DFA's:-

— Transition Diagram

Transition diagram for a DFA  $A = (Q, \Sigma, \delta, q_0, F)$  is a graph defined as follows:



Transition Table:-

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

$\bullet \rightarrow \Rightarrow$  Initial state;  $* \Rightarrow$  final state

Extending transition function to strings:-

- Extended transition function:-

If  $x = a_1 a_2 \dots a_n$  and  $\delta$  is such that

$$\delta(q, a_1) = q_1, \delta(q_1, a_2) = q_2, \dots, q_n$$

$$\delta(q_{i-1}, a_i) = q_i, \dots, \delta(q_{n-1}, a_n) = q_n$$

then we define to be

$$\delta(q, x) = q_n$$

- Also may be defined recursively as in the textbook

- Recursive definition for

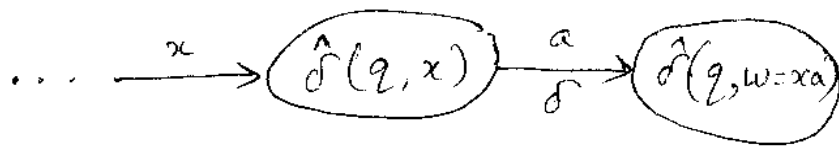
$$\text{Basis: } \delta(q, \epsilon) = q$$

$$\text{Induction: if } w = xa \text{ (a is the last symbol of w), then } \delta(q, w) = \delta(\delta(q, x), a)$$

A graphic diagram for the following concept:

Induction:

$$\text{if } w = xa \text{ (a is the last symbol of w), then } \delta(q, w) = \delta(\delta(q, x), a)$$



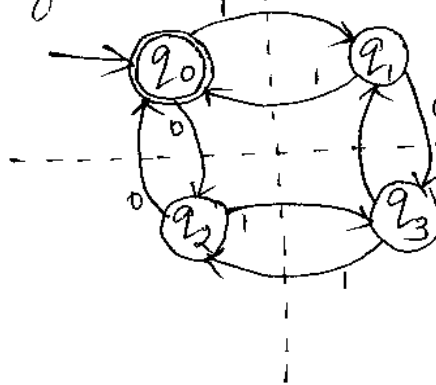
## THE LANGUAGE OF A DFA:

- is defined as
- The language of a DFA  $A = (Q, \Sigma, \delta, q_0, F)$ .

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

- If  $L$  is  $L(A)$  for some DFA  $A$ , then we say  $L$  is a Regular Language.

Example: Let us design a DFA to accept the language  $L = \{w \mid w \text{ has both an even number of 0's and an even number of 1's}\}$ .



DFA for language  $L$ .  
 $L(A) = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\})$

	0	1
* $\rightarrow q_0$	$q_2$	$q_1$
$q_1$	$q_3$	$q_0$
$q_2$	$q_0$	$q_3$
$q_3$	$q_1$	$q_2$

\* Input string is 110101

\* We expect that  $\delta(q_0, 110101) = q_0$

\* Accepting state is  $q_0$

\* Compute the  $\delta(q_0, w)$  for each

prefix  $w$  of 110101.

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

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

$$\hat{\delta}(q_0, 11) = \delta(\hat{\delta}(q_0, 1), 1) = \delta(q_1, 1) = q_0$$

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

$$\hat{\delta}(q_0, 1101) = \delta(\hat{\delta}(q_0, 110), 1) = \delta(q_2, 1) = q_3$$

$$\hat{\delta}(q_0, 11010) = \delta(\hat{\delta}(q_0, 1101), 0) = \delta(q_3, 0) = q_1$$

$$\hat{\delta}(q_0, 110101) = \delta(\hat{\delta}(q_0, 11010), 1) = \delta(q_1, 1) = q_0$$

### Deterministic Finite Automata (DFA)

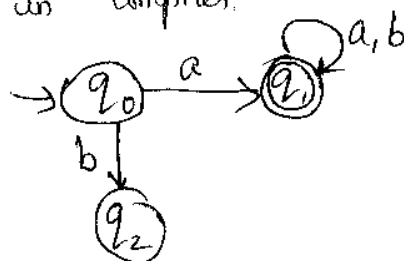
The finite automata are called deterministic finite automata if the machine is read an input string one symbol at a time.

Deterministic refers to the uniqueness of the computation.

In DFA, there is only one path for specific input from the current state to the next state.

DFA does not accept the null move, (i.e.) DFA cannot change state without any input character.

DFA can contain multiple final states. It is used in lexical analysis in compilers.



$q_0$  - initial  
 $q_1$  - final  
 $\Sigma = \{a, b\}$

### Formal Definition of DFA:-

A DFA is a collection of 5 tuples (same as FA)

$Q$ : finite set of states

$\Sigma$ : finite set of input symbols.

$q_0$ : Initial State

$F$ : final State

$\delta$ : Transition function

Transition function can be defined as:  $\delta: Q \times \Sigma \rightarrow Q$

### Graphical Representation of DFA:-

DFA can be represented by digraphs called state diagram.

- 1) The state is represented by vertices.
- 2) The arc labeled with an input character show the transition.
- 3) The initial state is marked with an arrow.
- 4) The final state is denoted by a double circle.

### Acceptance of Language:-

- A language acceptance is defined by "if a string  $w$  is accepted by the machine  $M$  (ie). if it is reaching the final state  $F$  by taking the string  $w$ ."

- Not accepted if not reaching the final state.

Eg:-  $L = \{\emptyset\} \Rightarrow \rightarrow (q_0)$  (no string)

$L = \{\epsilon\} \Rightarrow \rightarrow (q_0)$

• DFA to accept "a"  $\Rightarrow \rightarrow (q_0) \xrightarrow{a} (q_1)$  {states

• DFA to accept zero or more 'a' depends on length of string }

$L = \{\epsilon, a, aa, aaa, \dots\}$

## CONVERSION OF NFA WITH $\epsilon$ -TRANSITIONS TO NFA WITHOUT $\epsilon$ -TRANSITIONS

In this method we try to remove all the  $\epsilon$  transitions from given NFA. The ~~new~~ method will be

$\Rightarrow$  Find out all the  $\epsilon$ -transitions from each state from  $Q$  that will be called as  $\epsilon$ -closure  $\{E_x\}$  where  $E_x \in Q$ .

$\Rightarrow$  The  $\delta'$  transitions can be obtained. The  $\delta'$  transitions means an  $\epsilon$  closure on  $\delta$  moves.

$\Rightarrow$  Step 2 is repeated for each input symbol and for each state of given NFA.

$\Rightarrow$  Using the resultant states the transitions table for equivalent NFA without  $\epsilon$  can be built.

Example:



Solution:

Step 1:  $M = \{Q, \Sigma, \delta, q_0, F\}$

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

$\Sigma = \{0, 1\}$ ,  $F = q_1$

Transition Table:

Input States	Input		
	0	1	$\epsilon$
$q_0$	$\{q_0\}$	-	$\{q_1\}$
$q_1$	-	$\{q_1\}$	-

Step 2: Find the  $\epsilon$ -closure

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

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

Step 3: Find the processing states of  $q_0, q_1$

$$\begin{aligned} q_0 \quad \delta'(q_0, 0) &= \epsilon\text{-closure}(\delta(q_0, 0)) \Rightarrow \delta(q_0, 0) \\ &\Rightarrow \delta(q_1, 0) \\ &= \epsilon\text{-closure}(q_0) \\ &= \{q_0, q_1\} \end{aligned}$$

$$\begin{aligned} \delta'(q_0, 1) &= \epsilon\text{-closure}(\delta(q_0, 1)) \Rightarrow \delta(q_0, 1) \\ &\Rightarrow \delta(q_1, 1) \\ &= \epsilon\text{-closure}(q_1) \\ &= \{q_1\} \end{aligned}$$

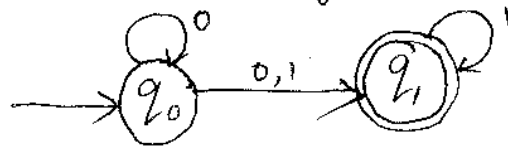
$$\begin{aligned} \delta'(q_1, 0) &= \epsilon\text{-closure}(\delta(q_1, 0)) \\ &= \epsilon\text{-closure}(\phi) \\ &= \phi \end{aligned}$$

$$\begin{aligned} \delta'(q_1, 1) &= \epsilon\text{-closure}(\delta(q_1, 1)) \\ &= \epsilon\text{-closure}(q_1) \\ &= \{q_1\} \end{aligned}$$

Step 4: Transition Table

Input States	0	1
$q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$q_1$	$\phi$	$\{q_1\}$

Step 5: Transition diagram for NFA without Epsilon



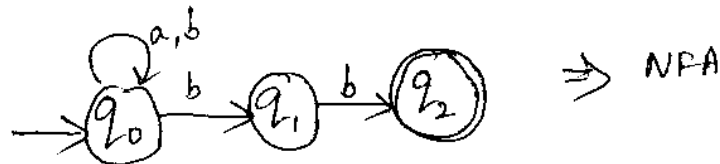
CONVERSION OF NFA TO DFA:-

$$\text{NFA} \Rightarrow \delta: Q \times \Sigma \Rightarrow 2^Q$$

$$\text{DFA} \Rightarrow \delta: Q \times \Sigma \Rightarrow Q$$

DFA is Current state entered the input symbol into the new or next state

NFA is Current state and input symbol reach to the multiple path.



Step 1: Construct NFA transition table

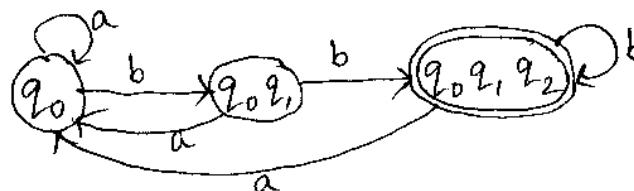
	a	b
q <sub>0</sub>	q <sub>0</sub>	{q <sub>0</sub> , q <sub>1</sub> }
q <sub>1</sub>	<del>q<sub>0</sub></del>	q <sub>2</sub>
q <sub>2</sub>	-	-

NFA

$\delta$	a	b
q <sub>0</sub>	q <sub>0</sub>	{q <sub>0</sub> , q <sub>1</sub> }
q <sub>1</sub>	-	q <sub>2</sub>
q <sub>2</sub>	-	-

DFA

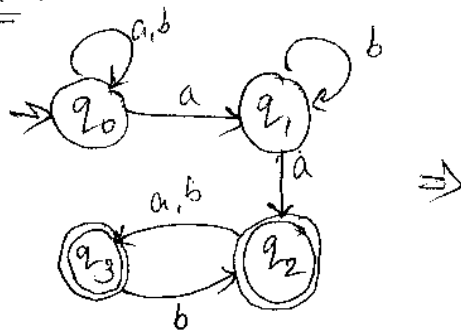
$\delta$	a	b
q <sub>0</sub>	q <sub>0</sub>	{q <sub>0</sub> , q <sub>1</sub> }
[q <sub>0</sub> , q <sub>1</sub> ]	{q <sub>0</sub> }	{q <sub>0</sub> , q <sub>1</sub> , q <sub>2</sub> }
[q <sub>0</sub> , q <sub>1</sub> , q <sub>2</sub> ]	{q <sub>0</sub> }	{q <sub>0</sub> , q <sub>1</sub> , q <sub>2</sub> }





Example 2:

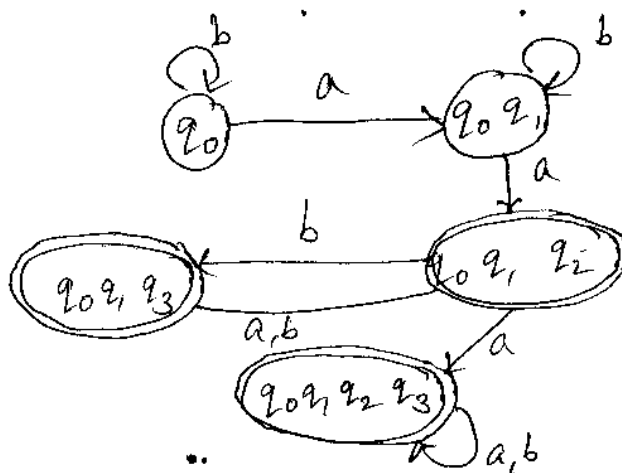
NFA



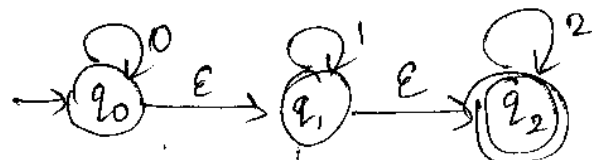
$\delta$	a	b
$q_0$	$\{q_0, q_1\}$	$q_0$
$q_1$	$q_2$	$q_1$
$q_2$	$q_3$	$q_3$
$q_3$	-	$q_2$

DFA:

$\delta$	a	b
$q_0$	$[q_0, q_1]$	$q_0$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$
$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3, q_2]$
$[q_0, q_1, q_3]$	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$



Examples: Convert the given NFA with  $\epsilon$  to NFA without  $\epsilon$ .



Solution:

We will first obtain  $\epsilon$ -closure of each state (i.e.) we will find out  $\epsilon$  reachable states from current state.

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

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

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

As  $\epsilon\text{-closure}(q_0)$  means with null input (no input symbol) we can reach to  $q_0, q_1, q_2$ . In a similar manner for  $q_1$  and  $q_2$ ,  $\epsilon\text{-closures}$  are obtained. Now we will obtain  $\delta'$  transitions for each state on each input symbol.

$$\begin{aligned}\delta'(q_0, 0) &= \epsilon\text{-closure}(\delta(\delta'(q_0, \epsilon), 0)) \\ &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), 0)) \\ &= \epsilon\text{-closure}(\delta(q_0, q_1, q_2), 0) \\ &= \epsilon\text{-closure}(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)) \\ &= \epsilon\text{-closure}(q_0 \cup \phi \cup \phi) \\ &= \epsilon\text{-closure}(q_0) = \underline{\{q_0, q_1, q_2\}}\end{aligned}$$

$$\begin{aligned}\delta'(q_0, 1) &= \epsilon\text{-closure}(\delta(\delta'(q_0, \epsilon), 1)) \\ &= \epsilon\text{-closure}(\delta(q_0, q_1, q_2), 1) \\ &= \epsilon\text{-closure}(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)) \\ &= \epsilon\text{-closure}(\phi \cup q_1 \cup \phi) \\ &= \epsilon\text{-closure}(q_1) = \underline{\{q_1, q_2\}}\end{aligned}$$

$$\begin{aligned}\delta(q_1, 0) &= \epsilon\text{-closure}(\delta(\delta(q_1, \epsilon), 0)) \\ &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), 0)) \\ &= \epsilon\text{-closure}(\delta(q_1, q_2), 0) \\ &= \epsilon\text{-closure}(\delta(q_1, 0) \cup \delta(q_2, 0)) \\ &= \epsilon\text{-closure}(\phi \cup \phi) = \underline{\phi}\end{aligned}$$

$$\begin{aligned}\delta(q_0, 2) &= \epsilon\text{-closure}(\delta(\delta(q_0, \epsilon), 2)) \\ &= \epsilon\text{-closure}(\delta(\delta(q_0, q_1, q_2), 2)) \\ &= \epsilon\text{-closure}(\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2)) \\ &= \epsilon\text{-closure}(\phi \cup \phi \cup q_2) \\ &= \epsilon\text{-closure}(q_2) = \underline{\{q_2\}}\end{aligned}$$

$$\begin{aligned}\delta(q_1, 1) &= \epsilon\text{-closure}(\delta(\delta(q_1, \epsilon), 1)) \\ &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), 1))\end{aligned}$$

$$\begin{aligned}
 &= \epsilon\text{-closure}(\delta(q_1, q_2), 1) \\
 &= \epsilon\text{-closure}(\delta(q_1, 1) \cup \delta(q_2, 1)) \\
 &= \epsilon\text{-closure}(\phi \cup \phi) \\
 &= \epsilon\text{-closure}(q_1) = \underline{\underline{\{q_1, q_2\}}}
 \end{aligned}$$

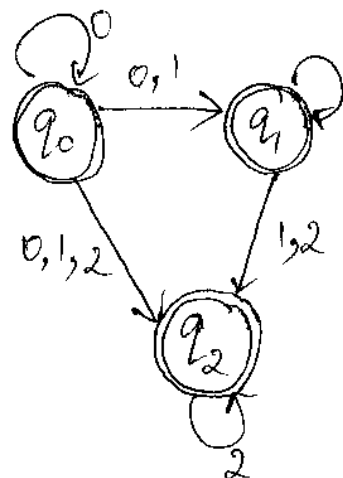
$$\begin{aligned}
 \delta(q_1, 2) &= \epsilon\text{-closure}(\delta(\delta(q_1, \epsilon), 2)) \\
 &= \epsilon\text{-closure}(\delta(q_1, q_2), 2) \\
 &= \epsilon\text{-closure}(\delta(q_1, 2) \cup \delta(q_2, 2)) \\
 &= \epsilon\text{-closure}(\phi \cup q_2) \\
 &= \epsilon\text{-closure}(q_2) = \underline{\underline{\{q_2\}}}
 \end{aligned}$$

$$\begin{aligned}
 \delta(q_2, 0) &= \epsilon\text{-closure}(\delta(\delta(q_2, \epsilon), 0)) \\
 &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), 0)) \\
 &= \epsilon\text{-closure}(\delta(q_2, 0)) \\
 &= \epsilon\text{-closure}(\phi) = \underline{\underline{\{\phi\}}}
 \end{aligned}$$

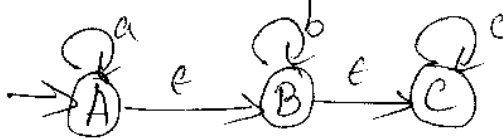
$$\begin{aligned}
 \delta(q_2, 1) &= \epsilon\text{-closure}(\delta(\delta(q_2, \epsilon), 1)) \\
 &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), 1)) \\
 &= \epsilon\text{-closure}(\delta(q_2, 1)) \\
 &= \epsilon\text{-closure}(\phi) = \underline{\underline{\{\phi\}}}
 \end{aligned}$$

$$\begin{aligned}
 \delta(q_2, 2) &= \epsilon\text{-closure}(\delta(\delta(q_2, \epsilon), 2)) \\
 &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), 2)) \\
 &= \epsilon\text{-closure}(\delta(q_2, 2)) \\
 &= \epsilon\text{-closure}(q_2) = \underline{\underline{\{q_2\}}}
 \end{aligned}$$

Input State	0	1	2
$q_0$	$q_0, q_1, q_2$	$q_1, q_2$	$q_2$
$q_1$	$\phi$	$q_1, q_2$	$q_2$
$q_2$	$\phi$	$\phi$	$q_2$



E-NFA to NFA for  $a^* b^* c^*$



Solution:

$\epsilon^* a \epsilon^*$   
 $A \rightarrow A \rightarrow A \rightarrow A$   
 $\swarrow \searrow$   
 $B \rightarrow \phi \rightarrow B$   
 $\searrow$   
 $C \rightarrow \phi$

	a	b	c
A	{ABC}	{B,C}	{C}
B	{ $\phi$ }	{B,C}	{C}
C	{ $\phi$ }	{ $\phi$ }	{C}

$\epsilon^* b \epsilon^*$   
 $A \rightarrow A \rightarrow \phi$   
 $\swarrow \searrow$   
 $B \rightarrow B \rightarrow B$   
 $\searrow$   
 $C \rightarrow \phi \rightarrow C$

$\epsilon^* c \epsilon^*$   
 $A \rightarrow A \rightarrow \phi$   
 $\swarrow \searrow$   
 $B \rightarrow \phi$   
 $\searrow$   
 $C \rightarrow C \rightarrow C$

$\epsilon^* a \epsilon^*$   
 $B \rightarrow B \rightarrow \phi$   
 $\searrow$   
 $C \rightarrow \phi$

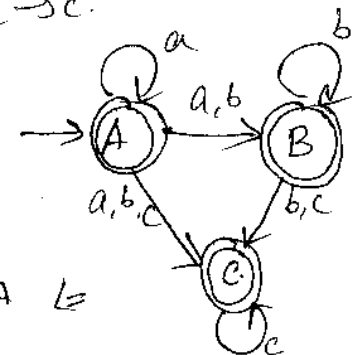
$\epsilon^* b \epsilon^*$   
 ~~$A \rightarrow A \rightarrow \phi$~~   
 ~~$B \rightarrow B \rightarrow B$~~   
 $\searrow$   
 $C \rightarrow \phi$

$\epsilon^* c \epsilon^*$   
 $B \rightarrow B \rightarrow \phi$   
 $\searrow$   
 $C \rightarrow C \rightarrow C$

$\epsilon^* a \epsilon^*$   
 $C \rightarrow C \rightarrow \phi$

$\epsilon^* b \epsilon^*$   
 $C \rightarrow C \rightarrow \phi$

$\epsilon^* c \epsilon^*$   
 $C \rightarrow C \rightarrow C \rightarrow C$



Drawn FA  $\Leftarrow$

## MOORE AND MEALY MACHINES:

Finite Automata may have outputs corresponding to each transition. There are two types of FSM that generate output.

- ① MEALY Machines
- ② MOORE machines.

Mealey Machine:- A mealey machine is an FSM whose output depends on the present state as well as present input.

It can be described by a 6 tuple  $(Q, E, D, \delta, \lambda, q_0)$  where



$q_0$  is the initial/state from where any input is processed ( $q_0 \in Q$ ).

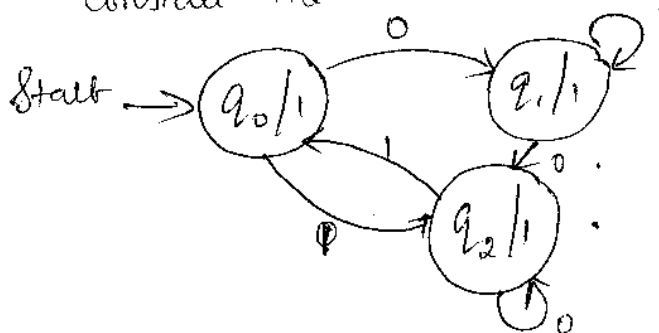
The state table of a moore machine is shown below

Present state	next state		Output
	Input = 0	Input = 1	
$\rightarrow a$	b	c	$x_2$
b	b	d	$x_1$
c	c	d	$x_2$
d	d	d	$x_3$

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

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

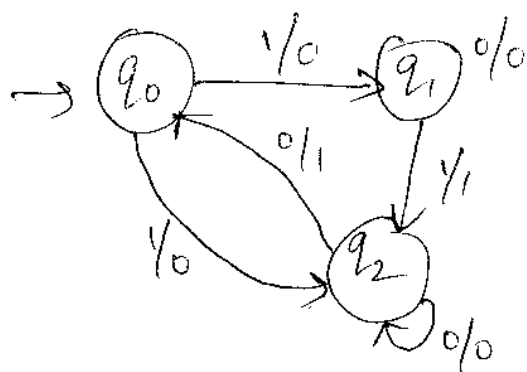
Example: Consider the moore machine given below



where output values are decided by its current state

The transition table will be

Current state	next state(s)		output ( $\lambda$ )
	0	1	
$q_0$	$q_1$	$q_2$	1
$q_1$	$q_2$	$q_1$	1
$q_2$	$q_2$	$q_0$	0



For the input string 1001 the output will be 0001. In mealy machine the length of input string is equal to length of output string.

### Moore Machine to Mealy Machine:

Input: Moore Machine

Output: Mealy Machine

Step 1: Take a blank mealy machine transition table format.

Step 2: Copy all the moore machine transition states into this table format.

Step 3: Check the present states and their corresponding outputs in the moore machine state table, if for a state  $Q_i$  output is  $m$ , copy into the output column of the

Step 4: mealy machine state table wherever  $Q_i$  appears in the next state.

### Example:

Let us consider the following moore machine.

Present State	next state		Output
	$a=0$	$a=1$	
→ a	d	b	1
b	a	d	0
c	c	c	0
d	b	a	1

Step 1 & 2:

Present State	next State			
	$a = 0$		$a = 1$	
	State	Output	State	Output
→ a	d		b	
b	a		d	
c	c		c	
d	b		a	

Step 3:

Present State	next State			
	$a = 0$		$a = 1$	
	State	Output	State	Output
⇒ a	d	1	b	0
b	a	1	d	1
c	c	0	c	0
d	b	0	a	1

Mealy Machine

vs

Moore Machine

→ Output depends both upon the present state and present input.

→ Generally it has fewer states than moore machine.

→ The value of the output function is a function of the transition and the changes, when the input logic on the present state is dummy.

Output depends only upon the present state.

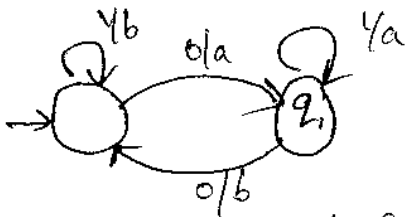
Generally it has more than mealy machines.

The value of output function is a function of the current state of the changes at the clock edges, wherever state changes occur.

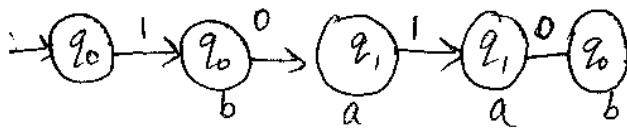


→ mealy machine react faster to input, they generally react in the same clock cycle.

Eg:



Input sequence 1010 - o/p is baab

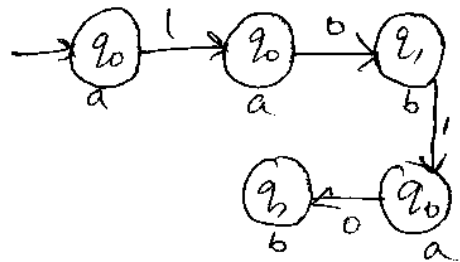


In moore machine, more logic is required to decide the output resulting in more circuit delays. They generally react one clock cycle later.

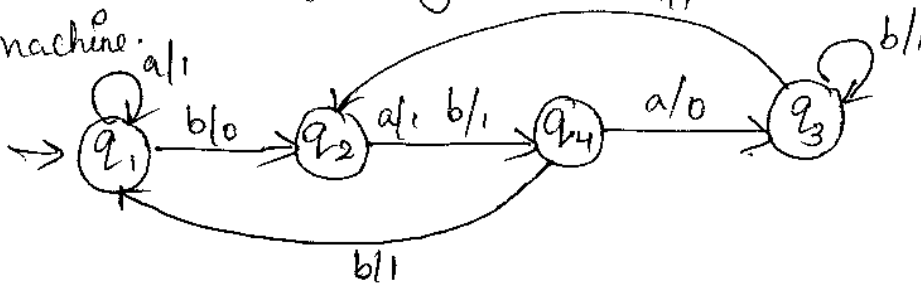
Eg:



Input sequence 1010 - o/p is acbab



Example: Convert the following mealy machine into equivalent moore machine.



Solution:

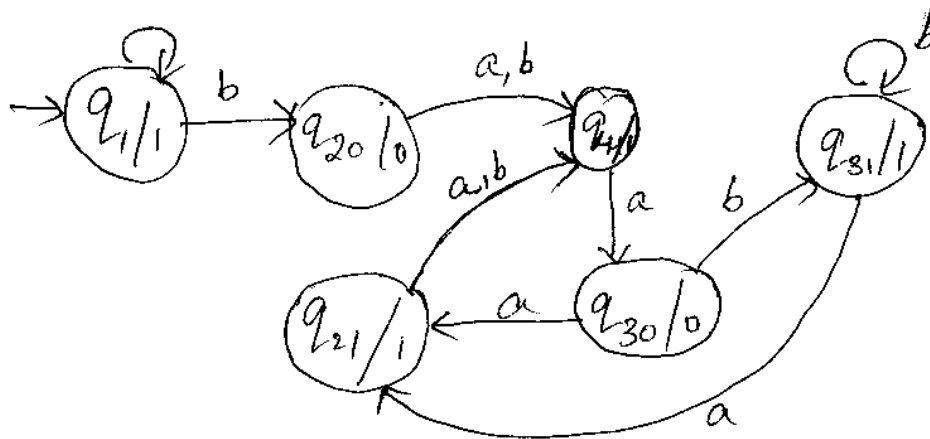
Transition table for mealy machine:-

Present State	Next State			
	a		b	
	o/p state	O/p	input state	O/p
→ q <sub>1</sub>	q <sub>1</sub>	1	q <sub>2</sub>	0
q <sub>2</sub>	q <sub>4</sub>	1	q <sub>4</sub>	1
q <sub>3</sub>	q <sub>2</sub>	1	q <sub>3</sub>	1
q <sub>4</sub>	q <sub>3</sub>	0	q <sub>1</sub>	1

## Moose Machine:

Present state	a	b	o/p
$q_1$	$q_1$	$q_{20}$	1
$q_{20}$	$q_4$	$q_4$	0
$q_{21}$	$q_4$	$q_4$	1
$q_{30}$	$q_{21}$	$q_{31}$	0
$q_{31}$	$q_{21}$	$q_{31}$	1
$q_4$	$q_{30}$	$q_1$	1

$q_2 \rightarrow q_{20} - 0$   
 $q_2 \rightarrow q_{21} - 1$   
 $q_3 \rightarrow q_{30} - 0$   
 $q_3 \rightarrow q_{31} - 1$



## Example for relay machine:

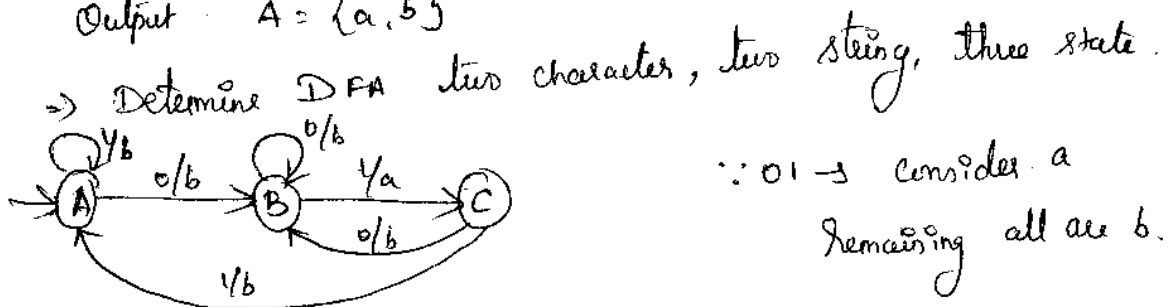
1) Construct a relay machine that prints 'a' whenever the sequence '01' is encountered in any input binary search.

Solution:

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

Output  $A = \{a, b\}$

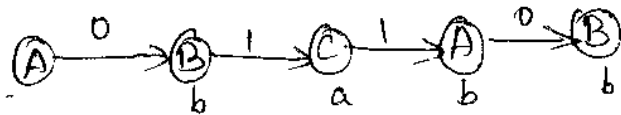
$\therefore \frac{0101001}{a} \quad \text{sequence of encounter '01' print a.}$



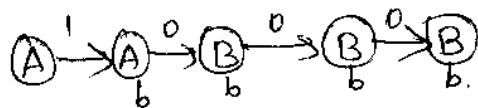
$\therefore 01 \rightarrow \text{consider a}$   
Remaining all are b.

In relay machine no final state.

Assume string 0110  
ba bb.



1000  $\rightarrow$  a should not be encountered



Example 2)

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

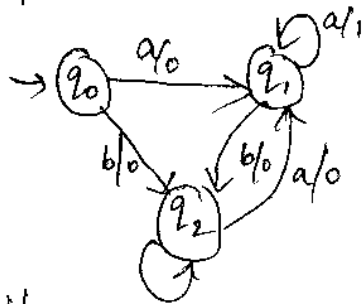
Solution: Input  $\Sigma = \{a, b\}$

Output  $\Lambda = \{0, 1\}$

String should end with either aa or bb.

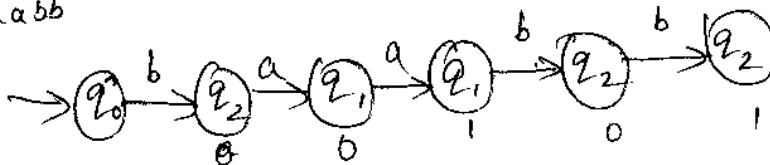
Assume that the string ends with either aa or bb if

Generate output is 1 else output is 0.

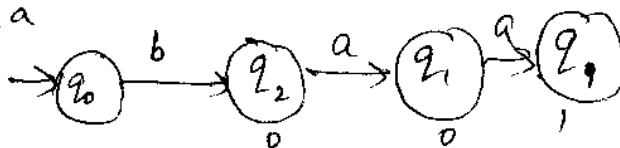


aaabaa  
bbbabaa.

eg: baabbb



eg: baa



## Examples for Moore Machines:-

1) Design a moore machine for a binary input sequence such that if it has a substring 101, the machine ~~output~~ <sup>output</sup> is A. If the input has substring 110, its output is B otherwise B. output is c.

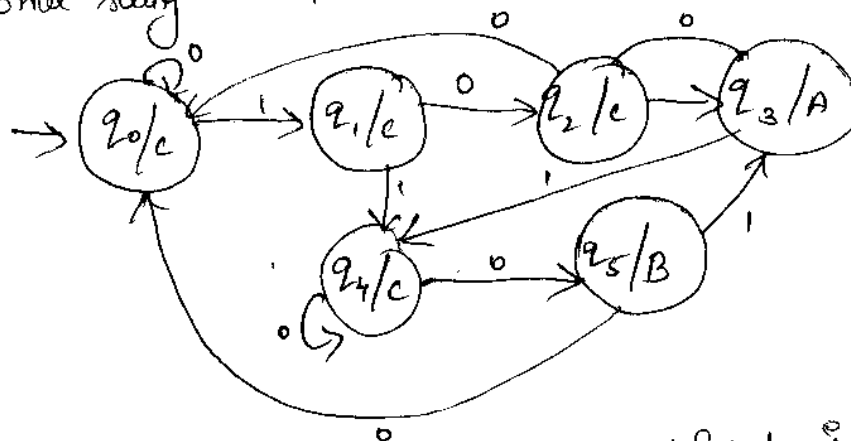
Solution:

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

input		output
101	—	A
110	—	B
---	—	C.

for designing we need to check 3 conditions.

- ⇒ if we get 101 - output is A
- ⇒ if we recognize 110 - output is B.
- ⇒ for other string - output is c.



0101 - A  
001 - C  
101 - A  
110 - B  
--- - C.

2) Design moore machine The input alphabet is  $\Sigma = (a, b)$  & the output alphabet is  $\Delta = (0, 1)$ . Run the following input sequence and find the respective output i) aabab ii) abbb iii) ababb.

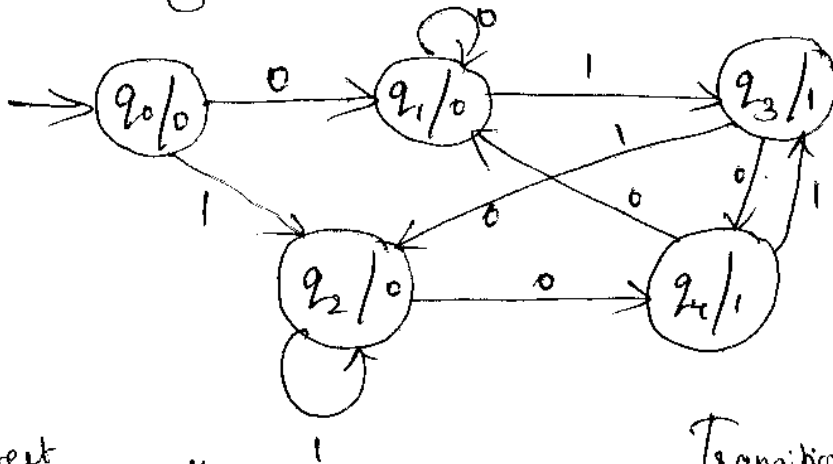
State	a	b	output
→ q <sub>0</sub>	q <sub>1</sub>	q <sub>2</sub>	0
q <sub>1</sub>	q <sub>2</sub>	q <sub>3</sub>	0
q <sub>2</sub>	q <sub>3</sub>	q <sub>4</sub>	1
q <sub>3</sub>	q <sub>4</sub>	q <sub>4</sub>	0
q <sub>4</sub>	q <sub>0</sub>	q <sub>0</sub>	0

(i).  $\rightarrow q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_4 \xrightarrow{a} q_0 \xrightarrow{b} q_2 \Rightarrow 01001$

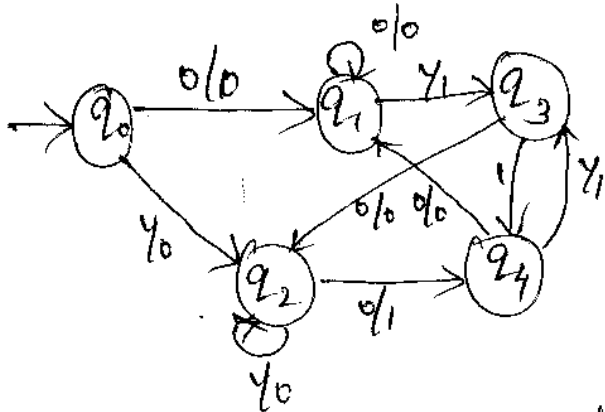
(ii)  $\rightarrow q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_3 \xrightarrow{a} q_4 \xrightarrow{b} q_0 \xrightarrow{b} q_2 \Rightarrow 00001$

(iii)  $\rightarrow q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_3 \xrightarrow{b} q_4 \xrightarrow{b} q_0 \Rightarrow 00000$

Conversion of Moore Machine to Mealy Machine



To convert  
Moore Transition Diagram



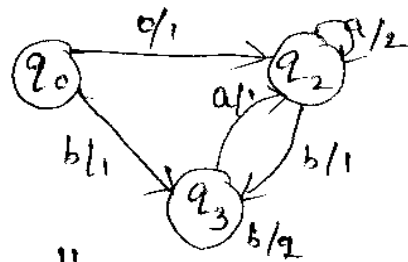
Transition Table

State	0	1	O/p
$\rightarrow q_0$	$q_1$	$q_2$	0
$q_1$	$q_1$	$q_3$	0
$q_2$	$q_2$	$q_4$	0
$q_3$	$q_1$	$q_3$	1
$q_4$	$q_2$	$q_3$	1

Mealy Transition Diagram

State	Next State			
	State	O/p	State	O/p
$\rightarrow q_0$	$q_1$	0	$q_2$	0
$q_1$	$q_1$	0	$q_3$	1
$q_2$	$q_2$	0	$q_4$	0
$q_3$	$q_1$	1	$q_3$	1
$q_4$	$q_2$	1	$q_3$	1

Convert Melay to Moore:-

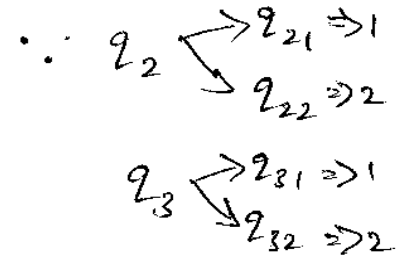


⇒

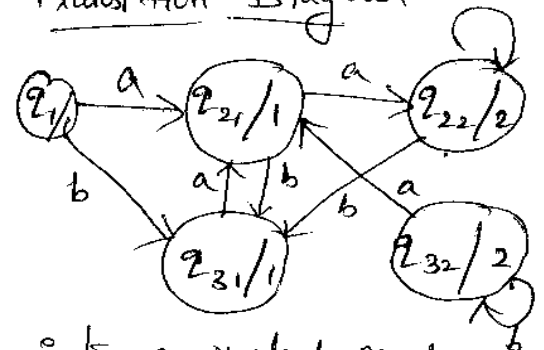
Present State	a		b	
	State	O/P	State	O/P
→ q <sub>1</sub>	q <sub>2</sub>	1	q <sub>3</sub>	1
q <sub>2</sub>	q <sub>2</sub>	2	q <sub>3</sub>	1
q <sub>3</sub>	q <sub>2</sub>	1	q <sub>3</sub>	2

to convert moore transition table

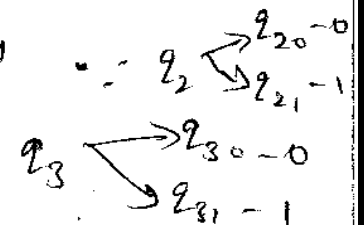
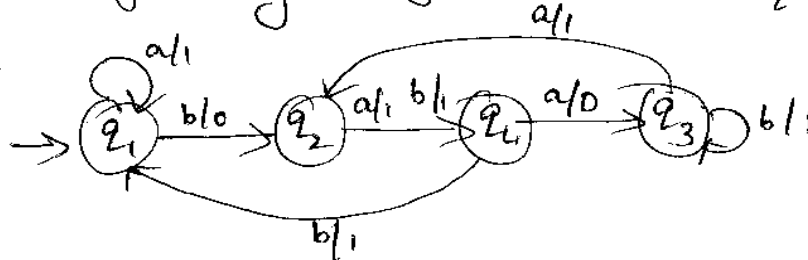
State	a	b	O/P
→ q <sub>1</sub>	q <sub>21</sub>	q <sub>31</sub>	1
q <sub>21</sub>	q <sub>22</sub>	q <sub>31</sub>	1
q <sub>22</sub>	q <sub>22</sub>	q <sub>31</sub>	2
q <sub>31</sub>	q <sub>21</sub>	q <sub>32</sub>	1
q <sub>32</sub>	q <sub>21</sub>	q <sub>32</sub>	2



Moore Transition Diagram



Convert the following mealy machine into equivalent moore machine.



Solution:

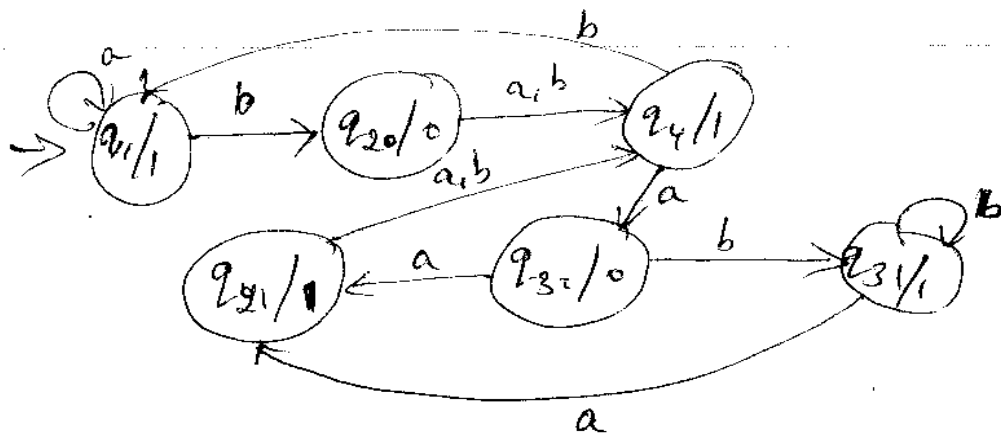
Transition Table:

Present State	Next State			
	a		b	
	State	O/P	State	O/P
→ q <sub>1</sub>	q <sub>1</sub>	1	q <sub>2</sub>	0
q <sub>2</sub>	q <sub>4</sub>	1	q <sub>4</sub>	1
q <sub>3</sub>	q <sub>2</sub>	1	q <sub>3</sub>	1
q <sub>4</sub>	q <sub>3</sub>	0	q <sub>1</sub>	1

⇒

Present State	a	b	O/P
→ q <sub>1</sub>	q <sub>1</sub>	q <sub>2</sub>	1
q <sub>20</sub>	q <sub>4</sub>	q <sub>4</sub>	0
q <sub>21</sub>	q <sub>4</sub>	q <sub>4</sub>	1
q <sub>30</sub>	q <sub>21</sub>	q <sub>31</sub>	0
q <sub>31</sub>	q <sub>23</sub>	q <sub>31</sub>	1
q <sub>4</sub>	q <sub>30</sub>	q <sub>1</sub>	1

45



\* ————— \*

q1b