Automata Theory

Automata theory is the study of abstract ***computing devices (machines)***. In 1930s, **Turing** studied an abstract machine (***Turing machine***) that had all the capabilities of today's computers. Turing's goal was to describe precisely the boundary between what ***a computing machine could do and what it could not do.***

In 1940s and 1950s, simpler kinds of machines (**finite automata**) were studied.**Chomsky** began the study of **formal grammars** that have close relationships to abstract automata and serve today as the basis of some important software components.

Why Study Automata?

Automata theory is the ***core of computer science***. Automata theory presents ***many useful models for software and hardware***.

1. Software for designing and checking the behaviour of digital circuits.

2. The 'lexical analyzer" of a typical compiler, that is, the compiler component that breaks the input text into logical units, such as identifiers, keywords, and punctuation.

3. Software for scanning large bodies of text, such as collections of Web pages, to find occurrences of words, phrases, or other patterns.

4. Software for verifying systems of all types that have a finite number of distinct states, such as communications protocols or protocols for secure exchange of information

Automata are essential for **the study of limits of computation**. Some of them are

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

2. what can a computer do efficiently? This study is called "intractability" and the problems that can be solved by a computer in a polynomial amount of time are called "tractable"

Central Concepts of Automata Theory:

Alphabet:

An alphabet is a finite, non empty set of symbols. We use the symbol Σ for an alphabet.

Ex:

- Σ = {0,1} - binary alphabet
- Σ = {a,b,c,…,z} - lowercase letters
- The set of ASCII characters is an alphabet

String:

A **string** is a sequence of symbols chosen from some alphabet. A string sometimes is called as **word**.

- 01101 is a string from the alphabet $\Sigma = \{0,1\}$.
  - Some other strings: 11, 010, 1, 0

- The **empty string**, denoted as ε, is a string of zero occurrences of symbols.

Length of string:

The length of a string is the number of symbols in the string.The length of string w is denoted as |w|

Ex: $|ab| = 2$ $|b| = 1$ $|\varepsilon| = 0$

Concatenation of strings:

If **x** and **y** are strings then **xy** represents their concatenations.

Ex: If **x**=abc and **y**=de then **xy** = abcde

Powers of an alphabet:

If Σ is an alphabet, the set of all strings of a certain length from the alphabet by using an exponential notation. We define Σ^k is the set of strings of length k from Σ.

Ex: Let $\Sigma = \{0,1\}$.

Then $\Sigma^0 = \{\varepsilon\}$ $\Sigma^1 = \{0,1\}$ $\Sigma^2 = \{00,01,10,11\}$

Note: The set of all strings over an alphabet is denoted by Σ^*.

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

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \dots$ - set of nonempty strings

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

Language:

A set of strings that are chosen from Σ* is called as a **language** i.e., if Σ is an alphabet, and **L ⊆ Σ*** , then L is a **language** over Σ.

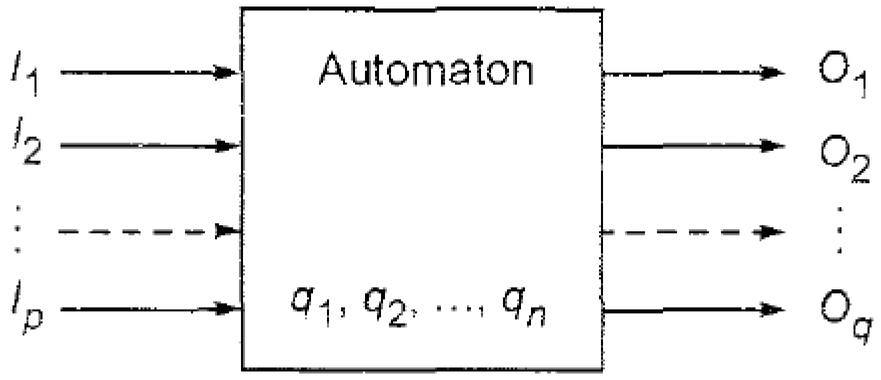
Ex:

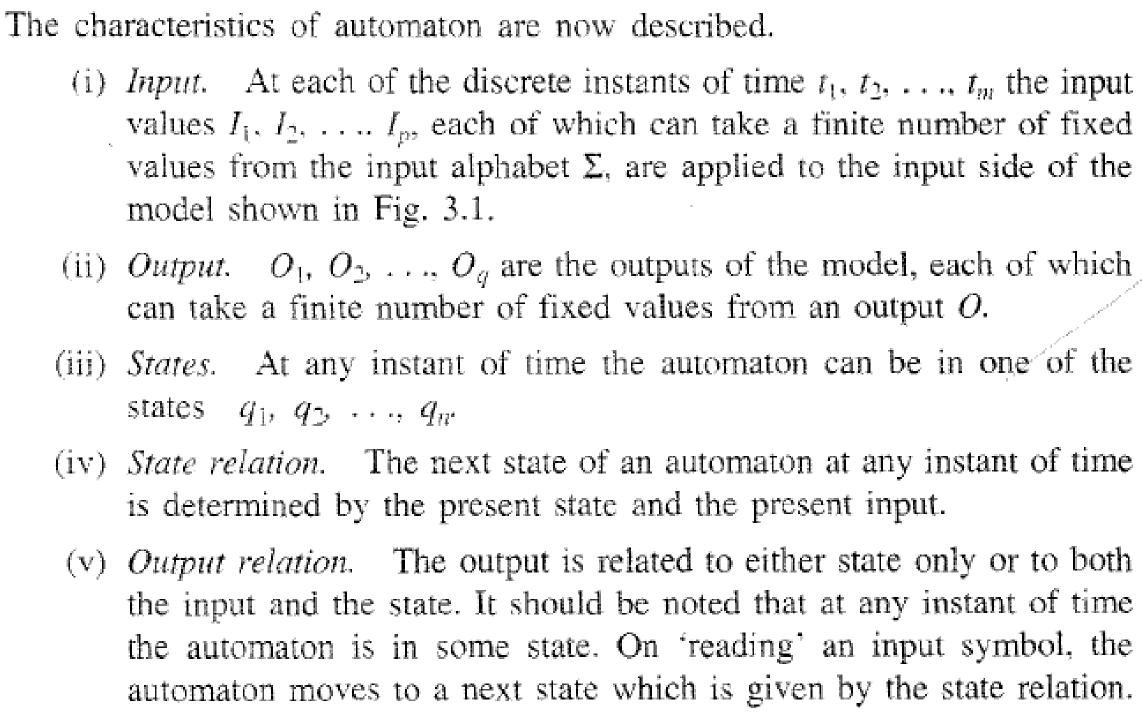
- The language of all strings consisting of n 0’s followed by n 1’ for some n≥0 : {Ɛ, 01, 0011, 000111, …}
- Σ* is a language
- Empty set is a language. The empty language is denoted by Φ
- The set {Ɛ} is a language, {Ɛ} is not equal to the empty language.

DEFINITION OF AN AUTOMATON:

In general, an automaton is defined as a system where energy, materials and information are transformed. transmitted and used for performing some functions without direct participation of man. Examples are automatic machine tools, automatic packing machines, and automatic photo printing machines.

In computer science the term 'automaton' means 'discrete automaton' and is defined in a more abstract way as shown in the following figure.





The characteristics of automaton are now described.

(i) *Input.* At each of the discrete instants of time $t_1, t_2, \ldots, t_m$ the input values $I_1, I_2, \ldots, I_p$, each of which can take a finite number of fixed values from the input alphabet Σ, are applied to the input side of the model shown in Fig. 3.1.

(ii) *Output.* $O_1, O_2, \ldots, O_q$ are the outputs of the model, each of which can take a finite number of fixed values from an output O.

(iii) *States.* At any instant of time the automaton can be in one of the states $q_1, q_2, \ldots, q_n$.

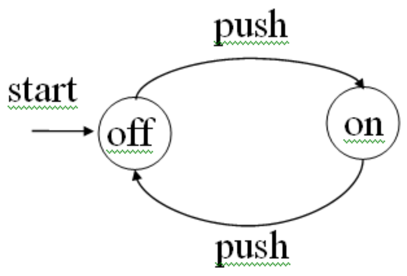
(iv) *State relation.* The next state of an automaton at any instant of time is determined by the present state and the present input.

(v) *Output relation.* The output is related to either state only or to both the input and the state. It should be noted that at any instant of time the automaton is in some state. On 'reading' an input symbol, the automaton moves to a next state which is given by the state relation.

Finite Automata

Finite automata are *finite collections of states with transition rules* that take you from one state to another. **So a finite automaton has finite number of states and its control moves from state to state in response to external inputs**. The *purpose of a state* is to remember the relevant portion of the history. Since there are only a *finite number of states*, the entire history cannot be remembered. So the system must be designed carefully to remember what is important and forget what is not. The advantage of having only a finite number of states is that we can implement the system with a fixed set of resources.

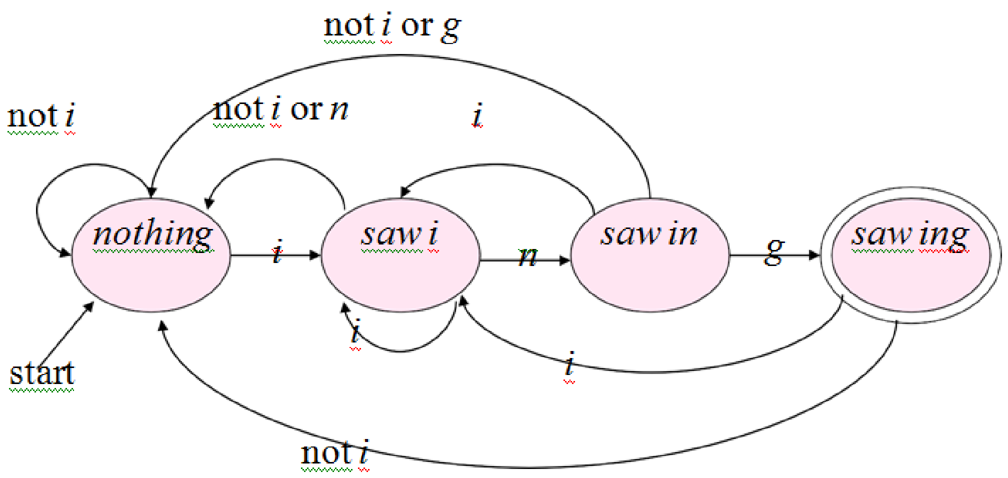
A simple Finite automaton: on/off switch



In a **finite automaton**:

- **States** are represented by **circles**.
- **Accepting (final) states** are represented by **double circles**.
- One of the states is a **starting state**.
- **Arcs** represent **state transitions** and **labels on arcs** represent **inputs** (external influences) causing trasitions.

Example 2: Finite automata recognizing strings ending with “ing”



There are two different types of finite automata. They are

1. Deterministic Finite Automata (DFA)

2. Non- Deterministic Finite Automata (NFA)

<u>Deterministic Finite Automata (DFA):</u>

- A deterministic finite automaton (DFA) is a 5-tuple (Q,Σ, δ, q_0, F) where
  - Q is a finite non empty set of states
  - Σ is an input alphabet
  - δ: $Q \times \Sigma \to Q$ is a transition function
  - $q_0 \in Q$ is the initial state
  - $F \subseteq Q$ is a set of accepting states (or final states).

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.

Ex: The DFA accepting all strings over {0, 1} with a substring 01 is given as follows

A = (Q, Σ, δ, q_0, F) where

- Q ={q_0,q_1,q_2}
- Σ={0,1}
- q_0 Є Q is the initial state
- F={q_2} is a set of final states
- The transition function δ is given as follows.

δ(q_0, 0) = q_1

δ(q_0, 1) = q_0

δ(q_1, 0) = q_1

δ(q_1, 1) = q_2

δ(q_2, 0) = q_2

δ(q_2, 1) = q_2

Simpler notations for DFA:

There are two preferred notations for describing the automata. They are

1. Transition diagram (or) transition graph (or) transition system

2. Transition table

Transition diagram:

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

a) For each state in Q there is a node.

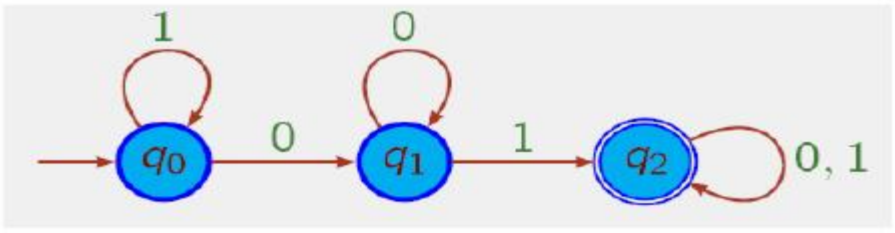
b) For each state q in Q and each input symbol a in Σ, let $\delta(q, a) = p$. Then the transition diagram has an arc from node q to node p, labeled a. If there are several input symbols that cause transitions from q to p, then the transition diagram can have one arc, labeled by the list of these symbols.

c) There is an arrow into the start state q_0, labeled *Start*. This arrow does not originate at any node.

d) Nodes corresponding to accepting states (those in F) are marked by a double circle. States not in F have a single circle.

Ex:

The transition diagram for the DFA accepting all strings over {0,1}with a substring 01 is given as follows.

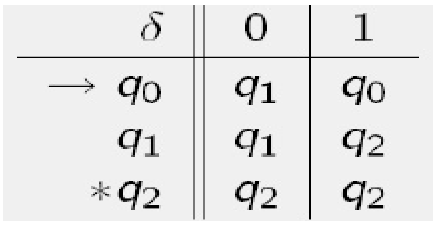


Transition table:

A transition table is a tabular representation of a transition function δ that takes two arguments and returns a value. The rows of the table correspond to the states, and the columns correspond to the inputs. The entry for the row corresponding to state q and the column corresponding to input a is the state δ(q,a). Also the start state is marked with an arrow, and the accepting states are marked with a star.

Ex:

The transition table for the DFA accepting all strings over {0,1}with a substring 01 is given as follows.



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

Processing of strings by DFA:

Suppose that $a_1a_2\ldots\ldots.a_n$ is a given input string and q_0 is the initial state of the DFA.

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

$\delta(q_1, a_2) = q_2$

$\delta(q_2, a_3) = q_3$

……………..

……………..

$\delta(q_{n-1}, a_n) = q_n$

Now after processing the last input symbol a_n, the state of the DFA is q_n. If q_n is a member of F, then the input string $a_1a_2\ldots\ldots a_n$ is accepted by the DFA otherwise it is rejected. The language of the DFA is the set of all strings that the DFA accepts.

Extended Transition function:

The extended transition function is a function that takes a state q and a string w and returns a state p - the state that the automaton reaches when starting in state q and processing the sequence of inputs w. We denote it by $\hat{\delta}$ and is defined by the induction on the length of the input string, as follows.

Basis:

$$\hat{\delta}(q,\varepsilon) = q$$

Induction:

Suppose w is a string of the form xa i.e., a is the last symbol of w, and x is the string consisting of all but the last symbol. Then

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

Language of the DFA:

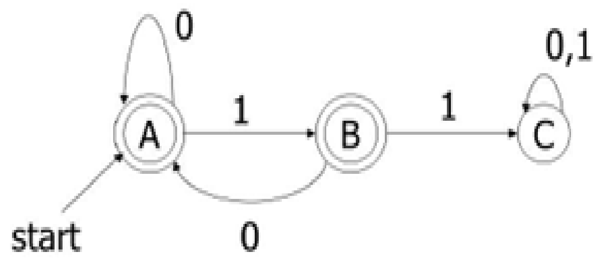
The language of the DFA M = (Q, Σ, δ, q_0, F) is denoted as L(M) and is defined as

$$L(M) = \left\{ w \mid \hat{\delta}(q_0,w)\, is\, in\ F \right\}$$

Note : A language L is said to be a regular language if L=L(M) for some DFA M.

Problems:

1. consider the following DFA.



Determine whether the string 0100 is accepted by the DFA or not.

Sol: To determine whether the string 0100 is accepted by the DFA or not, compute

$\hat{\delta}(A,0100)$

Now we have

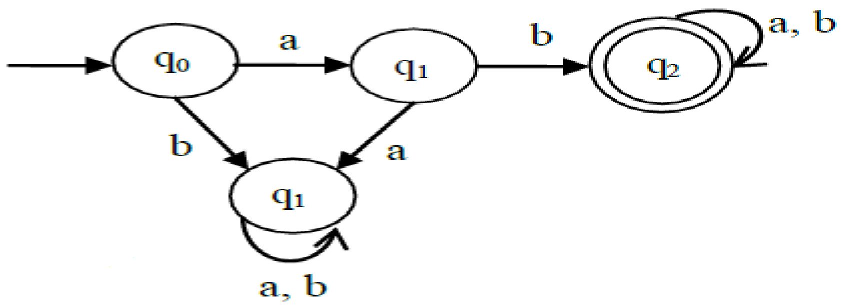
- $\hat{\delta}(A,\varepsilon) = A$
- $\hat{\delta}(A,0) = \delta(\hat{\delta}(A,\varepsilon),0) = \delta(A,0) = A$
- $\hat{\delta}(A,01) = \delta(\hat{\delta}(A,0),1) = \delta(A,1) = B$
- $\hat{\delta}(A,010) = \delta(\hat{\delta}(A,01),0) = \delta(B,0) = A$
- $\hat{\delta}(A,0100) = \delta(\hat{\delta}(A,010),0) = \delta(A,0) = A$

Since $\hat{\delta}(A,0100)=A$ and A is a final state, the string 0100 is accepted by this DFA.

Design of DFA:

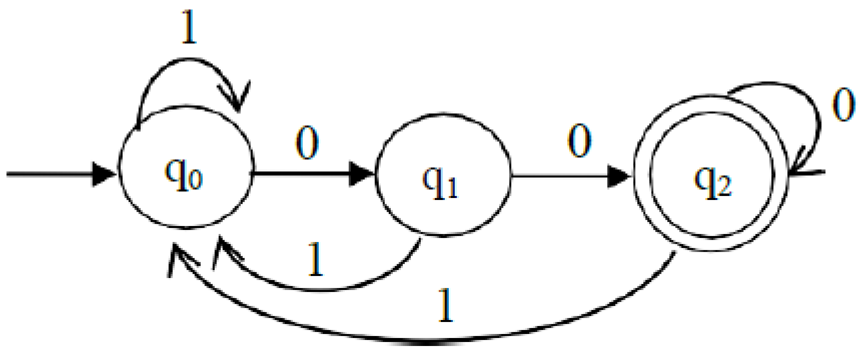
Ex: Design a DFA that accepts the set of all strings over alphabet {a,b} beginning with ‘ab’

Sol:



Ex: Design a DFA that accepts the set of all strings over alphabet {0,1} ending with ‘00’

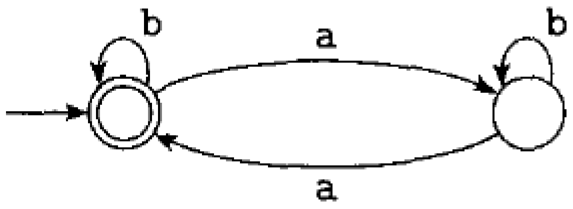
Sol:



Ex: Design a DFA that accepts the following language L over alphabet {a,b}

L ={ w | w is of even number of a’s}

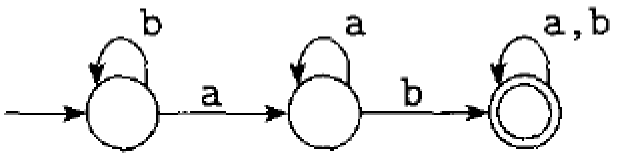
Sol:



Ex: Design a DFA that accepts the following language L over alphabet {a,b}

L ={ w | w contains ab as a substring}

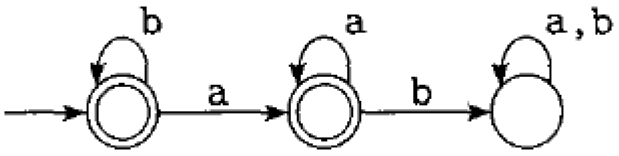
Sol:



Ex: Design a DFA that accepts the following language L over alphabet {a,b}

L ={ w | w does not contain ab as a substring}

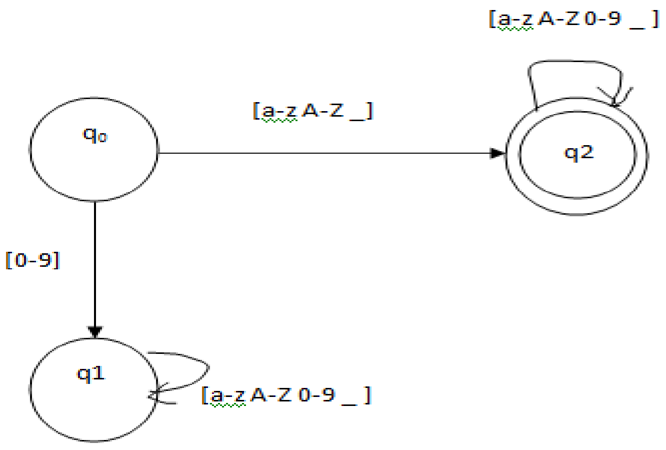
Sol:



Ex: construct a DFA that accepts an identifier of a 'C' Programming language.

Sol: An identifier of a 'C' Programming language is formed with letters (uppercase [A-Z] or lowercase [a-z]), alphabets([0-9]) and underscore character. The identifier must begin with letter or underscore.

Thus the DFA accepting the identifier of a 'C' Programming language is given as follows.(without considering the keywords)

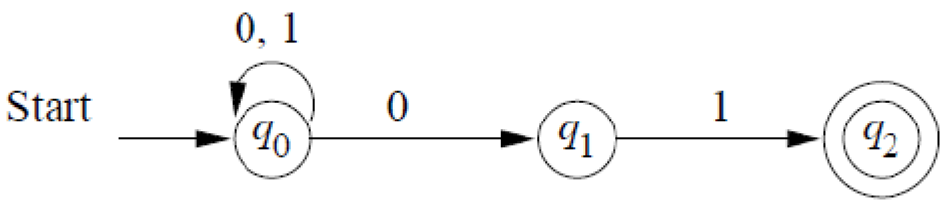


Non-deterministic Finite Automata(NFA):

- A Non-deterministic Finite Automaton (NFA) is a 5-tuple (Q,Σ, δ, q_0, F) where
  - Q is a finite non empty set of states
  - Σ is an input alphabet
  - δ: $Q \times \Sigma \rightarrow 2^Q$ is a transition function
  - $q_0 \in Q$ is the initial state
  - $F \subseteq Q$ is a set of accepting states (or final states).

So NFA can have zero, one or multiple number of transitions to next states from a given state on a given input symbol.

Ex: The NFA that accepts all strings over {0,1} ending with 01 is given as follows.



The above NFA can be specified as M={{q_0,q_1,q_2}, {0,1},δ, q_0,{q_2} }

Where δ is given by the following table

	0	1
→ q_0	{ q_0, q_1 }	{ q_0 }
q_1	ϕ	{ q_2 }
* q_2	ϕ	ϕ

Extended Transition function:

The extended transition function is denoted as $\hat{\delta}$. It 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.

It is defined by the induction on the length of the input string, as follows.

Basis:

$$\hat{\delta}(q,\varepsilon)=\{q\}$$

Induction:

Suppose w is a string of the form xa i.e., a is the last symbol of w, and x is the string consisting of all but the last symbol.Also suppose that $\hat{\delta}(q,x)=\{p_1, p_2,\ldots\ldots, p_k\}$ and

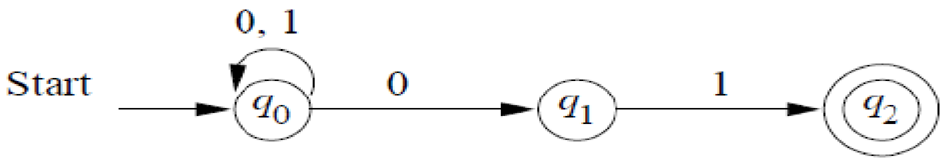
$$\bigcup_{i=1}^{k}\delta(p_i,a)=\{r_1,r_2,\ldots\ldots\ldots,r_m\}$$

Then

$$\hat{\delta}(q,w)=\{r_1,r_2,\ldots\ldots,r_m\}$$

Processing of strings by NFA:

Consider the following NFA.



Now the processing of input string 00101 by the above NFA by using extended transition function is given below.

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

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

3. $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$.

4. $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$.

5. $\hat{\delta}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$.

6. $\hat{\delta}(q_0, 00101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$.

Since one of the states among $\{q_0, q_2\}$ is a final state, the string is accepted by the NFA.

Language of the NFA:

Let M = (Q, Σ, δ, q_0, F) be the given NFA. Then the language of NFA M is denoted as L(M) and is defined as

$$L(M) = \{ w | \hat{\delta}(q_0, w) \cap F \neq \phi \}$$

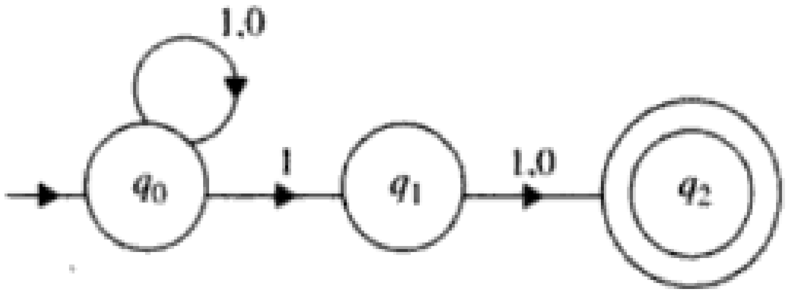
i.e., L(M) is the set of strings w in Σ* such that $\hat{\delta}(q_0, w)$ contains at least one accepting state.

<u>Design of NFA:</u>

Ex: Design an NFA that accepts all the strings over {0,1} whose second last symbol is 1

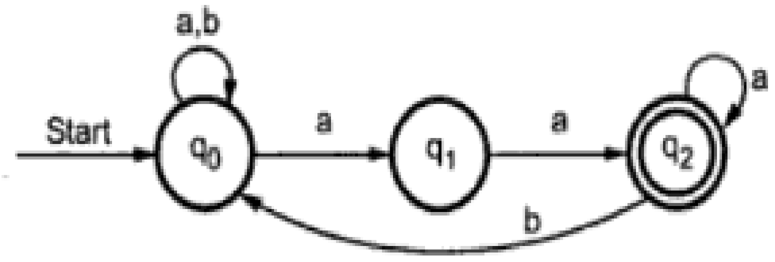
Sol:

The required NFA is



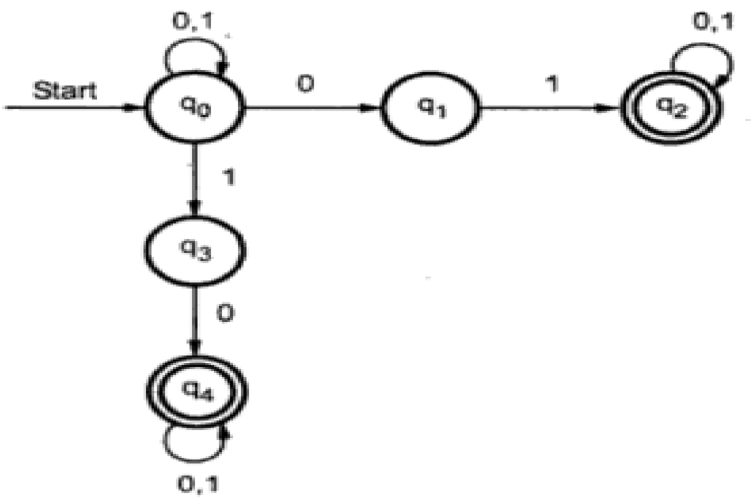
Ex: Design an NFA that accepts all the strings over {a,b} ending with 'aa'

Sol:



Ex: Design an NFA that accepts the strings over Σ={0,1} containing either '01' or '10'

Sol:



Equivalence of NFA and 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.

Procedure to convert NFA to DFA:

Let M = (Q, Σ, δ, q_0, F) be an NFA accepting language L. We construct an equivalent DFA M'= (Q', Σ, δ', q_0', F') accepting the same language L as follows.

1) Q'=2^Q (where 2^Q is the power set of Q)

2) q_0'={q_0} is the initial state

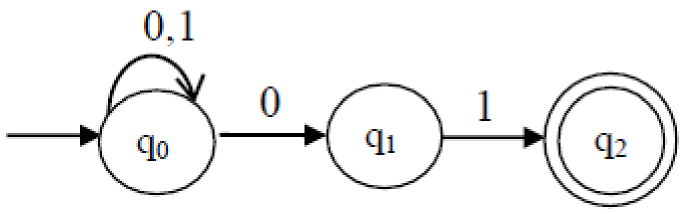
3) F' is the set of all subsets of Q containing an element of F.

4) δ' is defined as follows.

$$\delta'(\{q_1,q_2,\ldots q_n\},a)=\delta(q_1,a)\cup\delta(q_2,a)\cup\ldots\ldots\cup\delta(q_n,a)$$

Problems:

1. Convert the following NFA to the equivalent DFA.



Sol: for the given NFA, M = (Q, Σ, δ, q_0, F),

Q = {q_0,q_1,q_2}

Σ = {0,1}

q_0 Є Q is the initial state

F ={q2} is the set of Final states and

The transition function δ is defined as follows.

δ

$Q \backslash \Sigma$	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$
$* q_2$	ϕ	ϕ

Now the equivalent DFA to the given NFA, M'= (*Q'*, Σ, δ', q_0', *F'*), is given as follows.

a) $Q' = \{\phi, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

b)) q_0'={q_0} is the initial state

c) $F' = \{\{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

d) the transition function δ' is given as follows.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

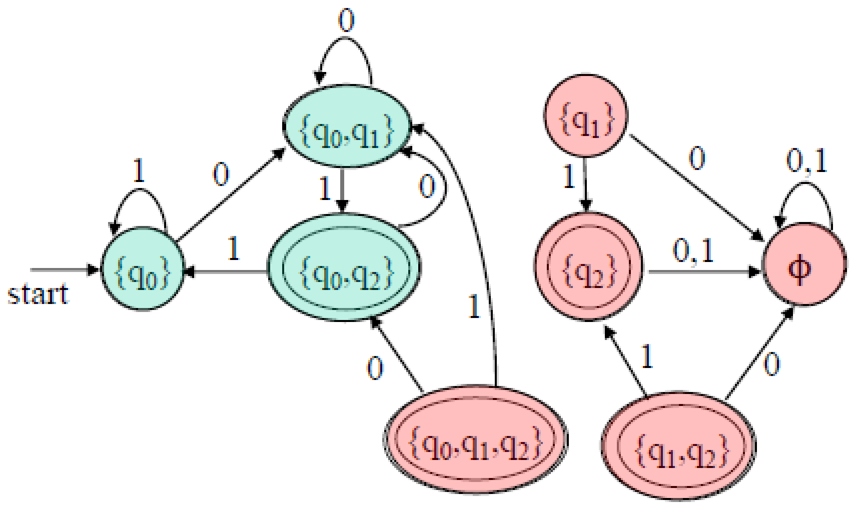
$$\begin{aligned}\delta'(\{q_0,q_1,q_2\},0)&=\delta(\{q_0\},0)\cup\delta(\{q_1\},0)\cup\delta(\{q_2\},0)\\&=\{q_0,q_1\}\cup\phi\cup\phi=\{q_0,q_1\}\end{aligned}$$

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

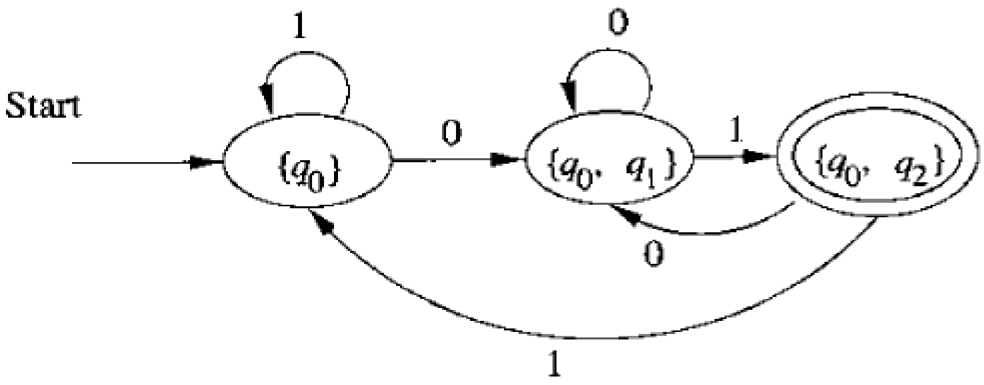
The transition table for δ' is given as follows.

Q \ Σ	**0**	**1**
ϕ	ϕ	ϕ
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	ϕ	$\{q_2\}$
$*\{q_2\}$	ϕ	ϕ
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$*\{q_1, q_2\}$	ϕ	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

The equivalent DFA is



Here the states $\{q_0\}$, $\{q_0,q_1\}$ and $\{q_0,q_2\}$ are only reachable from the initial state $\{q_0\}$. All the remaining states are not reachable from the initial state. So the equivalent DFA for the given NFA is given as follows.



Finite Automata with Epsilon-transitions:

Definition:

Epsilon-Transitions(ε – Transitions): Transitions that takes place on an empty input i.e., no input symbol at all are called ε- Transitions(Transitions that takes place without reading any input symbol are called ε – Transitions)

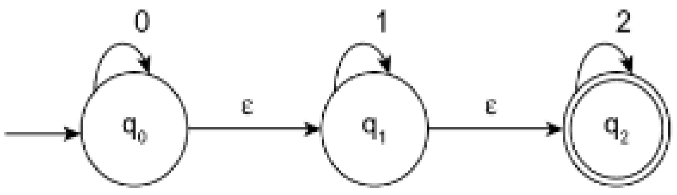
We can extend the NFA by allowing ε- transition, which are called as NFA with ε – Transitions (or) **ε – NFA.**

Definition:

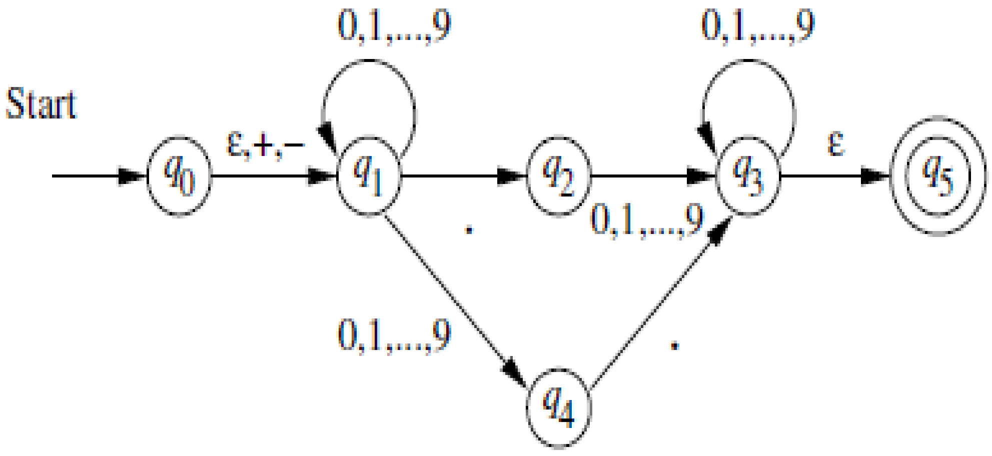
- An ε-NFA is a 5-tuple (Q,Σ, δ, q_0, F) where
    - Q is a finite non empty set of states
    - Σ is an input alphabet
    - δ: $Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$ is a transition function
    - $q_0 \in Q$ is the initial state
    - $F \subseteq Q$ is a set of accepting states (or final states).

Note: the empty string ε is not a member of input alphabet Σ.

Ex 1:



Ex 2:



Eliminating ε-Transitions:

Definition:ε –closure of a state:

ε –closure of a state q is the set of all states that can be reached from q along any path whose arcs are all labelled with ε. It is denoted as ECLOSE (q) (or) ε-closure (q).

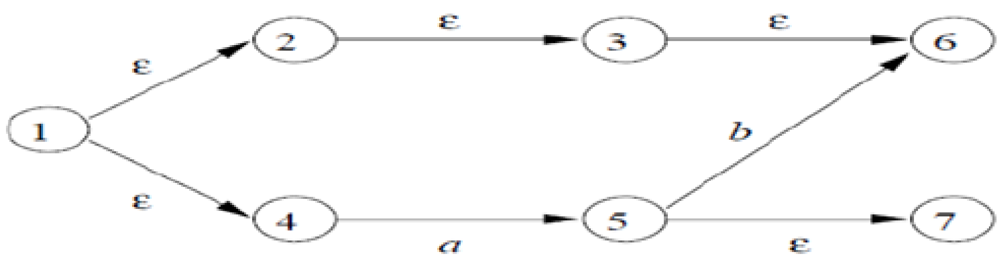
Inductive definition of ECLOSE (q):

Basis: state q is in ECLOSE (q) i.e., q ∈ ECLOSE (q)

Induction: If p is in ECLOSE (q) and there is a transition from state p to state r labelled ε, then r is in ECLOSE (q) i.e.,

If p ∈ ECLOSE (q) and r ∈ δ (p, ε), then r ∈ ECLOSE (q)

Ex:

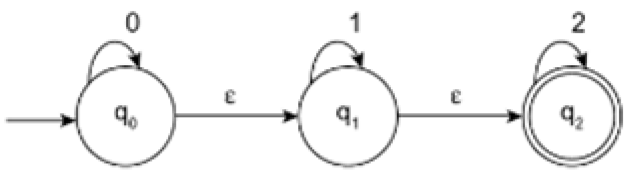


Here ECLOSE(1) = {1,2,3,4,6}, ECLOSE(2) = {2,3,6}, ECLOSE(3) = {3,6}

ECLOSE (4) = {4}, ECLOSE (5) = {5,7}, ECLOSE(6) = {6} and

ECLOSE (7) = {7}

Ex:



Here ε-closure(q0) = {q0, q1, q2} ,

ε-closure(q1) = {q1, q2} and

ε-closure(q2) = {q2}

Theorem: A language L is accepted by some ε-NFA if and only if L accepted by some DFA.

Procedure to Convert ε-NFA to DFA:

Given an ε-NFA $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$ accepting the language L. Then the equivalent DFA

$D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ accepting the same language is given as follows.

1. Q_D is the set of all subsets of Q_E i.e., $Q_D = \{ S \mid S \subseteq Q_E\}$

2. q_D= ECLOSE(q_0)

3. $F_D = \{ S \subseteq Q_E \mid S \cap F_E \neq \phi\}$

4. $\delta_D(S,a)$ is computed for all a in Σ and sets S in Q_D as follows.

For every $S \subseteq Q_N$ and $a \in \Sigma$

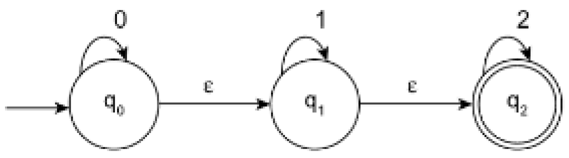
$$\delta_D(S,a) = \bigcup_{p \in S} ECLOSE(\delta_N(p,a))$$

Note:

1. In the above procedure, include those states that are reachable from the initial state in the DFA.

2. Before converting, find the ε-closure of each state of the given NFA.

Ex: Eliminate the ε- transitions from the following NFA (or) convert the following ε-NFA to DFA.



Sol:

Here for the given ε-NFA,

ε-closure(q0) = {q0, q1, q2}
ε-closure(q1) = {q1, q2}
ε-closure(q2) = {q2}

The equivalent DFA, D = (Q', Σ, δ', q_0', F'), is given as follows.

1. $Q' = \{\phi, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

2. q_0'= ε-closure(q0) = {q0, q1, q2} .

3. $F' = \{\{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

4. δ' is obtained as follows. (Consider only that are reachable from the initial state

q_0'= ε-closure(q0) = {q0, q1, q2})

δ'({q0, q1, q2}, 0) = ε-closure{δ((q0, q1, q2), 0)}

= ε-closure{δ(q0, 0) ∪ δ(q1, 0) ∪ δ(q2, 0)}

= ε-closure{q0}

= {q0, q1, q2}

δ'({q0, q1, q2} , 1) = ε-closure{δ((q0, q1, q2), 1)}

= ε-closure{δ(q0, 1) ∪ δ(q1, 1) ∪ δ(q2, 1)}

= ε-closure{q1}

= {q1, q2}

δ'({q0, q1, q2}, 2) = ε-closure{δ((q0, q1, q2), 2)}

= ε-closure{δ(q0, 2) ∪ δ(q1, 2) ∪ δ(q2, 2)}

= ε-closure{q2}

= {q2}

δ'({q1, q2} , 0) = ε-closure{δ((q1, q2), 0)}

= ε-closure{δ(q1, 0) ∪ δ(q2, 0)}

= ε-closure{φ}

= φ

δ'({q1, q2}, 1) = ε-closure{δ((q1, q2), 1)}

= ε-closure{δ(q1, 1) ∪ δ(q2, 1)}

= ε-closure{q1}

= {q1, q2}

δ'({q1, q2}, 2) = ε-closure{δ((q1, q2), 2)}

= ε-closure{δ(q1, 2) ∪ δ(q2, 2)}

= ε-closure{q2}

= {q2}

δ'({q2}, 0) = ε-closure{δ(q2, 0)}

= ε-closure{φ}

= φ

δ'({q2} , 1) = ε-closure{δ(q2, 1)}

= ε-closure{φ}

= φ

δ'({q2} , 2) = ε-closure{δ(q2, 2)}

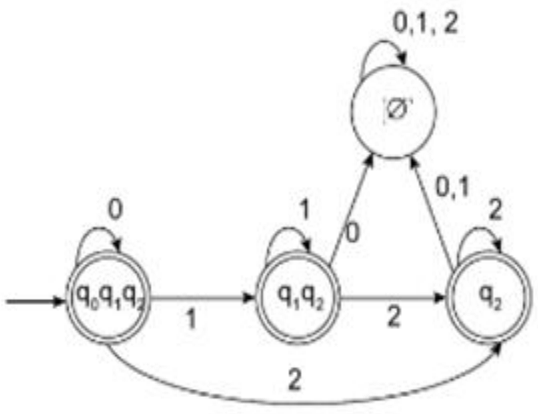
= {q2}

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

$$\begin{aligned}\delta'(\phi,1) &= \varepsilon - closure\{\delta(\phi,1)\}\\ &= \varepsilon - closure(\phi)\\ &= \phi\end{aligned}$$

$$\begin{aligned}\delta'(\phi,2) &= \varepsilon - closure\{\delta(\phi,2)\}\\ &= \varepsilon - closure(\phi)\\ &= \phi\end{aligned}$$

Here the only states that are reachable from the initial state are {q0,q1,q2} ,{q1,q2} ,{q2} and Φ only. Remaining all the states in Q' are not reachable from the initial state. Neglecting these states , the equivalent DFA for the given ε-NFA is



<u>**Minimization of Finite Automata:**</u>

we construct an automaton with the minimum number of states equivalent to a given automaton M.

<u>Definition:</u>

Two states ql and q2 are equivalent (denoted by q1 == q2) if both δ(q1, x) and δ(q2, x) are final states or both of them are non final states for all x Є Σ*.

<u>Definition:</u>

Two states ql and q2 are k-equivalent (k≥0) if both δ(q1, x) and δ(q2, x) are final states or both of them are non final states for all strings of length k or less.

In particular, any two final states are 0-equivalent and any two non final states are also 0-equivalent.

<u>Note:</u> The partition of Q under the k-equivalence relation is denoted as $\prod_k$

<u>Definition:</u>

Two states ql and q2 are (k+1)-equivalent if

i) they are k-equivalent

ii) δ(q1, x) and δ(q2, x) are also k-equivalent for every x Є Σ.

<u>**Construction of minimum Automata:**</u>

Step 1 (Construction of π_0). By definition of 0-equivalence, $\pi_0 = \{ Q_1^0, Q_2^0 \}$ where Q_1^0 is the set of all final states and $Q_2^0 = Q - Q_1^0$.

Step 2 (Construction of π_{k+1} from π_k). Let Q_i^k be any subset in π_k. If q_1 and q_2 are in Q_i^k, they are $(k + 1)$-equivalent provided $\delta(q_1, a)$ and $\delta(q_2, a)$ are k-equivalent. Find out whether $\delta(q_1, a)$ and $\delta(q_2, a)$ are in the same equivalence class in π_k for every $a \in \Sigma$. If so, q_1 and q_2 are $(k + 1)$-equivalent. In this way, Q_i^k is further divided into $(k + 1)$-equivalence classes. Repeat this for every Q_i^k in π_k to get all the elements of π_{k+1}.

Step 3 Construct π_n for $n = 1, 2, \ldots$ until $\pi_n = \pi_{n+1}$.

Step 4 (Construction of minimum automaton). For the required minimum state automaton, the states are the equivalence classes obtained in step 3, i.e. the elements of π_n. The state table is obtained by replacing a state q by the corresponding equivalence class $[q]$.

Ex: construct a Minimum state Automata equivalent to the DFA whose transition table is given as follows.

State	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_4	q_3
q_2	q_4	q_3
$\textcircled{q_3}$	q_5	q_6
$\textcircled{q_4}$	q_7	q_6
q_5	q_3	q_6
q_6	q_6	q_6
q_7	q_4	q_6

Sol:

Since any two final states are 0-equivalent and any two non final states are also 0-equivalent, we have

$$\pi_0 = \{\{q_3, q_4\}, \{q_0, q_1, q_2, q_5, q_6, q_7\}\}$$

Now we have

q_3 is 1-equivalent to q_4. So, $\{q_3, q_4\} \in \pi_1$.

q_0 is not 1-equivalent to q_1, q_2, q_5 but q_0 is 1-equivalent to q_6.

Hence $\{q_0, q_6\} \in \pi_1$. q_1 is 1-equivalent to q_2 but not 1-equivalent to q_5, q_6 or q_7. So, $\{q_1, q_2\} \in \pi_1$.

q_5 is not 1-equivalent to q_6 but to q_7. So, $\{q_5, q_7\} \in \pi_1$

Hence,

$$\pi_1 = \{\{q_3, q_4\}, \{q_0, q_6\}, \{q_1, q_2\}, \{q_5, q_7\}\}$$

now we have

q_3 is 2-equivalent to q_4. So, $\{q_3, q_4\} \in \pi_2$.

q_0 is not 2-equivalent to q_6. So, $\{q_0\}, \{q_6\} \in \pi_2$.

q_1 is 2-equivalent to q_2. So, $\{q_1, q_2\} \in \pi_2$.

q_5 is 2-equivalent to q_7. So, $\{q_5, q_7\} \in \pi_2$.

Hence,

$$\pi_2 = \{\{q_3, q_4\}, \{q_0\}, \{q_6\}, \{q_1, q_2\}, \{q_5, q_7\}\}$$

q_3 is 3-equivalent to q_4; q_1 is 3-equivalent to q_2 and q_5 is 3-equivalent to q_7. Hence,

$$\pi_3 = \{\{q_0\}, \{q_1, q_2\}, \{q_3, q_4\}, \{q_5, q_7\}, \{q_6\}\}$$

As $\pi_3 = \pi_2$, the minimum state automaton is

$$M' = (Q', \{a, b\}, \delta', [q_0], \{[q_3, q_4]\})$$

Where δ' is given as

State	a	b
$[q_0]$	$[q_1, q_2]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_3, q_4]$	$[q_3, q_4]$
$[q_3, q_4]$	$[q_5, q_7]$	$[q_6]$
$[q_5, q_7]$	$[q_3, q_4]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_6]$

Finite Automata with outputs- Mealy and Moore Machines(Transducers):

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

1. Mealy Machine
2. Moore Machine

1. Mealy Machine:

A Mealy Machine is a finite state machine whose output depends on the present state as well as the present input. Mathematically, a mealy machine is a 6-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

– Q is a finite non empty set of states

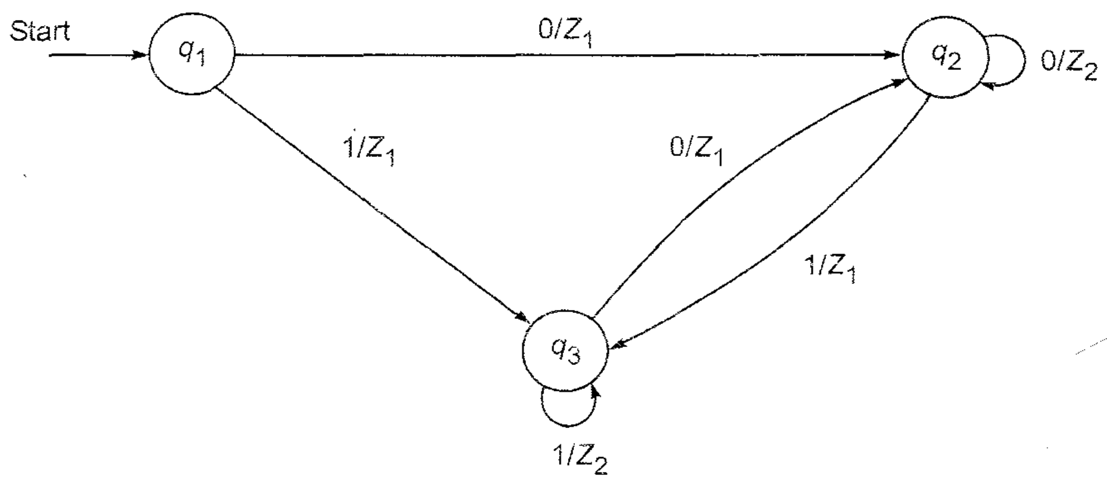
– Σ is an finite non empty set of input symbols called input alphabet

– Δ is a finite non empty set of output symbols called output alphabet.

– $\delta: Q \times \Sigma \rightarrow Q$ is a transition function

– $\lambda: Q \times \Sigma \rightarrow \Delta$ is a output function

– $q_0 \in Q$ is the initial state



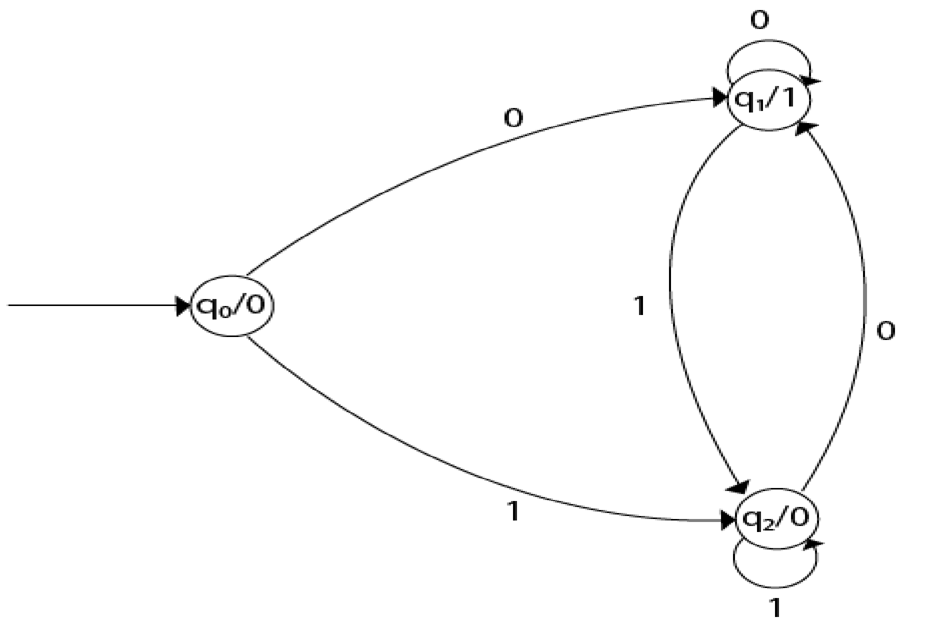
The transition table for the above mealy machine is given as follows.

Present state	Next state			
	$a = 0$		$a = 1$	
	state	output	state	output
$\to q_1$	q_2	Z_1	q_3	Z_1
q_2	q_2	Z_2	q_3	Z_1
q_3	q_2	Z_1	q_3	Z_2

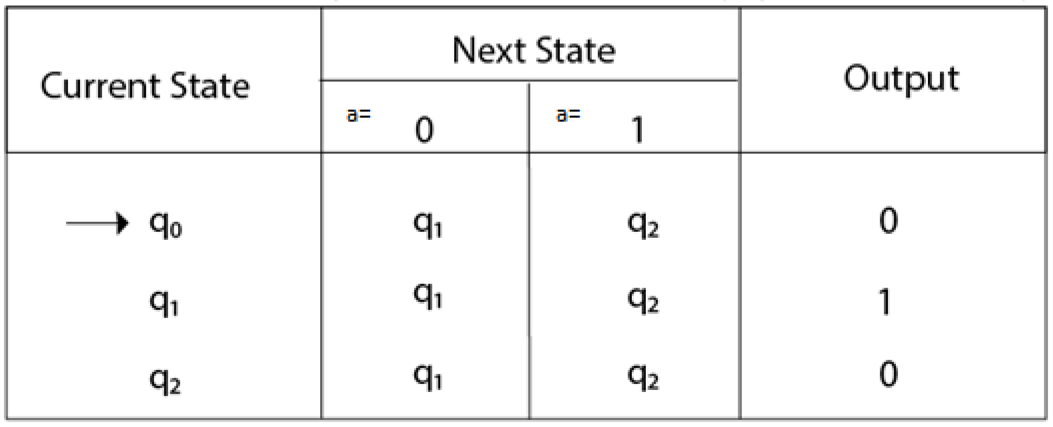
2. Moore Machine:

A Moore Machine is a finite state machine whose output depends only on the present state and is independent of current input symbol. Mathematically, a Moore Machine is a 6- tuple (Q, Σ, Δ, δ, λ, q_0) where

- Q is a finite non empty set of states
- Σ is an finite non empty set of input symbols called input alphabet
- Δ is a finite non empty set of output symbols called output alphabet.
- δ: $Q \times \Sigma \to Q$ is a transition function
- λ: $Q \to \Delta$ is a output function
- $q_0 \in Q$ is the initial state



The transition table for the above moore machine is given as follows.



Current State	Next State		Output
	a= 0	a= 1	
→ q_0	q_1	q_2	0
q_1	q_1	q_2	1
q_2	q_1	q_2	0

<u>Conversion of Mealy machine to Moore Machine:</u>

Step1:

For a state q_i, determine the number of outputs associated with q_i in the next state column

Step 2:

If the outputs corresponds to state q_i in the next state column are same, then retain the state q_i else split q_i into different states with the number of new states being equal to the number of different outputs of q_i.

Step 3:

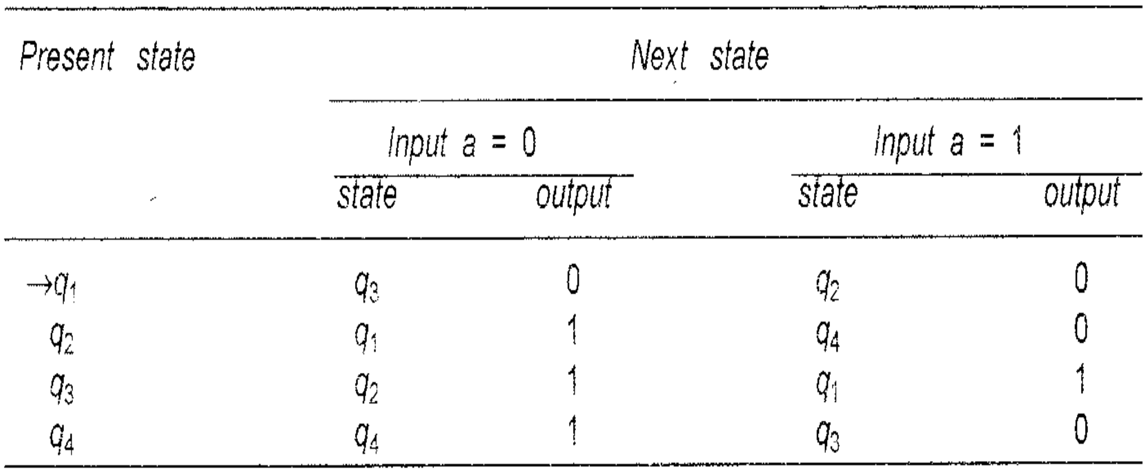
Rearrange the states and outputs in the format of Moore machine.

Step 4:

If the output in the constructed new state table corresponding to the initial state is 1, then this specifies the acceptance of null string ε by mealy machine. Hence to make both the mealy and moore machines equivalent, we either need to ignore the response of moore machine to input ε, or insert a new initial state at beginning where output as 0 while the other new elements would remain the same.

Ex:

Convert the following Mealy Machine to Moore Machine



Present state	Next state			
	Input a = 0		Input a = 1	
	state	output	state	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

q_1 is associated with one output 1 and q_2 is associated with two different outputs 0 and 1. Similarly, q_3 *is* associated with the output 0 and q_4 are associated with the outputs 0 and 1 respectively. So. we split q_2 into q_{20} and q_{21} Similarly, q_4 is split into q_{40} and q_{41}.

Rearranging the transition table for the new states ,we get

Present state	*Next state*			
	Input a = 0		*Input a = 1*	
	state	*output*	*state*	*output*
$\rightarrow q_1$	q_3	0	q_{20}	0
q_{20}	q_1	1	q_{40}	0
q_{21}	q_1	1	q_{40}	0
q_3	q_{21}	1	q_1	1
q_{40}	q_{41}	1	q_3	0
q_{41}	q_{41}	1	q_3	0

Rearranging the states and outputs in the format of moore machine, we get

Present state	*Next state*		*Output*
	a = 0	*a = 1*	
$\rightarrow q_1$	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

To make Moore and Mealy Machines equivalent, for the input null string , insert a new initial state at beginning where output as 0 while the other new elements would remain the same. So the equivalent Moore Machine is

Present state	*Next state*		*Output*
	a = 0	*a = 1*	
$\rightarrow q_0$	q_3	q_{20}	0
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

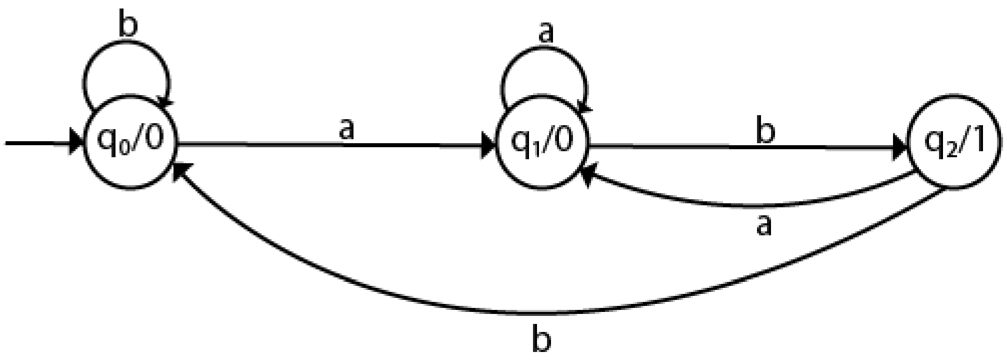
Conversion of Moore Machine to Mealy Machine:

Let the given Moore Machine be M=(Q, Σ, Δ, δ, λ, q_0). Then for the equivalent Mealy Machine

i) define the output function λ' as λ'(q,a) = λ(δ(q,a)) for all states q and input symbols a.

ii) The transition function is the same as that of the given Moore Machine.

Ex: convert the following Moore Machine to Mealy Machine.



The transition table for the given Moore Machine is

Present State	Next State		Output(λ)
	a	b	
q0	q1	q0	0
q1	q1	q2	0
q2	q1	q0	1

Now calculating the output function for all states and all outputs , we get

$$\begin{aligned} \lambda'(q0, a) &= \lambda(\delta(q0, a)) \\ &= \lambda(q1) \\ &= 0 \end{aligned}$$

$$\begin{aligned} \lambda'(q0, b) &= \lambda(\delta(q0, b)) \\ &= \lambda(q0) \\ &= 0 \end{aligned}$$

$$\begin{aligned} \lambda'(q1, a) &= \lambda(\delta(q1, a)) \\ &= \lambda(q1) \\ &= 0 \end{aligned}$$

$$\begin{aligned} \lambda'(q1, b) &= \lambda(\delta(q1, b)) \\ &= \lambda(q2) \\ &= 1 \end{aligned}$$

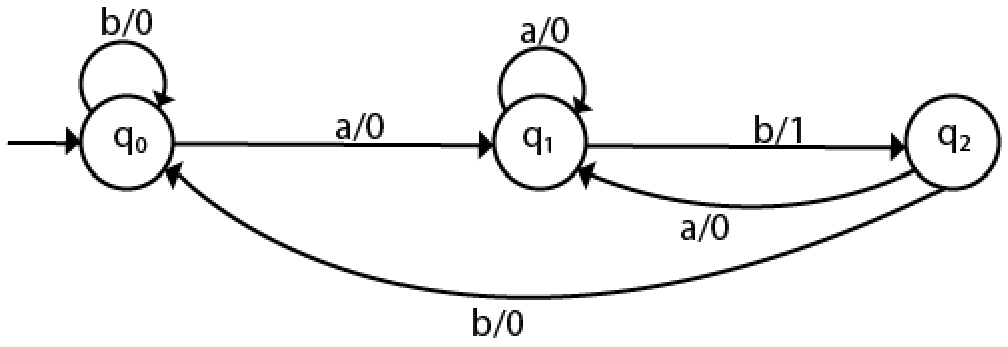
$$\begin{aligned} \lambda'(q2, a) &= \lambda(\delta(q2, a)) \\ &= \lambda(q1) \\ &= 0 \end{aligned}$$

$$\begin{aligned} \lambda'(q2, b) &= \lambda(\delta(q2, b)) \\ &= \lambda(q0) \\ &= 0 \end{aligned}$$

Hence the transition table for the Mealy machine can be drawn as follows:

Σ / Q	Input a		Input b	
	State	Output	State	Output
q_0	q_1	0	q_0	0
q_1	q_1	0	q_2	1
q_2	q_1	0	q_0	0

The equivalent Mealy Machine for the given Moore Machine is given as follows,



Applications of FiniteAutomata:

Some of the applications of finite automata are

1. In compilers we use finite automata for lexical analyzers, and push down automatons for parsers.

2. In search engines, we use finite automata to determine tokens in web pages.

3. Finite automata model protocols, electronic circuits.

4. Context-free grammars are used to describe the syntax of essentially every programming language.

5. Automata theory offers many useful models for natural language processing.

Limitations of Finite Automata:

The limitations of Finite automata are

1. A Finite Automata has finite number of states and so it does not have the capacity to remember arbitrary long amount of information.

2. Finite Automata have trouble recognizing various types of languages involving counting, calculating, storing the string.