UNIT-I

Coverage Topics:

- Central concepts of strings, languages, and automata theory,
- Introduction to Finite Automata
- Deterministic Finite Automata,
- Non-Deterministic Finite Automata,
- Equivalence of NFA with DFA
- Finite Automata with Epsilon Transitions,
- Equivalence of epsilon NFA to NFA / DFA
- Finite Automata with Output.

1. Central Concepts of Automata Theory:

1.1 Symbol:

A symbol is the smallest building block (an abstract entity), which can be any alphabet, letter, or any picture.

Example: Frequently used symbols are: a, b, c, ……, z, A, B,……,Z, 0, 1, …..9

1.2 Alphabet:

An alphabet is a *finite set of symbols*. It is denoted by **Σ**(Sigma).

Example: Frequently used alphabets are:

- Σ = {0,1} is an alphabet of binary digits
- Σ = {0,1,2,….9} is an alphabet of decimal digits
- Σ = {a,b,c} is one alphabet
- Σ = {A,B,C,….,Z} is an alphabet of Capital Letters
- Σ = {0,1,a,b} is one alphabet

1.3 String:

A string is a *finite sequence of symbols* chosen from some alphabet. It is generally denoted by **w** and string is also called a *word.*

Examples:

- a , b , aa , ab , ba , …… are the strings from the alphabet Σ={a,b}
- 01101 is a string from the binary alphabet Σ={0,1}

1.3.1 Length of a string:

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

Example: If w=abaa is a string then length is |w|=|abaa|=4.

- **The number of strings of length 2 that can be generated over an alphabet Σ={a,b} are 4. The strings are {aa,ab,bb,ba}**
  It is denoted as Σ^2 = {aa,ab,bb,ba}. No of symbols in Σ are 2 so, total strings= $\mathbf{2^2}$

- **The number of strings of length 3 that can be generated over an alphabet Σ={a,b} are 8. The strings are {aaa, aab, aba, abb, baa, bab, bba, bbb}. It is denoted as Σ^3 = {aaa, aab, aba, abb, baa, bab, bba, bbb}**
  No of symbols in Σ are 3 so, total strings= 2^3
- So finally, if number of symbols of Σ is represented by **|Σ|**, then number of strings of length **n** that can be generated over the alphabet Σ is $\mathbf{|\Sigma|^n}$

1.3.2 Powers of Σ (Powers of an alphabet):

Let Σ be an alphabet, then Σ^K is nothing but set of all strings of length K, over Σ.

If Σ={a,b}

- **Σ^1** is nothing but set of all strings over Σ of length '1'
  Σ^1 ={a,b} and total strings is: **$|\Sigma^1|$** = 2

- **Σ^2** is nothing but set of all strings over Σ of length '2'
  Σ^2 => **Σ.Σ (concatenation)** => {aa, ab, ba, bb} and total strings is: **$|\Sigma^2|$** = 4

- **Σ^3** is nothing but set of all strings over Σ of length '3'
  Σ^3 => **Σ.Σ.Σ (concatenation)** => {aaa, aab, aba, abb, baa, bab, bba, bbb} and total strings is: **$|\Sigma^3|$** = 8

1.3.3 Empty String (ε epsilon):

It is the empty string consisting of zero symbols, denoted by **ε(epsilon)**.

- **Σ^0** is nothing but set of all strings over Σ of length '0'
  Σ^0 ={**ε**}, **ε** is called epsilon, **|ε|=0**, and total strings is: **$|\Sigma^0|$ = 1**

1.3.4 Kleene Closure (Σ^*):

Set of all possible strings including epsilon(ε) over an alphabet Σ={a,b}.

- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots \Sigma^n$ i.e Union of all lengths of strings.
  Σ^* = {{ε} U {a,b} U {aa,ab,ba,bb} ……………} i.e All Possible Strings over Σ

1.3.5 Positive Kleene Closure (Σ^+):

Set of all possible strings excluding epsilon(ε) over an alphabet Σ={a,b}.

- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots \Sigma^n$
  Σ^+ = {{a,b} U {aa,ab,ba,bb} ....}

1.3.6 Prefix of a string:

A prefix of a string is any number of leading symbols of that string.

Ex: Let a string w=abc , then

Prefixes are: ε , a , ab and abc.

1.3.7 Suffix of a string:

A suffix of a string is any number of trailing symbols of that string.

Ex: Let a string w=abc , then

Suffixes are: ε , c , bc and abc

1.3.6 Substring:

A part of a string is called a substring.

Ex: Let a string w=abcd , then

Substrings are: a , b , c , d , ab , bc , cd , abc , bcd , and abcd.

But acd is not a substring. Similaryly ac , ad , bd , abd are not substrings.

1.4 Language:

A language is a *set of strings*, chosen from some alphabet Σ. It is denoted by L.

-
- If Σ is an alphabet, then Σ* gives the set of all possible strings of any length (including null string) and is an infinite language but a complete language.
- **Σ*** is the super set of all the languages i.e If L is a language, then **L** ⊆ **Σ***

Examples:

1. Set of all strings of length 2 that can be generated over the alphabet Σ ={a,b}
   L_1 = {aa, ab, ba, bb} // This is a “Finite Language”

2. Set of all strings which start with **a** that can be generated over the alphabet Σ ={a,b}
   L_2 = {a, aa, ab, aab, aba, aaa, ………………} //This is an “Infinite Language”

3. Set of all strings which end with 0, that can be generated over the alphabet Σ ={0,1}
   L_3 = {0, 00, 10, 000, 010, 100, 110, ………………} //This is an “Infinite Language”

Note:

1. language formed over Σ can be **Finite** Or **Infinite**.

2. L={} that is an empty set is also a valid language; it is called **empty language.** It is denoted by Φ(phi).

3. L={ε} is the set consisting of an empty string(epsilon) is also a valid language but not an empty language.

1.5 Formal Languages:

For a C’ program, every statement is written with the defined character set and joined in some definite manner to form statements of the C’ language. The compiler is in the position to understand each statement which follows some pre-defined rules as sentences follow grammar in English.

In conclusion, formal language also sets of strings meant for designing a language to write instructions for machines.

Example: Regular languages, Context Free languages are two languages used in compiler construction.

1.6 Finite Automata/Finite State Machine:

Automata:

An automaton (Plural: automata) is an "abstract computing machine/device" (simply called a mathematical model) or a mechanized unit that performs certain specified actions without human intervention. i.e., something that works automatically or a self-operating machine/device.

Ex: ATM, Cola-Machine, etc.

In computer science, an automaton is a machine that can perform the computation in a mechanized manner. As a CSE student, we should know what is computable and what is not, and if it is computable then how it can be implemented as a machine.

The basic aim of automata theory is to draw a boundary between what is computable and what is not computable and encompass the study of abstract machines that can perform computing.

The computing machines are 2 types: **Problem-specific dedicated machine** which takes given input and produces desired output, and the instructions are implicit. Whereas in the **generic machine** that performs a variety of computations on different kinds of input, the instructions to be supplied explicitly.

In summary, Formal Language defines a language to write instructions for a machine and automata theory refers to theoretical or mathematical description of the design of a computing machine. Computation is nothing but knowing what is computable and what is not.

1.7 Finite Automata (FA):

A finite state machine or finite automata is the simplest language recognition machine. It consists of a finite no of states and basic rules for transitions from one state to another.

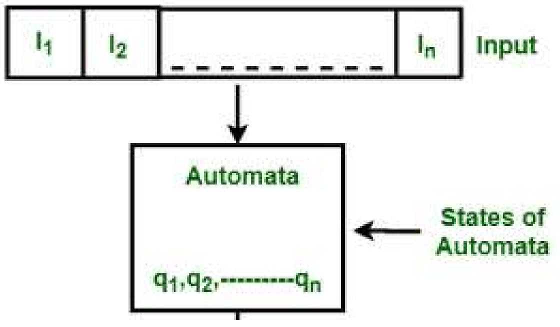


Fig: Architecture of Finite State machine/Finite Automata

A finite automata or finite state machine is an abstract machine (a mathematical model) that contains a finite number of states and transitions. It is the simplest machine of digital computers to recognize patterns(languages). It is a recognizer for regular languages.

Note:

- Abstract means existing in thought or as an idea but not having a physical existence.
- A device need not even be a physical hardware.
- An automaton with finite states is called a Finite Automaton (FA) or Finite State Machine (FSM).

1.7.1 Formal Definition of Finite Automata:

An automaton can be represented by 5-tuples (Q, ∑, δ, q_0, F) and denoted by M.

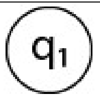
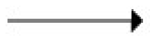
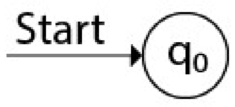
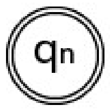
Where:

1. Q is a finite set of states.
2. ∑ is a finite set of input symbols called the alphabet of the automaton.
   ∑ is called Sigma.
3. δ (delta) is the transition function which denotes the transition from one state to another over an input symbol.
   Ex: δ(q,a) = p, where q and p are states, a is an input symbol. The FA is in state q, and upon reading an input symbol a, it goes to state p.
4. q_0 is the initial state/starting state from where any input is processed (q_0 ∈ Q).
5. F is a set of final state/states (F ⊆ Q).

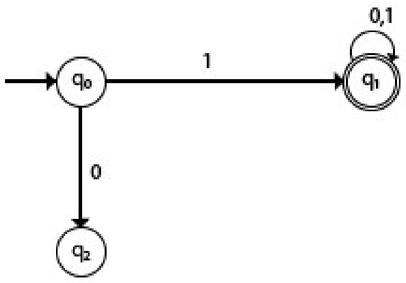
1.7.2 Transition Diagram / State Transition Diagram

A transition diagram or state transition diagram is a special kind of flow chart or a directed graph where the vertices correspond to the states of FA and edges correspond to transitions from one state to another on an input symbol. Among the states, the first stage is denoted as the initial state and is represented by an arrow before the vertex, and the final states are denoted by double circles.

Some Notations that are used in the transition diagram:

 State	 Transition from one state to another
	

Example: A FA that all strings that begins with 1 is as follows:



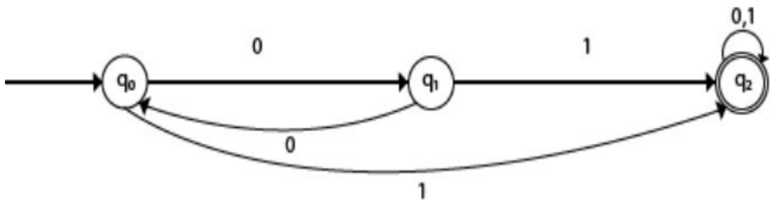
In the above transition diagram, q0 is initial state, q1 is final state and q2 whenever it reaches, the FA can never enter final state. So, this state is called either dead state or trap state. The transitions over q1 are self-loop over 0,1.

1.7.3 State Transition Table or Transition Table: A transition table is a tabular representation of the transition functions. It takes two arguments (a state and a symbol) and returns a state (next state).

A transition table is represented by the following things:

- The rows of the table correspond to the states.
- The columns correspond to the input symbols.
- The next state is the entry for the row corresponding to state q and the column corresponding to input a.
- An arrow denotes the starting state.
- A star or a circle denotes the final statExample:

Transition Diagram:



Transition Table: Transition table for the above transition diagram is as follows:

Present State	Next state for Input 0	Next State of Input 1
→q0	q1	q2
q1	q0	q2
*q2	q2	q2

In the above q0 is initial state, q2 is final state and q1 is intermediate state.

1.7.4 Properties of Finite Automata

1. δ(q_i, ε) = q_i i.e., the state transition happens only on an input symbols

2. For all strings w and input symbol a there exist the transitions:

δ(q_0, aw) = δ(δ(q_0, a), w)

δ(q_0, wa) = δ(δ(q_0, w), a)

1.7.5 Acceptance of String:

Given a string 'w,' we can determine whether the string 'w' is accepted by the Finite Automata (DFA/NFA). A string 'w' is accepted by a finite automata M(Q, ∑, δ, q_0, F) if and only if there is a transition on w from initial state that leads to some final state. Otherwise, the string is not accepted by the machine M.

i.e., δ(q_0, w) = q_f for some final state in F

1.7.6 Language Accepted by a Finite Automata (Set of Strings):

The language accepted by a finite automata M(Q, ∑, δ, q_0, F) denoted by L(M) is the set of all strings accepted by the same finite automata.

L(M) = { w | δ(q_0, w) = q_f for some final state in F}

A language is a regular set /language if it is the set accepted by some finite automata.

1.7.7 Automata can model many things

- They can describe the operation of a small device, like the control component of an alarm clock or a microwave.
- They are also used in lexical analyzers to recognize well-formed expressions in programming languages like XML:
  - ab1 is a legal name of a variable in Java
  - 5u= is not
- Similarly, the Email address is validated using simple Lexical analyzers.

1.8 Deterministic Finite Automata (DFA):

It is a Finite Automata in which exactly one transition should exist from every state on every input symbol.

In DFA, one can determine the state to which the machine will move for each input symbol. Hence, it is called Deterministic Automaton. The machine is called Deterministic Finite Automaton because it has a finite number of states.

1.8.1 Definition of DFA: DFA consists of 5-tuples of

machine M = (Q, ∑, δ, q0, F) where:

1. Q is a finite set of states.
2. ∑ is a finite set of input symbols called the alphabet of the automaton.
3. δ (delta) is the transition function that denotes the transition from one state to another over an input symbol.
   δ maps from (Q x ∑) → Q
   Ex: δ(q,a) = p, where q and p are states, a is an input symbol. The DFA is in state q, and upon reading an input symbol a, it goes to state p.
4. q_0 is the initial state/starting state from where any input is processed (q_0 ∈ Q).
5. F is a set of final state/states (F ⊆ Q).

1.8.2 Representation of a DFA: A DFA is represented by State transition diagram or State transition table as mentioned in 1.7.2 and 1.7.3.

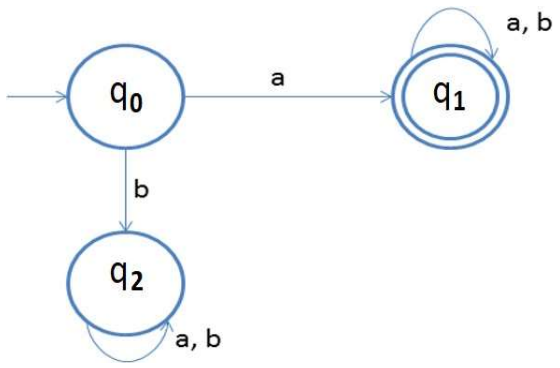
Example 1: Design a DFA which accepts set of all strings over an input alphabet Σ = {a, b}, that starts with 'a'.

Ans: The given language is sets of all strings which start with 'a'.

i.e L = {a, aa, ab, aaa, aab, aba, abb, aabb, aaabbba, …..}

Clearly the language is infinite because there is infinite number of strings.

Transition Diagram:



Where q0 is the initial state, q1 is the final state, and q2 is the dead or trap state.

The idea is very simple, follow the steps below, and you will understand.

1. Start with initial state q0, it is evident from the problem that the first transition must be on ‘a’ as every string begins with ‘a’. Since ‘a’ also accepted string, the transitioned state say q1 becomes the final state. Two states are confirmed: start/initial state and final state i.e q0 and q1.
2. Once we reach the final state, the string accepts everything on ∑ so, we have a loop on ‘a’ and ‘b’ input symbols.
3. And if the first input is something other than ‘a’ then string should not be accepted. So, from q0 there a transition on ‘b’ to state q2 and is considered as dead/trap stae having gone, the FA can never reach the final state.

1.8.3 Acceptance of a string ‘w’ on DFA:

Let’s take an input string w = “abab”, and check whether it is accepted or not by the designed DFA.

$$\begin{aligned}\delta(q0,w) &= \delta(q0,abab) = \delta(\delta(q0,a),bab) \quad \text{(properties of DFA)}\\ &= \delta(q1,bab) = \delta(\delta(q1,b),ab)\\ &= \delta(q1,ab) = \delta(\delta(q1,a),b)\\ &= \delta(q1,b) = \delta(\delta(q1,b),\varepsilon)\\ &= \delta(q1,\varepsilon) = q1 \in F \quad \text{(properties of DFA)}\end{aligned}$$

i.e., $\delta(q0,w) \vdash q1 \in F$, upon the string w = abab, reaches to a final state q1, we can conclude that the string w is accepted by DFA.

The above can also written as if you understand above thoroughly a:

$$\delta(q0,w) = \delta(\mathbf{q0,a}bab) \vdash \delta(\mathbf{q1,b}ab) \vdash \delta(\mathbf{q1,a}b) \vdash \delta(\mathbf{q1,b}) \vdash \delta(q1, \varepsilon) = q1 \in F$$

1.9 Non-Deterministic Finite Automata (NFA or NDFA):

The term non-determinism means from a given state on an input symbol, the transition function δ may lead to a set of states which are subset of Q.

i.e., A state have more than one transition on the same input symbol.

1.9.1 Definition of NFA: Formally NFA can be represented by 5 tuples

M=(Q , ∑ , δ , q0 ,F) Where:

1. Q- non-empty finite set of states.
2. ∑- finite no of input symbols
3. δ- transition function that is mapped as $\delta : Q \times \Sigma \rightarrow 2^{Q}$
4. q0-initial state
5. F- set of final states.

1.9.2 Acceptance of String by NFA:

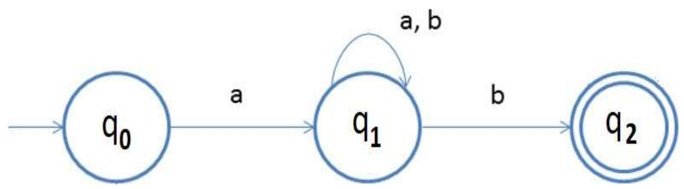
- In DFA, the string is said to be accepted if the finite automata stop at a final state after processing the entire string.
- Since there is only one path, one can confirm acceptance deterministically.
- In the case of NFA, for a string ‘w,’ multiple paths may exist for the given string; some paths may lead to the final state(s), and some may not.
- If at least we find a path that takes the finite automata to a final state after processing the entire string, then we can conclude that NFA accepts the string ‘w’.

Example 1: Design a NFA which accepts set of all strings over an input alphabet Σ={a,b}, that starts with ‘a’ and ends with ‘b’.

Ans: The given language is set of all strings over an alphabet Σ={a,b}, that start with ‘a’ and end with ‘b’.

i.e L = {ab, aab, abb, aaab, aabb, abab, abbb, aaabb, aababbab, ….}, an infinite language as there are infinite strings.

Transition Diagram:


$$\begin{tikzcd} \arrow[r] & q_0 \arrow[r, "a"] & q_1 \arrow[loop above, "a, b"] \arrow[r, "b"] & \boxed{q_2} \end{tikzcd}$$

Where q0 is the initial state, q1 is the intermediate state, and q2 is the final state.

The idea is very simple:

1. From the problem, there must be a transition to another state (q1) on ‘a’ from the initial state q0. So, both q0 and q1 are states.
2. Since the string ends with b, there must be a transition from q1 to another state q2, to accept the string. Make q2 as the final state.
3. We should accept everything in between the first input, ‘a’, and last input, ‘b’; that is why we have a loop on ‘a’ and ‘b’ inputs at q1 state.
4. And if the first input is something other than ‘a’, we don’t care, so there is no transition on input ‘b’ from the q0 state. And we don’t care about transitions on inputs ‘a’ and ‘b’ from the q2 state.

Limitations of NFA:

- It cannot be implemented because of non-deterministic in nature. i.e from the state on an input symbol, there may be multiple transitions.
- To overcome this, it must be converted back to Deterministic FA which is implementable.

Note:

1: Every DFA is NFA but vice-versa is not true.
The above property is true because the δ of DFA is ⊆ δ of NFA
As $\delta : Q \times \Sigma \rightarrow 2^Q \supseteq Q \times \Sigma \rightarrow Q$

2: We do not have dead states in NFA.

3: NFA and DFA are both equivalent in power.
This is because "Every DFA is NFA" and "We can convert NFA to DFA."

1.10 Equivalence of NFA with DFA:

- A language L, described by some NFA, can also be described by some DFA.
- A DFA can have up to 2^n states, while NFA has n states for the same language
- The subset construction method is used to prove the equivalence of DFA and NFA.
- The subset construction starts from an NFA $M_N = (Q_N, \sum, \delta_N, q_0, F_N)$ and constructs its equivalent DFA $M_D = (Q_D, \sum, \delta_D, [q_0], F_D)$.
    - Q_D is a subset of Q_N where Q_D has up to 2^n states in the worst case.
    - $[q_0]$ is the initial state of Q_D where q_0 is the initial state of M_N.
    - F_D is the set of all final states that contain the final states of Q_N.
    - δ_D is denoted as $Q_D \times \sum \rightarrow Q_D$

  where $\delta_D\ ([q_0], x) = [q_1, q_2, q_3, \ldots\ q_j]$ iff

  $\delta_N\ (q_0, x) = \{q_1, q_2, q_3, \ldots\ q_j\}$

  Similarly, $\delta_D\ ([q_0\ q_1], x) = [\delta_N\ (q_0, x)\ \mathrm{U}\ \delta_N\ (q1, x)]$

Steps for converting NFA to DFA:

Step 1: Initialize $Q_D = \{[q_0]\}$ ($[q_0]$ the initial state of M_D where q_0 - the initial state of M_N)

Step 2: Then, find transitions over all input symbols in $\sum$ on the initial state of M_D as:

$\delta D\ ([q0], x) = [q1, q2, q3, \ldots\ qj]$ iff $\delta N\ (q0, x) = \{q1, q2, q3, \ldots\ qj\}$

Then add $[q1, q2, q3, \ldots\ qj]$ as a new state to the MD

i.e., in the M_N, if the transitions take to multiple states, then combine all the states as a single state and add as a new state to Q_D if not exist already.

Step 3: For every new state in Q_D, find the transitions on each input symbol as:

$\delta_D\ ([q_0\ q_1], x) = [\delta_N\ (q_0, x)\ \mathrm{U}\ \delta_N\ (q_0, x)]$

If the obtained state is not in Q_D, then add it to Q_D.

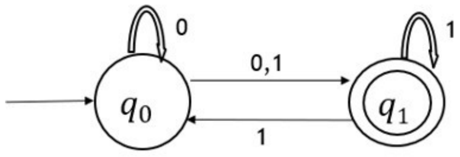
Step 4: Repeat step 3 until no more new states are added to Q_D.

Step 5: In the obtained DFA M_D, make the state as final state when at least one state contains in the states of F_N (final states of NFA)

Example: Conversion of NFA to DFA

- Let M=($\{q_0, q_1\}$,$\{0,1\}$,δ, q_0,$\{q_1\}$) be NFA where $\delta(q_0, 0) = \{q_0, q_1\}$, $\delta(q_0, 1) = \{q_1\}$, $\delta(q_1, 0) = \emptyset$,$\delta(q_1, 1) = \{q_0, q_1\}$. Construct the equivalent DFA.

Solution: Let us draw a transition diagram & table for the given NFA



Transition Diagram

States	Input 0	Input 1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$* q_1$	$\emptyset$	$\{q_0, q_1\}$

Transition Table

Let the DFA be MD(Q_D,∑,δ_D, [q0],F_D)

Step 1:The initial state of NFA q0 becomes the initial state of DFA [q_0]

Step 2: Computing the transition function δ_D for the M_D

δ_D([q0],0) = [q0, q1] as δ(q0, 0) = {q0, q1}

δ_D([q0],1) = [q1]

Step 3: Add the new states to the Q_D of M_D i.e Q_D={[q0],[q1],[q0, q1]}

Step 4: Find the transitions for the new states with all the input symbols in ∑.

δ_D(([q1],0) = $\emptyset$

δ_D(([q1],1) = [q0, q1]

δ_D(([q0, q1], 0) = [q0, q1]

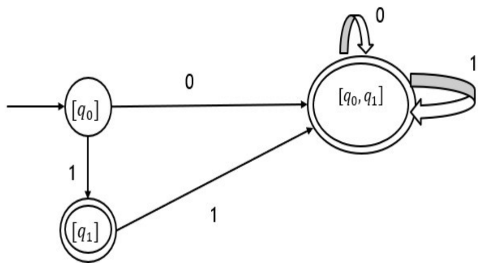
δ_D(([q0, q1], 1) = [q0, q1].

Step 5: No new state is obtained from the above transitions. So, we will stop the process and an equivalent DFA is obtained.

Transition Table for the DFA

states	Input 0	Input 1
$\rightarrow q_0$	$[q_0, q_1]$	$[q_1]$
$[q_1]$	$\emptyset$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

Transition Diagram of Equivalent DFA



1.11 ε-NFA (NFA with ε-moves):

- In general, for finite automata, there is no transition over ε i.e., δ (q, ε) = q and We can extend the class of NFA's by allowing instantaneous ε transitions. The automaton may change its state without reading any input symbol.
    - ε -NFAs add a convenient feature, but (in a sense) they bring us nothing new. They do not extend the class of languages that can be represented. Both NFA & ε -NFAs recognize the same languages.
    - However, to distribute the design's complexity or enhance the reader's understanding, the NFA can be extended by including transitions on 'ε'. The final automaton doesn't include 'ε' transitions.
    - In obtaining finite automata from regular expressions, generally, the ε-NFA is only produced.
    - Formally NFA with ε-moves defined as a 5-tuple machine M=(Q,∑, δ**,** q0, F) where
        1. Q - set of finite no of states
        2. ∑- set of finite no of input symbols
        3. δ - transition function Q × ∑ U { ε } → 2^Q

           i.e., δ(q , a) consisting of a state 'p' such that there is a transition labeled 'a' from q to p, where a is either ε or input symbol.
        4. q0-initial state
        5. F-Set of final states.

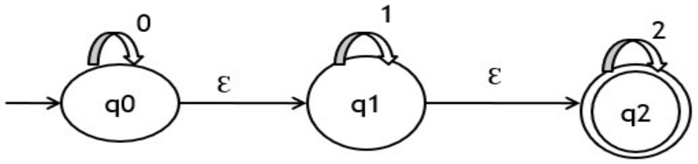
Example1: Design a ε-NFA (NFA with ε-moves) which accept set of all strings over an input alphabet Σ={0,1, 2}, that any number of 0's followed by any number of 1's followed by any number of 2's.

Ans: The given language is set of all strings over an alphabet ={0,1, 2}, that any number of 0's followed by any number of 1's followed by any number of 2's.

i.e L = { ε , 0, 1, 2, 00, 01, 02, 11, 12, 22, 011, 012, 0012, ……..000111122,....... }

Clearly the language is infinite because there is infinite number of strings.

Transition Diagram:


$$\begin{tikzcd} \arrow[r] & q0 \arrow[loop above, "0"] \arrow[r, "\varepsilon"] & q1 \arrow[loop above, "1"] \arrow[r, "\varepsilon"] & \boxed{q2} \arrow[loop above, "2"] \end{tikzcd}$$

Transition Table:

	0	1	2	ε
->q0	q0	Φ	Φ	q1
q1	Φ	q1	Φ	q2
*q2	Φ	Φ	q2	Φ

1.11.1 Acceptance of a string w:

Define $\hat{\delta}$(q, w) as consisting of all states 'p' such that one can go from q to p along a path labelled 'w', which includes edges labeled with ε – transitions.

Ex. For the string W = 012, the path sequence as follows: q0 (0) q0 (ε) q1 (1) q1 (ε) q2 (2) q2.

Any Finite automata with ε-moves is always a NFA with ε-transitions/moves as DFA can't have ε – transitions.

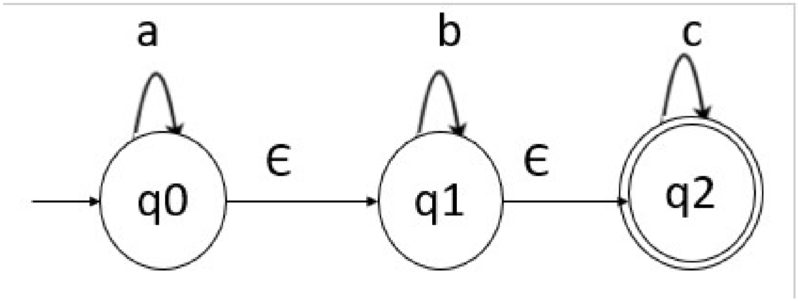
1.11.2 ε-closure of a state:

For constructing $\hat{\delta}$, it is important to compute the set of states reachable from a given state q with ε – transitions only. To obtain those state we need to compute ε-closure from the state 'q'. **ε-closure of a state or ε-closure(q)** is defined as the set of all states reachable from a given state q using ε-transitions only.

Steps to find ε-closure(q):

1. Add q to ε-closure(q).
2. For any new state 'p' in ε-closure(q), define δ (p, ε) and add it in ε-closure(q).
3. There can't be any other element in ε-closure(q).

Example:



ε-closure (q0) = {q0, q1, q2} - self-state+ ε-reachable states.

ε-closure (q1) = {q1, q2} -q1 is self-state and q2 is a state obtained from q1 with ε .

ε-closure (q2) = {q2}

Redefining $\hat{\delta}$ as follows:

1. $\hat{\delta}$(q, ε) = ε-closure (q)
2. $\hat{\delta}$(q, wa) = ε-closure ($\delta(\hat{\delta}(q, w)$, a))

1.11.3 Language accepted by ε-NFA

denoted by L(M) as { w | $\hat{\delta}$(q, w) contains a state in F}

1.11.4 Elimination of ε-Transitions or Equivalence of ε- NFA with NFA

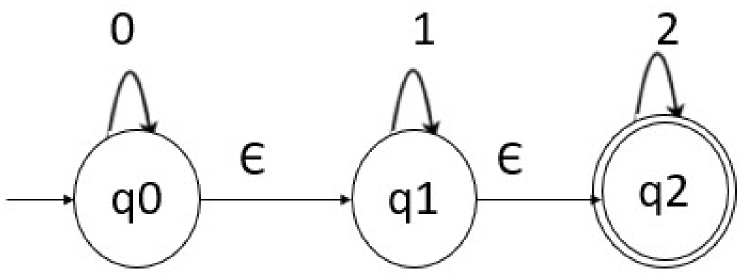
Let DFA Mε = (Q_ε, ∑, δ_ε, q_0, F_ε) be the ε-NFA and we need to construct its equivalent NFA as M$_N$ (Q_N, ∑, δ_N, q_0, F_N).

In NFA is constructed the same way the moves of ε-NFA. So, all Q_N, δ_N, q_0, F_N are same as ε-NFA and the transitions are obtained by

$$\delta_N(q, a) = \varepsilon - \text{closure } (\delta_\varepsilon(\hat{\delta}(\text{q}, \varepsilon), a))$$

Example

Convert the given NFA with epsilon to NFA without epsilon.



Solution

First find ε-closure of each state i.e., find ε-reachable states from the current state.

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

The transition of NFA can be obtained as follows:

δ_N (q0, 0) = ε-closure (δ (δ^(q0, ε),0))
= ε-closure(δ(ε-closure(q0),0))
= ε-closure (δ(q0, q1,q2), 0))
= ε-closure (δ(q0, 0) ∪ δ(q1, 0) U δ(q2, 0))
= ε-closure (q0 U Φ ∪ Φ)
= ε-closure(q0)
= {q0, q1, q2}

δ_N (q0, 1) = ε-closure (δ (δ^(q0, ε),1))
= ε-closure (δ (q0, q1, q2), 1))
= ε-closure (δ (q0, 1) ∪ δ (q1, 1) U δ(q2, 1))
= ε-closure (Φ ∪q1 U Φ)
= ε-closure (q1)
= {q1, q2}

δ_N (q0, 2) = ε-closure (δ (δ^ (q0, ε),2))
= ε-closure (δ (q0, q1, q2), 2))
= ε-closure (δ(q0, 2) ∪ δ(q1, 2) U δ(q2, 2))
= ε-closure (Φ U ΦU q2)
= ε-closure(q2)
= {q2}

δ_N (q1, 0) = ε-closure (δ (δ^ (q1, ε),0))
= ε-closure (δ (q1, q2), 0))
= ε-closure (δ (q1, 0) U δ (q2, 0))
= ε-closure (Φ ∪ Φ)
= ε-closure(Φ)
= Φ

δ_N (q1,1) = ε-closure(δ(δ^(q1, ε),1))
= ε-closure(δ(q1,q2), 1))
= ε-closure(δ(q1, 1) U δ(q2, 1))
= ε-closure(q1 ∪ Φ)
= ε-closure(q1)
= {q1,q2}
δ_N (q1, 2) = ε-closure(δ(δ^(q1, ε),2))
= ε-closure(δ(q1,q2), 2))
= ε-closure(δ(q1, 2) U δ(q2, 2))
= ε-closure(Φ ∪ q2)
= ε-closure(q2)
= {q2}
δ_N (q2, 0) = ε-closure(δ(δ^(q2, ε),0))
= ε-closure(δ(q2), 0))
= ε-closure(δ(q2, 0))
= ε-closure(Φ)
= Φ
δ_N (q2, 1) = ε-closure(δ(δ^(q2, ε),1))
= ε-closure(δ(q2), 1)
= ε-closure(δ(q2, 1))
= ε-closure(Φ)
= Φ
δ_N (q2, 2) = ε-closure(δ(δ^(q2, ε),))
= ε-closure(δ(q2), 2))
= ε-closure(δ(q2, 2))
= ε-closure(q2)
= {q2}

Now, we will summarize all the computed δ_N transitions as given state transition table as

States\inputs	0	1	2
q0	{q0,q1,q2}	{q1,q2}	{q2}
q1	Φ	{q1,q2}	{q2}
q2	Φ	Φ	{q2}

1.11.5 Equivalence of ε - NFA with DFA:

- A language L, described by some ε - NFA, can also be described by some DFA.
- A DFA can have up to 2^n states, while NFA has n states for the same language
- The subset construction method is used to prove the equivalence of DFA and NFA.
- The subset construction starts from an NFA Mε = $(Q\varepsilon, \sum, \delta\varepsilon, q_0, F\varepsilon)$ and constructs its equivalent DFA M_D = $(Q_D, \sum, \delta_D, [q]\ , F_D)$.
  - Q_D is a subset of $Q\varepsilon$ where Q_D has up to 2^n states in the worst case.
  - $[q]$ is the initial state of Q_D where q_0 is the initial state of Mε.
  - F_D is the set of all final states that contain the final states of $Q\varepsilon$.
  - δ_D is denoted as $Q_D \times \sum \rightarrow Q_D$

where $\delta_D\ ([q_0], x) = \varepsilon\text{-closure}(q_1, q_2, q_3, \ldots.. q_j)$ iff

$\delta\varepsilon\ (q_0, x) = \{q_1, q_2, q_3, \ldots.. q_j\}$

Similarly, $\delta_D\ ([q_0\ q_1], x) = [\delta\varepsilon\ (q_0, x)\ \text{U}\ \delta\varepsilon\ (q1, x)]$

Steps for converting NFA to DFA:

Step 1: Initialize Q_D = { ε-closure(q_0)} (make ε-closure(q_0) as the initial state of M_D where q_0 - the initial state of Mε)

Step 2: Then, find transitions over all input symbols in ∑ on all the new states in Q_D of M_D as:

$\delta_D\ ([q], x) = \varepsilon\text{-closure}(q1, q2, q3, \ldots.. qj)$ iff $\delta\varepsilon(q, x) = \{q1, q2, q3, \ldots.. qj\}$

Then add $[q1, q2, q3, \ldots.. qj]$ as a new state in Q_D of M_D

Or

$\delta_D\ ([q_0\, q_1], x) = \varepsilon\text{-closure}(\ \delta\varepsilon(q_0, x)\ \text{U}\ \delta\varepsilon(q_0, x))$

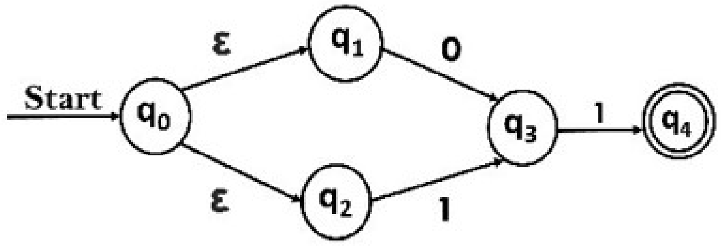
i.e., in the Mε, if the transitions take to multiple states, then find the ε-closure() on all states and combine them as a single state and add as a new state to Q_D if not exist already.

Step 3: Repeat step 2 until no more new states are added to Q_D.

Step 4: In the obtained DFA M_D, make the state as final state where at least one state contains in the states of Fε (final states of NFA)

Example 1:

Convert the NFA with ε into its equivalent DFA.



Solution:

Let us obtain ε-closure of each state.

1. ε-closure {q0} = {q0, q1, q2}
2. ε-closure {q1} = {q1}
3. ε-closure {q2} = {q2}
4. ε-closure {q3} = {q3}
5. ε-closure {q4} = {q4}

Now, let ε-closure {q0} = {q0, q1, q2} so, make [q0 q1 q2] be the initial state and call it as A.

Define transition over the initial state as:

$\delta_D([q0\ q1\ q2], 0) = \varepsilon\text{-closure}\ \{\delta((q0, q1, q2), 0)\ \}$

$= \varepsilon\text{-closure}\ \{\delta(q0, 0) \cup \delta(q1, 0) \cup \delta(q2, 0)\ \}$

$= \varepsilon\text{-closure}\ \{q3\}$

= [q3] **call it as state B**.

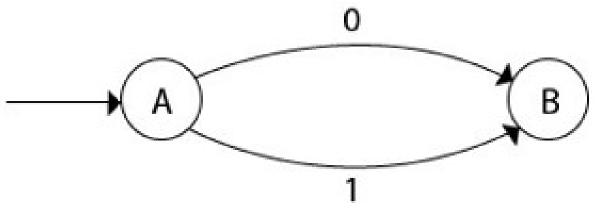
δ_D ([q0 q1 q2], 1) = ε-closure {δ((q0, q1, q2), 1) }

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

= ε-closure {q3}

= [q3] = B.

The partial DFA will be



Now define transitions over B,

δ_D([q3], 0) = ε-closure {δ(q3, 0) } = ϕ

δ_D ([q3], 1) = ε-closure {δ(q3, 1) }

= ε-closure {q4}

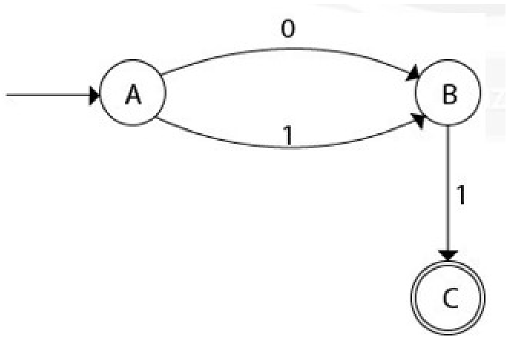
= [q4] **i.e. state C**

For state C:

δ_D([q4], 0) = ε-closure {δ(q4, 0) } = ϕ

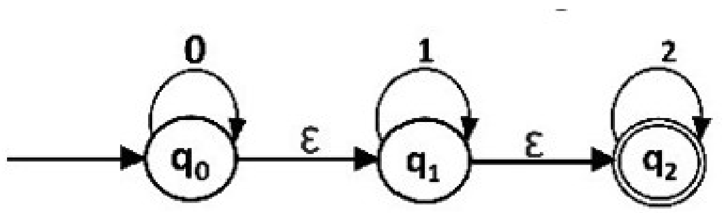
δ_D (C, 1) = ε-closure {δ(q4, 1) } = ϕ

The DFA will be,



Example 2:

Convert the given NFA into its equivalent DFA.



Solution: Let us obtain the ε-closure of each state.

1. ε-closure(q0) = {q0, q1, q2}
2. ε-closure(q1) = {q1, q2}
3. ε-closure(q2) = {q2}

Now we will obtain δ' transition. Let ε-closure(q0) = {q0, q1, q2} and make [q0 q1 q2] as initiatial state and it as **state A**.

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

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

= ε-closure{q1}

= [q1, q2]   **call it as state B**

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

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

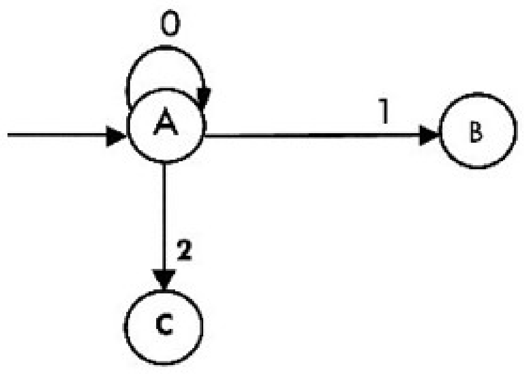
= ε-closure{q2}

= [q2]   **call it state C**

Thus we have obtained

1. δ'(A, 0) = A
2. δ'(A, 1) = B
3. δ'(A, 2) = C

The partial DFA will be:



Now we will find the transitions on states B and C for each input.

Hence

$\delta'(B, 0) = \varepsilon\text{-closure}\{\delta((q1, q2), 0)\}$

$= \varepsilon\text{-closure}\{\delta(q1, 0) \cup \delta(q2, 0)\}$

$= \varepsilon\text{-closure}\{\phi\} = \phi$

$\delta'(B, 1) = \varepsilon\text{-closure}\{\delta((q1, q2), 1)\}$

$= \varepsilon\text{-closure}\{\delta(q1, 1) \cup \delta(q2, 1)\}$

$= \varepsilon\text{-closure}\{q1\}$

$= [q1, q2]$ **i.e. state B itself**

$\delta'(B, 2) = \varepsilon\text{-closure}\{\delta((q1, q2), 2)\}$

$= \varepsilon\text{-closure}\{\delta(q1, 2) \cup \delta(q2, 2)\}$

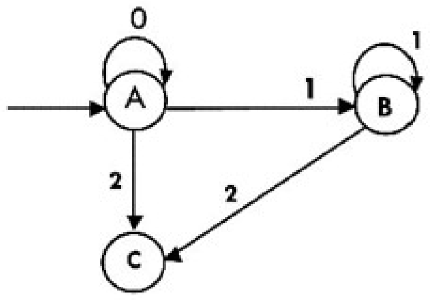
$= \varepsilon\text{-closure}\{q2\}$

$= [q2]$ **i.e. state C itself**

Thus we have obtained

1. $\delta'(B, 0) = \phi$
2. $\delta'(B, 1) = B$
3. $\delta'(B, 2) = C$

The partial transition diagram will be



Now we will obtain transitions for C:

δ'(C, 0) = ε-closure{δ(q2, 0)}

= ε-closure{ϕ} = ϕ

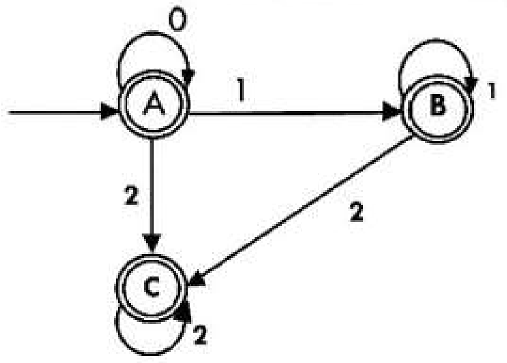
δ'(C, 1) = ε-closure{δ(q2, 1)}

= ε-closure{ϕ} = ϕ

δ'(C, 2) = ε-closure{δ(q2, 2)}

= [q2]

Hence the DFA is



As A = [q0, q1, q2] in which final state q2 lies hence A is final state. B = [q1, q2] in which the state q2 lies hence B is also final state. C = [q2], the state q2 lies hence C is also a final state.

1.12 Finite Automata with Output.

Finite automata may have outputs corresponding to each transition. Two types of finite state machines that generate output are:

- Mealy Machine
- Moore machine

1.12.1 Mealy Machine

A Mealy Machine is an FSM whose output depends on the present state as well as the present input.

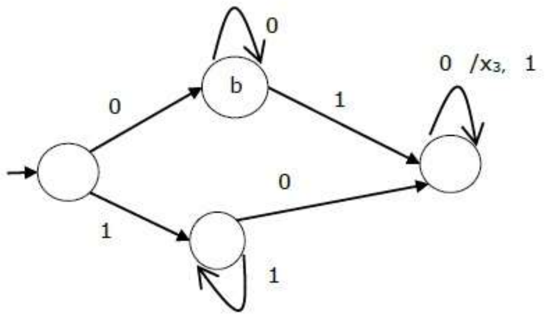
It can be described by a 6 tuple $(Q, \sum, \Delta, \delta, \lambda, q_0)$ where –

- **Q** is a finite set of states.
- **∑** is a finite set of symbols called the input alphabet.
- Δ is a finite set of symbols called the output alphabet.
- **δ** is the input transition function where $\delta: Q \times \sum \rightarrow Q$
- λ is the output transition function where $\lambda: Q \times \sum \rightarrow \Delta$
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).

The state table of a Mealy Machine is shown below –

Present state	Next state			
	input = 0		input = 1	
	State	Output	State	Output
→ a	b	x_1	c	x_1
b	b	x_2	d	x_3
c	d	x_3	c	x_1
d	d	x_3	d	x_2

The state transition diagram of the above Mealy Machine is –



1.12.2 Moore Machine

Moore machine is an FSM whose outputs depend on only the present state.

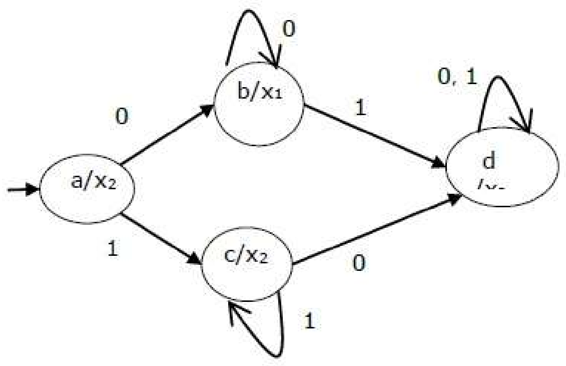
A Moore machine can be described by a 6 tuple (Q, ∑, Δ , δ, λ, q_0) where –

- **Q** is a finite set of states.
- **∑** is a finite set of symbols called the input alphabet.
- Δ is a finite set of symbols called the output alphabet.
- **δ** is the input transition function where δ: Q × ∑ → Q
- λ is the output transition function where λ: Q → Δ
- **q_0** is the initial state from where any input is processed (q_0 ∈ Q).

The state table of a Moore Machine is shown below –

Present state	Next State		Output
	Input = 0	Input = 1	
→ a	b	c	x_2
b	b	d	x_1
c	c	d	x_2
d	d	d	x_3

The state transition diagram of the above Moore Machine is –



1.12.3 Mealy Machine vs. Moore Machine

The following are the points that differentiate a Mealy Machine from a Moore Machine.

Mealy Machine	Moore Machine
Output depends both upon the present state and the present input	Output depends only upon the present state.
Generally, it has fewer states than Moore Machine.	Generally, it has more states than Mealy Machine.
The value of the output function is a function of the transitions and the changes, when the input logic on the present state is done.	The value of the output function is a function of the current state and the changes at the clock edges, whenever state changes occur.
Mealy machines react faster to inputs. They generally react in the same clock cycle.	In Moore machines, more logic is required to decode the outputs resulting in more circuit delays. They generally react one clock cycle later.

1.12.4 Converting Moore Machine to Mealy Machine

Input – Moore Machine

Output – Mealy Machine

Step 1 – Take a blank Mealy Machine transition table format.

Step 2 – Copy all the Moore Machine transition states into this table format.

Step 3 – Check the present states and their corresponding outputs in the Moore Machine state table; if for a state Q_i output is m, copy it into the output columns of the Mealy Machine state table wherever Q_i appears in the next state.

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

Now we apply Algorithm 4 to convert it to Mealy Machine.

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

1.12.5 Converting a Mealy Machine to Moore Machine

Input – Mealy Machine

Output – Moore Machine

Step 1 – Calculate the number of different outputs for each state (Q_i) that are available in the state table of the Mealy machine.

Step 2 – If all the outputs of Qi are same, copy state Q_i. If it has n distinct outputs, break Q_i into n states as Q_{in} where **n** = 0, 1, 2.......

Step 3 – If the output of the initial state is 1, insert a new initial state at the beginning which gives 0 output.

Example

Let us consider the following Mealy Machine –

Present State	Next State			
	a = 0		a = 1	
	Next State	Output	Next State	Output
→ A	d	0	b	1
B	a	1	d	0
C	c	1	c	0
D	b	0	a	1

Here, states ‘a’ and ‘d’ give only 1 and 0 outputs respectively, so we retain states ‘a’ and ‘d’. But states ‘b’ and ‘c’ produce different outputs (1 and 0). So, we divide **b** into **b_0, b_1** and **c** into **c_0, c_1**.

Present State	**Next State**		**Output**
	a = 0	**a = 1**	
→ a	d	b_1	1
b_0	a	d	0
b_1	a	d	1
c_0	c_1	C_0	0
c_1	c_1	C_0	1
D	b_0	a	0