

Theory of Automata

Theory of automata is a theoretical branch of computer science and mathematical. It is the study of abstract machines and the computation problems that can be solved using these machines. The abstract machine is called the automata. The main motivation behind developing the automata theory was to develop methods to describe and analyse the dynamic behaviour of discrete systems.

This automaton consists of states and transitions. The **State** is represented by **circles**, and the **Transitions** is represented by **arrows**.

Automata is the kind of machine which takes some string as input and this input goes through a finite number of states and may enter in the final state.

There are the basic terminologies that are important and frequently used in automata:

Symbols:

Symbols are an entity or individual objects, which can be any letter, alphabet or any picture.

Example:

1, a, b, #

Alphabets:

Alphabets are a finite set of symbols. It is denoted by Σ .

Examples:

1. $\Sigma = \{a, b\}$
- 2.
3. $\Sigma = \{A, B, C, D\}$
- 4.
5. $\Sigma = \{0, 1, 2\}$
- 6.
7. $\Sigma = \{0, 1, \dots, 5\}$
- 8.

$$9. \Sigma = \{\#, \beta, \Delta\}$$

String:

It is a finite collection of symbols from the alphabet. The string is denoted by w .

Example 1:

If $\Sigma = \{a, b\}$, various string that can be generated from Σ are $\{ab, aa, aaa, bb, bbb, ba, aba, \dots\}$.

- A string with zero occurrences of symbols is known as an empty string. It is represented by ϵ .
- The number of symbols in a string w is called the length of a string. It is denoted by $|w|$.

Example 2:

1. $w = 010$
- 2.
3. Number of Sting $|w| = 3$

Language:

A language is a collection of appropriate string. A language which is formed over Σ can be **Finite** or **Infinite**.

Example: 1

$L1 = \{\text{Set of string of length } 2\}$

$= \{aa, bb, ba, ab\}$

Finite Language

Example: 2

$L2 = \{\text{Set of all strings starts with 'a'}\}$

$= \{a, aa, aaa, abb, abbb, ababb\}$

Infinite Language

Finite Automata

- Finite automata are used to recognize patterns.
- It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, then the transition occurs.
- At the time of transition, the automata can either move to the next state or stay in the same state.
- Finite automata have two states, **Accept state** or **Reject state**. When the input string is processed successfully, and the automata reached its final state, then it will accept.

Formal Definition of FA

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

1. Q : finite set of states
2. Σ : finite set of the input symbol
3. q_0 : initial state
4. F : **final** state
5. δ : Transition function

Finite Automata Model:

Finite automata can be represented by input tape and finite control.

Input tape: It is a linear tape having some number of cells. Each input symbol is placed in each cell. **Finite control:** The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one by one from left to right, and at a time only one input symbol is read.

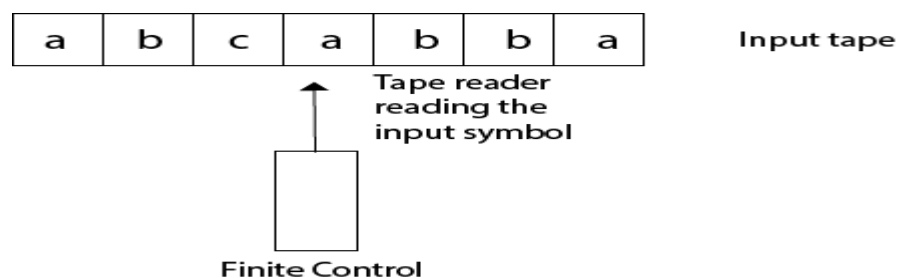
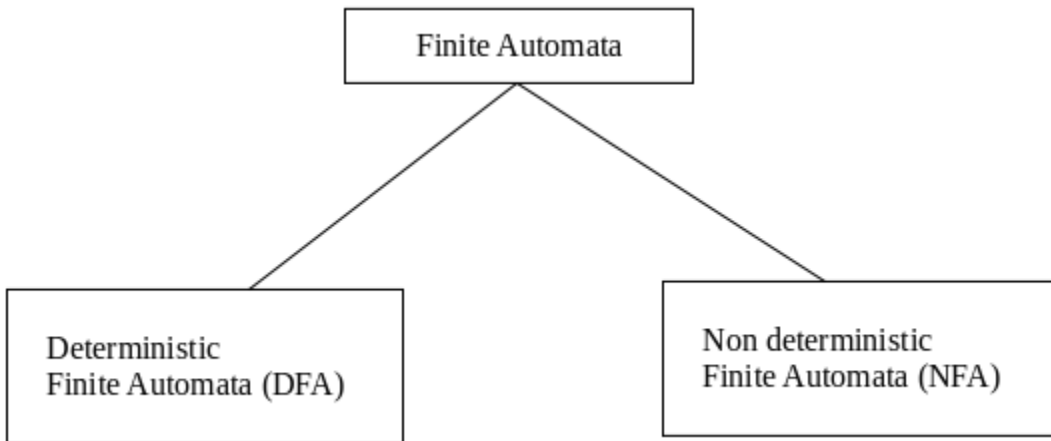


Fig :- Finite automata model

Types of Automata:

There are two types of finite automata:

1. DFA(deterministic finite automata)
2. NFA(non-deterministic finite automata)



1. DFA

DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. In the DFA, the machine goes to one state only for a particular input character. DFA does not accept the null move.

2. NFA

NFA stands for non-deterministic finite automata. It is used to transmit any number of states for a particular input. It can accept the null move.

Some important points about DFA and NFA:

1. Every DFA is NFA, but NFA is not DFA.
2. There can be multiple final states in both NFA and DFA.
3. DFA is used in Lexical Analysis in Compiler.
4. NFA is more of a theoretical concept.

Transition Diagram

A transition diagram or state transition diagram is a directed graph which can be constructed as follows:

- There is a node for each state in Q , which is represented by the circle.
- There is a directed edge from node q to node p labeled a if $\delta(q, a) = p$.
- In the start state, there is an arrow with no source.
- Accepting states or final states are indicating by a double circle.

Some Notations that are used in the transition diagram:

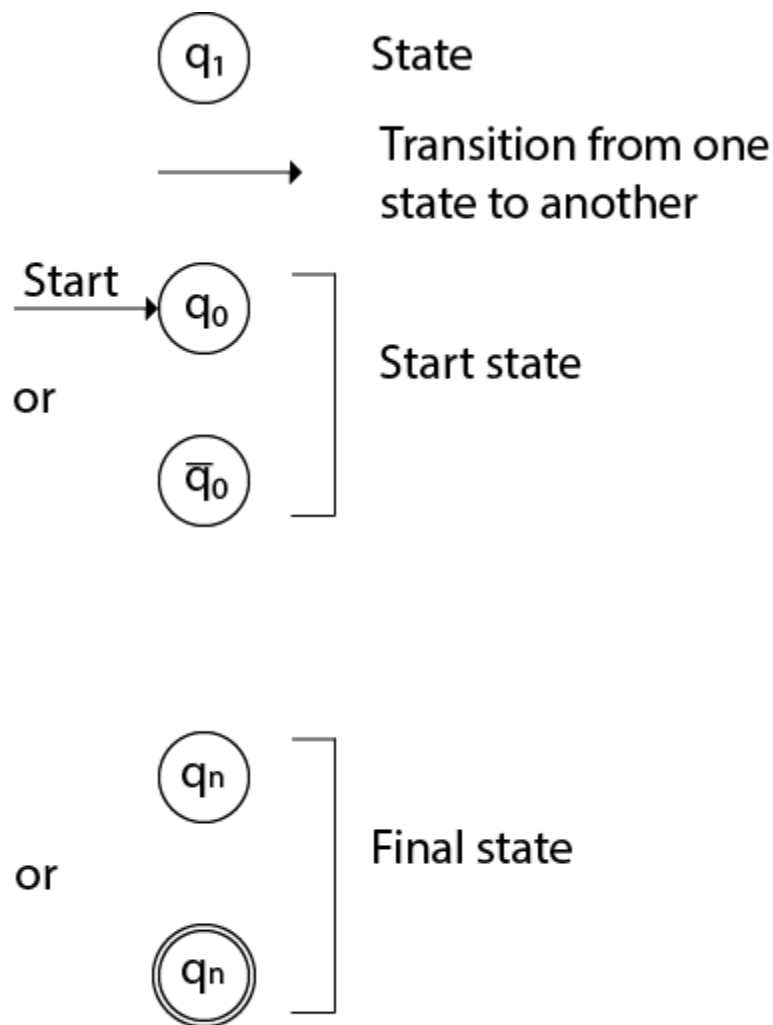


Fig:- Notations

There is a description of how a DFA operates:

1. In DFA, the input to the automata can be any string. Now, put a pointer to the start state q and read the input string w from left to right and move the pointer according to the transition function, δ . We can read one symbol at a time. If the next symbol of string w is a and the pointer is on state p , move the pointer to $\delta(p, a)$. When the end of the input string w is encountered, then the pointer is on some state F .
2. The string w is said to be accepted by the DFA if $r \in F$ that means the input string w is processed successfully and the automata reached its final state. The string is said to be rejected by DFA if $r \notin F$.

Example 1:

DFA with $\Sigma = \{0, 1\}$ accepts all strings starting with 1.

Solution:

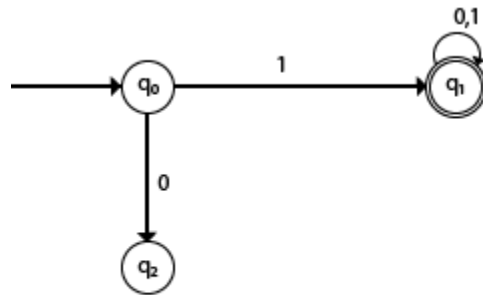


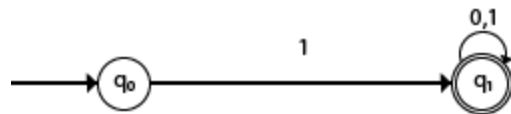
Fig: Transition diagram

The finite automata can be represented using a transition graph. In the above diagram, the machine initially is in start state q_0 then on receiving input 1 the machine changes its state to q_1 . From q_0 on receiving 0, the machine changes its state to q_2 , which is the dead state. From q_1 on receiving input 0, 1 the machine changes its state to q_1 , which is the final state. The possible input strings that can be generated are 10, 11, 110, 101, 111....., that means all string starts with 1.

Example 2:

NFA with $\Sigma = \{0, 1\}$ accepts all strings starting with 1.

Solution:



The NFA can be represented using a transition graph. In the above diagram, the machine initially is in start state q_0 then on receiving input 1 the machine changes its state to q_1 . From q_1 on receiving input 0, 1 the machine changes its state to q_1 . The possible input string that can be generated is 10, 11, 110, 101, 111....., that means all string starts with 1.

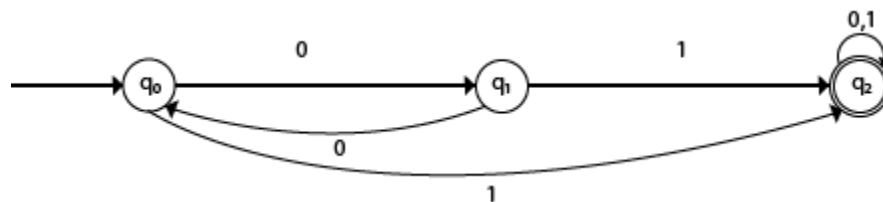
Transition Table

The transition table is basically a tabular representation of the transition function. It takes two arguments (a state and a symbol) and returns a state (the "next state").

A transition table is represented by the following things:

- Columns correspond to input symbols.
- Rows correspond to states.
- Entries correspond to the next state.
- The start state is denoted by an arrow with no source.
- The accept state is denoted by a star.

Example 1:



Solution:

Transition table of given DFA is as follows:

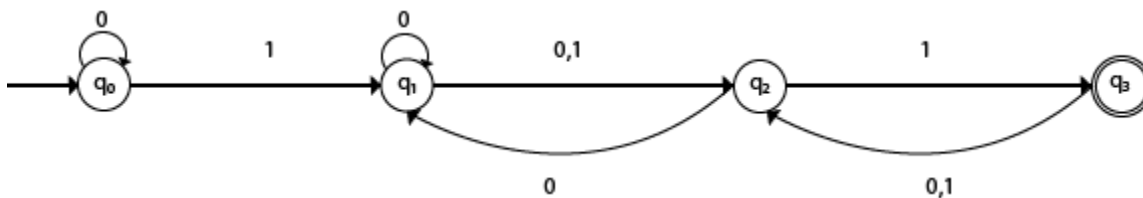
Present State	Next state for Input 0	Next State of Input 1
→q0	q1	q2
q1	q0	q2
*q2	q2	q2

Explanation:

- In the above table, the first column indicates all the current states. Under column 0 and 1, the next states are shown.

- The first row of the transition table can be read as, when the current state is q_0 , on input 0 the next state will be q_1 and on input 1 the next state will be q_2 .
- In the second row, when the current state is q_1 , on input 0, the next state will be q_0 , and on 1 input the next state will be q_2 .
- In the third row, when the current state is q_2 on input 0, the next state will be q_2 , and on 1 input the next state will be q_2 .
- The arrow marked to q_0 indicates that it is a start state and circle marked to q_2 indicates that it is a final state.

Example 2:



Solution:

Transition table of given NFA is as follows:

Present State	Next state for Input 0	Next State of Input 1
→ q_0	q_0	q_1
q_1	q_1, q_2	q_2
q_2	q_1	q_3
* q_3	q_2	q_2

Explanation:

- The first row of the transition table can be read as, when the current state is q_0 , on input 0 the next state will be q_0 and on input 1 the next state will be q_1 .
- In the second row, when the current state is q_1 , on input 0 the next state will be either q_1 or q_2 , and on 1 input the next state will be q_2 .

- In the third row, when the current state is q_2 on input 0, the next state will be q_1 , and on 1 input the next state will be q_3 .
- In the fourth row, when the current state is q_3 on input 0, the next state will be q_2 , and on 1 input the next state will be q_2 .

DFA (Deterministic finite automata)

- DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. The finite automata are called deterministic finite automata if the machine is read an input string one symbol at a time.
- 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., the DFA cannot change state without any input character.
- DFA can contain multiple final states. It is used in Lexical Analysis in Compiler.

In the following diagram, we can see that from state q_0 for input a , there is only one path which is going to q_1 . Similarly, from q_0 , there is only one path for input b going to q_2 .

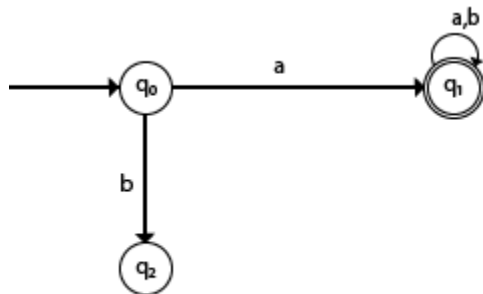


Fig:- DFA

Formal Definition of DFA

A DFA is a collection of 5-tuples same as we described in the definition of FA.

1. Q : finite set of states
2. Σ : finite set of the input symbol
3. q_0 : initial state
4. F : **final** state
5. δ : Transition function

Transition function can be defined as:

1. $\delta: Q \times \Sigma \rightarrow Q$

Graphical Representation of DFA

A DFA can be represented by digraphs called state diagram. In which:

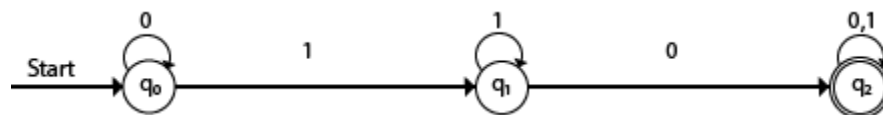
1. The state is represented by vertices.
2. The arc labeled with an input character show the transitions.
3. The initial state is marked with an arrow.
4. The final state is denoted by a double circle.

Example 1:

1. $Q = \{q_0, q_1, q_2\}$
2. $\Sigma = \{0, 1\}$
3. $q_0 = \{q_0\}$
4. $F = \{q_2\}$

Solution:

Transition Diagram:



Transition Table:

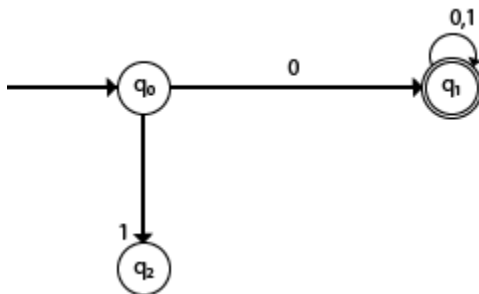
Present State	Next state for Input 0	Next State of Input 1
→q0	q0	q1
q1	q2	q1

*q2	q2	q2
-----	----	----

Example 2:

DFA with $\Sigma = \{0, 1\}$ accepts all starting with 0.

Solution:



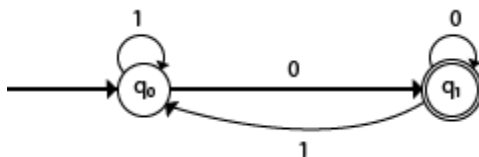
Explanation:

- In the above diagram, we can see that on given 0 as input to DFA in state q_0 the DFA changes state to q_1 and always go to final state q_1 on starting input 0. It can accept 00, 01, 000, 001....etc. It can't accept any string which starts with 1, because it will never go to final state on a string starting with 1.

Example 3:

DFA with $\Sigma = \{0, 1\}$ accepts all ending with 0.

Solution:



Explanation:

In the above diagram, we can see that on given 0 as input to DFA in state q_0 , the DFA changes state to q_1 . It can accept any string which ends with 0 like 00, 10, 110, 100....etc.

It can't accept any string which ends with 1, because it will never go to the final state q_1 on 1 input, so the string ending with 1, will not be accepted or will be rejected.

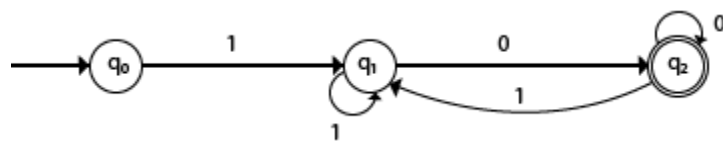
Examples of DFA

Example 1:

Design a DFA with $\Sigma = \{0, 1\}$ accepts those string which starts with 1 and ends with 0.

Solution:

The DFA will have a start state q_0 from which only the edge with input 1 will go to the next state.



In state q_1 , if we read 1, we will be in state q_1 , but if we read 0 at state q_1 , we will reach to state q_2 which is the final state. In state q_2 , if we read either 0 or 1, we will go to q_2 state or q_1 state respectively. Note that if the input ends with 0, it will be in the final state.

Example 2:

Design a DFA with $\Sigma = \{0, 1\}$ accepts the only input 101.

Solution:

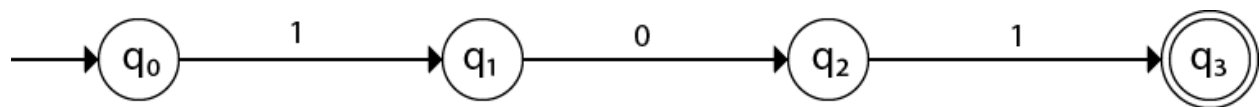


Fig: FA

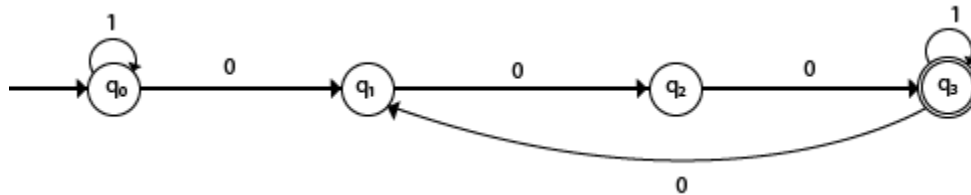
In the given solution, we can see that only input 101 will be accepted. Hence, for input 101, there is no other path shown for other input.

Example 4:

Design FA with $\Sigma = \{0, 1\}$ accepts the set of all strings with three consecutive 0's.

Solution:

The strings that will be generated for this particular languages are 000, 0001, 1000, 10001, in which 0 always appears in a clump of 3. The transition graph is as follows:



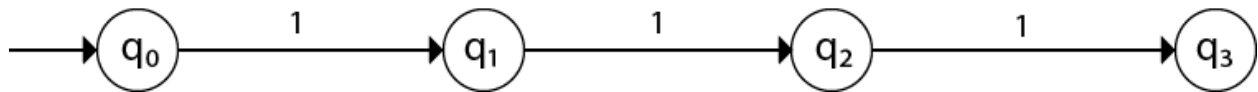
Note that the sequence of triple zeros is maintained to reach the final state.

Example 5:

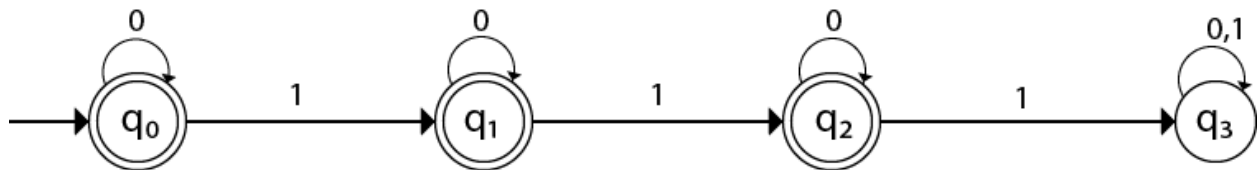
Design a DFA $L(M) = \{w \mid w \in \{0, 1\}^*\}$ and W is a string that does not contain consecutive 1's.

Solution:

When three consecutive 1's occur the DFA will be:



Here two consecutive 1's or single 1 is acceptable, hence



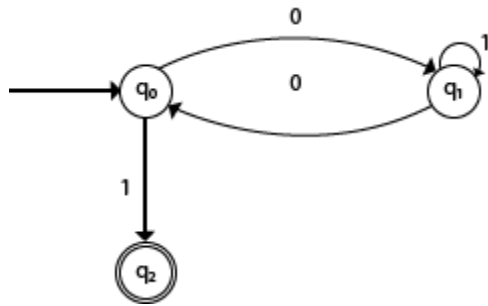
The states q_0 , q_1 , q_2 are the final states. The DFA will generate the strings that do not contain consecutive 1's like 10, 110, 101,..... etc.

Example 6:

Design a FA with $\Sigma = \{0, 1\}$ accepts the strings with an even number of 0's followed by single 1.

Solution:

The DFA can be shown by a transition diagram as:



NFA (Non-Deterministic finite automata)

- NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ϵ transition.

In the following image, we can see that from state q_0 for input a , there are two next states q_1 and q_2 , similarly, from q_0 for input b , the next states are q_0 and q_1 . Thus it is not fixed or determined that with a particular input where to go next. Hence this FA is called non-deterministic finite automata.

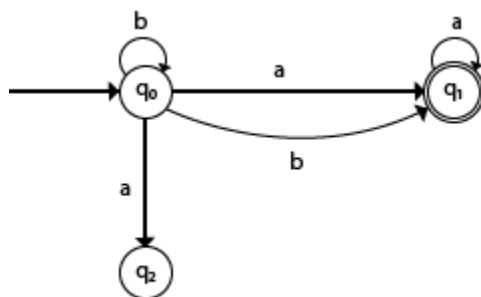


Fig:- NDFA

Formal definition of NFA:

NFA also has five states same as DFA, but with different transition function, as shown follows:

$$\delta: Q \times \Sigma \rightarrow 2^Q$$

where,

1. Q : finite set of states
2. Σ : finite set of the input symbol
3. q_0 : initial state
4. F : **final** state
5. δ : Transition function

Graphical Representation of an NFA

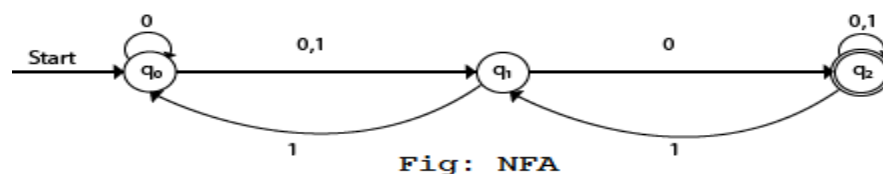
An NFA can be represented by digraphs called state diagram. In which:

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

Example 1:

1. $Q = \{q_0, q_1, q_2\}$
2. $\Sigma = \{0, 1\}$
3. $q_0 = \{q_0\}$
4. $F = \{q_2\}$

Solution: Transition diagram:



Transition Table:

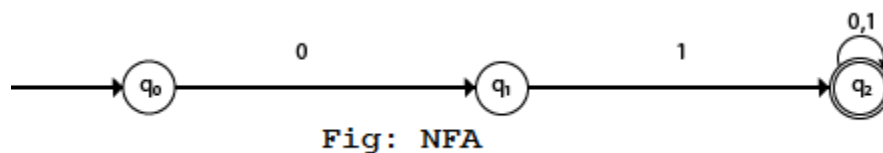
Present State	Next state for Input 0	Next State of Input 1
→q0	q0, q1	q1
q1	q2	q0
*q2	q2	q1, q2

In the above diagram, we can see that when the current state is q0, on input 0, the next state will be q0 or q1, and on 1 input the next state will be q1. When the current state is q1, on input 0 the next state will be q2 and on 1 input, the next state will be q0. When the current state is q2, on 0 input the next state is q2, and on 1 input the next state will be q1 or q2.

Example 2:

NFA with $\Sigma = \{0, 1\}$ accepts all strings with 01.

Solution:



Transition Table:

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

*q2	q2	q2
-----	----	----

Example 3:

NFA with $\Sigma = \{0, 1\}$ and accept all string of length atleast 2.

Solution:

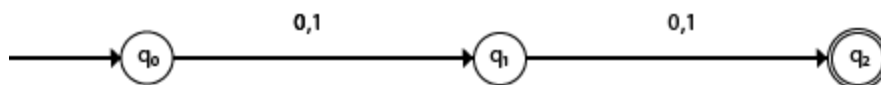


Fig: NFA

Transition Table:

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

Examples of NFA

Example 1:

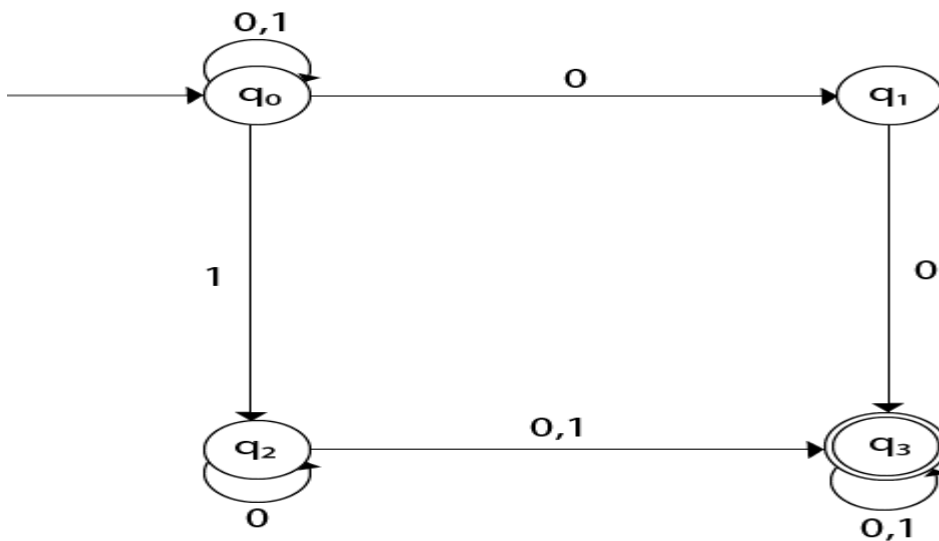
Design a NFA for the transition table as given below:

Present State	0	1
→q0	q0, q1	q0, q2

q1	q3	ϵ
q2	q2, q3	q3
$\rightarrow q3$	q3	q3

Solution:

The transition diagram can be drawn by using the mapping function as given in the table.



Here,

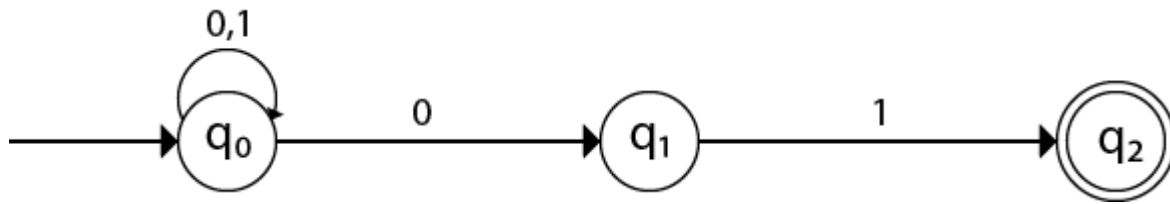
1. $\delta(q_0, 0) = \{q_0, q_1\}$
2. $\delta(q_0, 1) = \{q_0, q_2\}$
3. Then, $\delta(q_1, 0) = \{q_3\}$
4. Then, $\delta(q_2, 0) = \{q_2, q_3\}$
5. $\delta(q_2, 1) = \{q_3\}$
6. Then, $\delta(q_3, 0) = \{q_3\}$
7. $\delta(q_3, 1) = \{q_3\}$

Example 2:

Design an NFA with $\Sigma = \{0, 1\}$ accepts all string ending with 01.

Solution:

Hence, NFA would be:

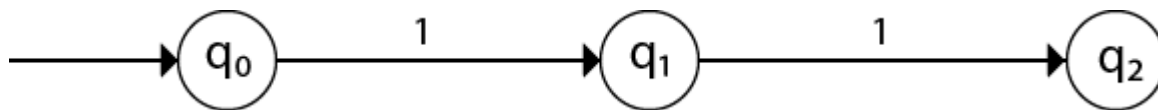


Example 3:

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

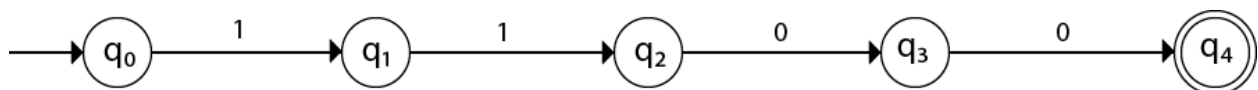
Solution:

The FA with double 1 is as follows:



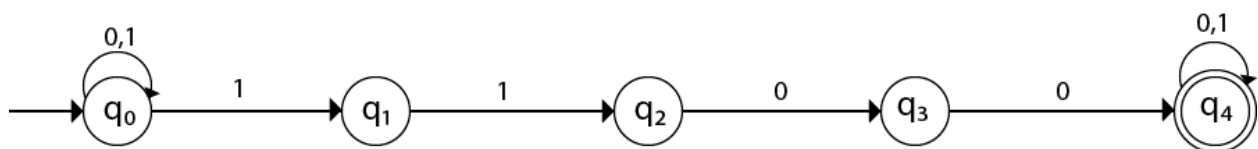
It should be immediately followed by double 0.

Then,



Now before double 1, there can be any string of 0 and 1. Similarly, after double 0, there can be any string of 0 and 1.

Hence the NFA becomes:



Now considering the string 01100011

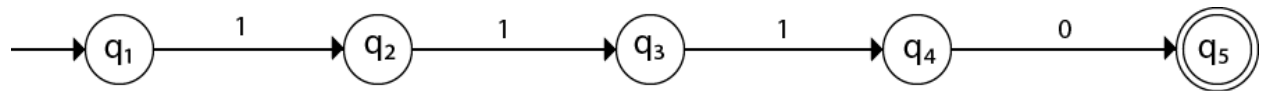
1. $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_4 \rightarrow q_4 \rightarrow q_4$

Example 4:

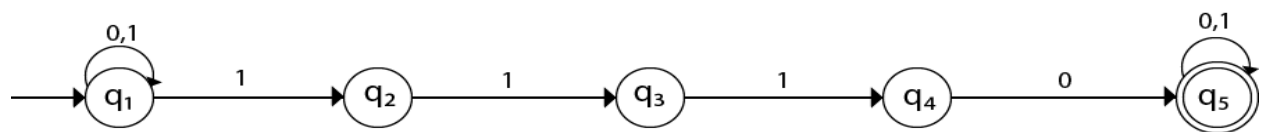
Design an NFA in which all the string contain a substring 1110.

Solution:

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



Now as 1010 could be the substring. Hence we will add the inputs 0's and 1's so that the substring 1010 of the language can be maintained. Hence the NFA becomes:



Transition table for the above transition diagram can be given below:

Present State	0	1
→q1	q1	q1, q2
q2		q3
q3		q4
q4	q5	
*q5	q5	q5

Consider a string 111010,

1. $\delta(q1, 111010) = \delta(q1, 1100)$
2. $\quad \quad \quad = \delta(q1, 100)$
3. $\quad \quad \quad = \delta(q2, 00)$

Got stuck! As there is no path from q_2 for input symbol 0. We can process string 111010 in another way.

1. $\delta(q_1, 111010) = \delta(q_2, 1100)$
2. $\quad \quad \quad = \delta(q_3, 100)$
3. $\quad \quad \quad = \delta(q_4, 00)$
4. $\quad \quad \quad = \delta(q_5, 0)$
5. $\quad \quad \quad = \delta(q_5, \epsilon)$

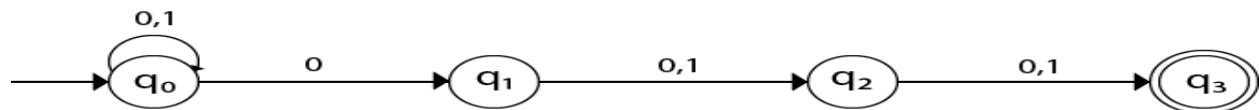
As state q_5 is the accept state. We get the complete scanned, and we reached to the final state.

Example 5:

Design an NFA with $\Sigma = \{0, 1\}$ accepts all string in which the third symbol from the right end is always 0.

Solution:

Thus we get the third symbol from the right end as '0' always. The NFA can be:



The above image is an NFA because in state q_0 with input 0, we can either go to state q_0 or q_1 .

Eliminating ϵ Transitions

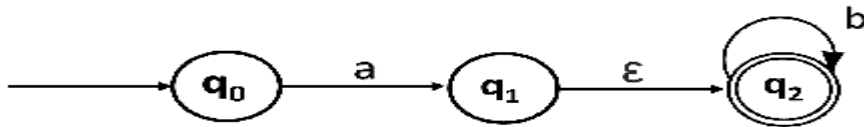
NFA with ϵ can be converted to NFA without ϵ , and this NFA without ϵ can be converted to DFA. To do this, we will use a method, which can remove all the ϵ transition from given NFA. The method will be:

1. Find out all the ϵ transitions from each state from Q . That will be called as ϵ -closure $\{q_i\}$ where $q_i \in Q$.
2. Then δ' transitions can be obtained. The δ' transitions mean a ϵ -closure on δ moves.

3. Repeat Step-2 for each input symbol and each state of given NFA.
4. Using the resultant states, the transition table for equivalent NFA without ϵ can be built.

Example:

Convert the following NFA with ϵ to NFA without ϵ .



Solutions: We will first obtain ϵ -closures of q_0 , q_1 and q_2 as follows:

1. ϵ -closure(q_0) = $\{q_0\}$
2. ϵ -closure(q_1) = $\{q_1, q_2\}$
3. ϵ -closure(q_2) = $\{q_2\}$

Now the δ' transition on each input symbol is obtained as:

1. $\delta'(q_0, a) = \epsilon$ -closure($\delta(\delta^*(q_0, \epsilon), a)$)
2. $= \epsilon$ -closure($\delta(\epsilon$ -closure(q_0), a))
3. $= \epsilon$ -closure($\delta(q_0, a)$)
4. $= \epsilon$ -closure(q_1)
5. $= \{q_1, q_2\}$
- 6.
7. $\delta'(q_0, b) = \epsilon$ -closure($\delta(\delta^*(q_0, \epsilon), b)$)
8. $= \epsilon$ -closure($\delta(\epsilon$ -closure(q_0), b))
9. $= \epsilon$ -closure($\delta(q_0, b)$)
10. $= \Phi$

Now the δ' transition on q_1 is obtained as:

1. $\delta'(q_1, a) = \epsilon$ -closure($\delta(\delta^*(q_1, \epsilon), a)$)
2. $= \epsilon$ -closure($\delta(\epsilon$ -closure(q_1), a))
3. $= \epsilon$ -closure($\delta(q_1, q_2), a$)

4. $= \epsilon\text{-closure}(\delta(q1, a) \cup \delta(q2, a))$
5. $= \epsilon\text{-closure}(\Phi \cup \Phi)$
6. $= \Phi$
- 7.
8. $\delta'(q1, b) = \epsilon\text{-closure}(\delta(\delta^*(q1, \epsilon), b))$
9. $= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q1), b))$
10. $= \epsilon\text{-closure}(\delta(q1, q2), b)$
11. $= \epsilon\text{-closure}(\delta(q1, b) \cup \delta(q2, b))$
12. $= \epsilon\text{-closure}(\Phi \cup q2)$
13. $= \{q2\}$

The δ' transition on $q2$ is obtained as:

1. $\delta'(q2, a) = \epsilon\text{-closure}(\delta(\delta^*(q2, \epsilon), a))$
2. $= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q2), a))$
3. $= \epsilon\text{-closure}(\delta(q2, a))$
4. $= \epsilon\text{-closure}(\Phi)$
5. $= \Phi$
- 6.
7. $\delta'(q2, b) = \epsilon\text{-closure}(\delta(\delta^*(q2, \epsilon), b))$
8. $= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q2), b))$
9. $= \epsilon\text{-closure}(\delta(q2, b))$
10. $= \epsilon\text{-closure}(q2)$
11. $= \{q2\}$

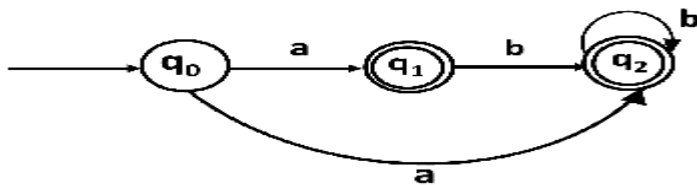
Now we will summarize all the computed δ' transitions:

1. $\delta'(q0, a) = \{q0, q1\}$
2. $\delta'(q0, b) = \Phi$
3. $\delta'(q1, a) = \Phi$
4. $\delta'(q1, b) = \{q2\}$
5. $\delta'(q2, a) = \Phi$
6. $\delta'(q2, b) = \{q2\}$

The transition table can be:

States	A	b
$\rightarrow q_0$	$\{q_1, q_2\}$	ϕ
$*q_1$	ϕ	$\{q_2\}$
$*q_2$	ϕ	$\{q_2\}$

State q_1 and q_2 become the final state as ϵ -closure of q_1 and q_2 contain the final state q_2 . The NFA can be shown by the following transition diagram:



Conversion from NFA to DFA

In this section, we will discuss the method of converting NFA to its equivalent DFA. In NFA, when a specific input is given to the current state, the machine goes to multiple states. It can have zero, one or more than one move on a given input symbol. On the other hand, in DFA, when a specific input is given to the current state, the machine goes to only one state. DFA has only one move on a given input symbol.

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

Steps for converting NFA to DFA:

Step 1: Initially $Q' = \phi$

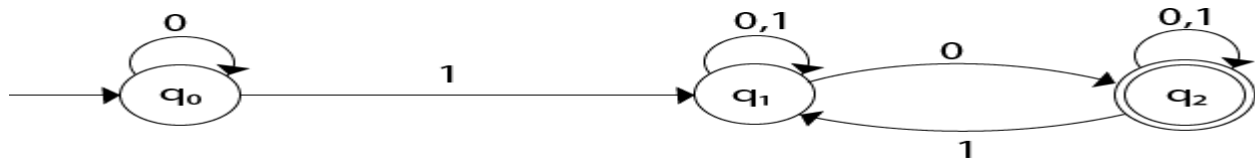
Step 2: Add q_0 of NFA to Q' . Then find the transitions from this start state.

Step 3: In Q' , find the possible set of states for each input symbol. If this set of states is not in Q' , then add it to Q' .

Step 4: In DFA, the final state will be all the states which contain F (final states of NFA)

Example 1:

Convert the given NFA to DFA.



Solution: For the given transition diagram we will first construct the transition table.

State	0	1
→q0	q0	q1
q1	{q1, q2}	q1
*q2	q2	{q1, q2}

Now we will obtain δ' transition for state q0.

1. $\delta'([q0], 0) = [q0]$
2. $\delta'([q0], 1) = [q1]$

The δ' transition for state q1 is obtained as:

1. $\delta'([q1], 0) = [q1, q2]$ (**new** state generated)
2. $\delta'([q1], 1) = [q1]$

The δ' transition for state q2 is obtained as:

1. $\delta'([q2], 0) = [q2]$
2. $\delta'([q2], 1) = [q1, q2]$

Now we will obtain δ' transition on [q1, q2].

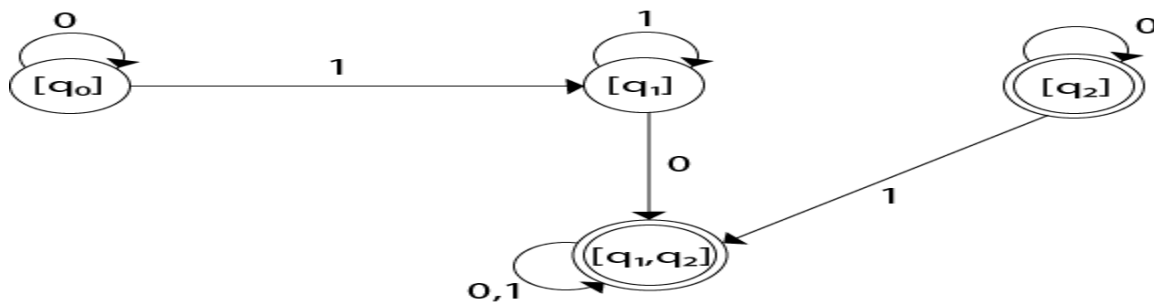
1. $\delta'([q1, q2], 0) = \delta(q1, 0) \cup \delta(q2, 0)$
2. $\quad \quad \quad = \{q1, q2\} \cup \{q2\}$

3. $\delta'([q_1, q_2], 1) = \delta(q_1, 1) \cup \delta(q_2, 1)$
4. $\delta'([q_1, q_2], 1) = \delta(q_1, 1) \cup \delta(q_2, 1)$
5. $\delta'([q_1, q_2], 1) = \{q_1\} \cup \{q_1, q_2\}$
6. $\delta'([q_1, q_2], 1) = \{q_1, q_2\}$
7. $\delta'([q_1, q_2], 1) = [q_1, q_2]$

The state $[q_1, q_2]$ is the final state as well because it contains a final state q_2 . The transition table for the constructed DFA will be:

State	0	1
$\rightarrow [q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1, q_2]$	$[q_1]$
$*[q_2]$	$[q_2]$	$[q_1, q_2]$
$*[q_1, q_2]$	$[q_1, q_2]$	$[q_1, q_2]$

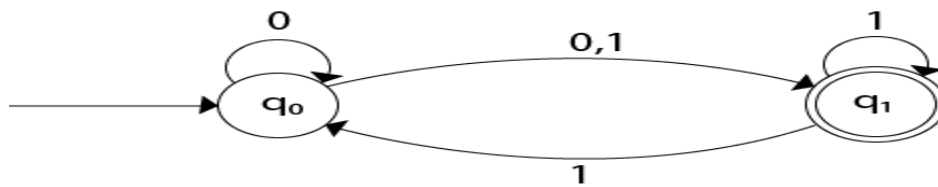
The Transition diagram will be:



The state q_2 can be eliminated because q_2 is an unreachable state.

Example 2:

Convert the given NFA to DFA.



Solution: For the given transition diagram we will first construct the transition table.

State	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$*q_1$	ϕ	$\{q_0, q_1\}$

Now we will obtain δ' transition for state q_0 .

1. $\delta'([q_0], 0) = \{q_0, q_1\}$
2. $\quad = [q_0, q_1]$ (new state generated)
3. $\delta'([q_0], 1) = \{q_1\} = [q_1]$

The δ' transition for state q_1 is obtained as:

1. $\delta'([q_1], 0) = \phi$
2. $\delta'([q_1], 1) = [q_0, q_1]$

Now we will obtain δ' transition on $[q_0, q_1]$.

1. $\delta'([q_0, q_1], 0) = \delta(q_0, 0) \cup \delta(q_1, 0)$
2. $\quad = \{q_0, q_1\} \cup \phi$
3. $\quad = \{q_0, q_1\}$

$$4. \quad = [q_0, q_1]$$

Similarly,

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

$$2. \quad = \{q_1\} \cup \{q_0, q_1\}$$

$$3. \quad = \{q_0, q_1\}$$

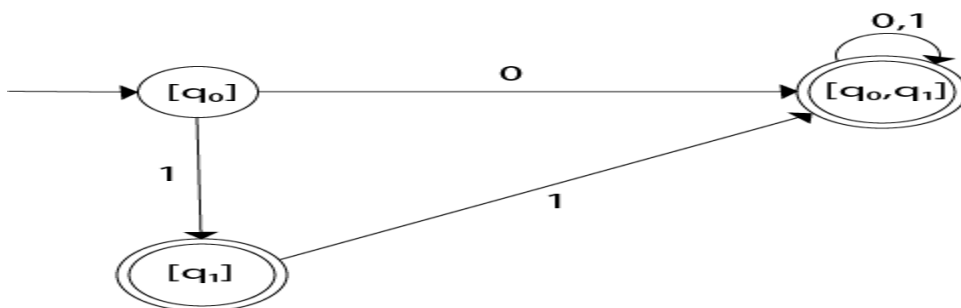
$$4. \quad = [q_0, q_1]$$

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

The transition table for the constructed DFA will be:

State	0	1
$\rightarrow [q_0]$	$[q_0, q_1]$	$[q_1]$
$*[q_1]$	Φ	$[q_0, q_1]$
$*[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

The Transition diagram will be:

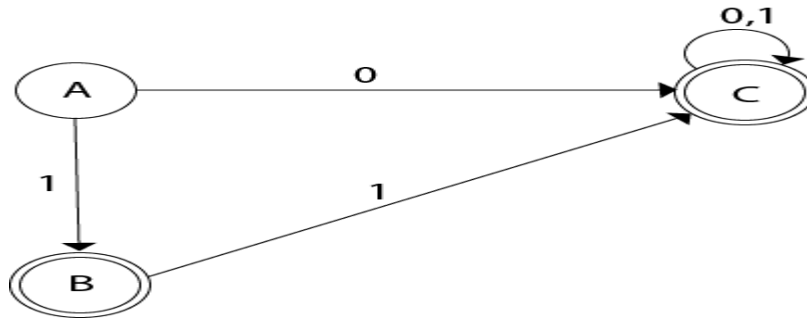


Even we can change the name of the states of DFA.

Suppose

1. $A = [q_0]$
2. $B = [q_1]$
3. $C = [q_0, q_1]$

With these new names the DFA will be as follows:



Conversion from NFA with ϵ to DFA

Non-deterministic finite automata (NFA) is a finite automata where for some cases when a specific input is given to the current state, the machine goes to multiple states or more than 1 states. It can contain ϵ move. It can be represented as $M = \{ Q, \Sigma, \delta, q_0, F \}$.

Where

1. Q : finite set of states
2. Σ : finite set of the input symbol
3. q_0 : initial state
4. F : **final** state
5. δ : Transition function

NFA with ϵ move: If any FA contains ϵ transition or move, the finite automata is called NFA with ϵ move.

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

Steps for converting NFA with ϵ to DFA:

Step 1: We will take the ϵ -closure for the starting state of NFA as a starting state of DFA.

Step 2: Find the states for each input symbol that can be traversed from the present. That means the union of transition value and their closures for each state of NFA present in the current state of DFA.

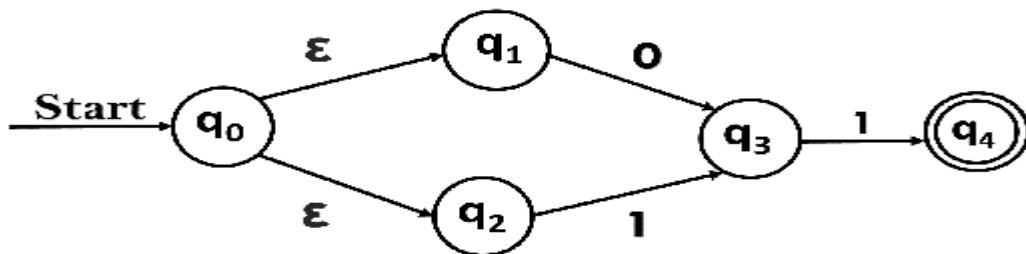
Step 3: If we found a new state, take it as current state and repeat step 2.

Step 4: Repeat Step 2 and Step 3 until there is no new state present in the transition table of DFA.

Step 5: Mark the states of DFA as a final state which contains the final state 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 $\{q_0\} = \{q_0, q_1, q_2\}$
2. ϵ -closure $\{q_1\} = \{q_1\}$
3. ϵ -closure $\{q_2\} = \{q_2\}$
4. ϵ -closure $\{q_3\} = \{q_3\}$
5. ϵ -closure $\{q_4\} = \{q_4\}$

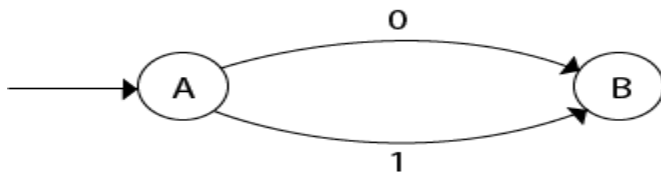
Now, let ϵ -closure $\{q_0\} = \{q_0, q_1, q_2\}$ be state A.

Hence

$$\begin{aligned}
 \delta'(A, 0) &= \epsilon\text{-closure} \{ \delta((q_0, q_1, q_2), 0) \} \\
 &= \epsilon\text{-closure} \{ \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0) \} \\
 &= \epsilon\text{-closure} \{ q_3 \} \\
 &= \{ q_3 \} \quad \text{call it as state B.}
 \end{aligned}$$

$$\begin{aligned}
 \delta'(A, 1) &= \varepsilon\text{-closure} \{ \delta((q_0, q_1, q_2), 1) \} \\
 &= \varepsilon\text{-closure} \{ \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1) \} \\
 &= \varepsilon\text{-closure} \{ q_3 \} \\
 &= \{ q_3 \} = B.
 \end{aligned}$$

The partial DFA will be



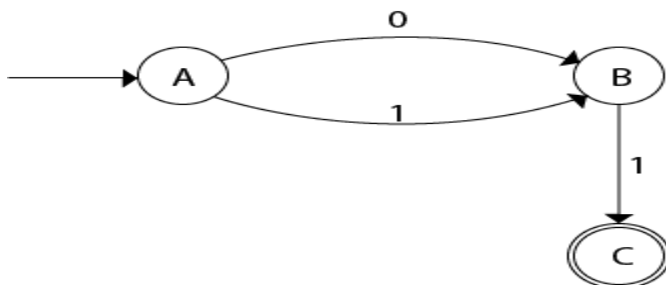
Now,

$$\begin{aligned}
 \delta'(B, 0) &= \varepsilon\text{-closure} \{ \delta(q_3, 0) \} \\
 &= \phi \\
 \delta'(B, 1) &= \varepsilon\text{-closure} \{ \delta(q_3, 1) \} \\
 &= \varepsilon\text{-closure} \{ q_4 \} \\
 &= \{ q_4 \} \quad \textbf{i.e. state C}
 \end{aligned}$$

For state C:

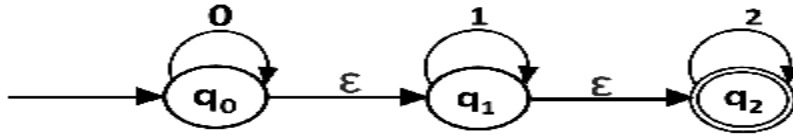
1. $\delta'(C, 0) = \varepsilon\text{-closure} \{ \delta(q_4, 0) \}$
2. $= \phi$
3. $\delta'(C, 1) = \varepsilon\text{-closure} \{ \delta(q_4, 1) \}$
4. $= \phi$

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(q_0) = { q_0 , q_1 , q_2 }
2. ϵ -closure(q_1) = { q_1 , q_2 }
3. ϵ -closure(q_2) = { q_2 }

Now we will obtain δ' transition. Let ϵ -closure(q_0) = { q_0 , q_1 , q_2 } call it as **state A**.

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

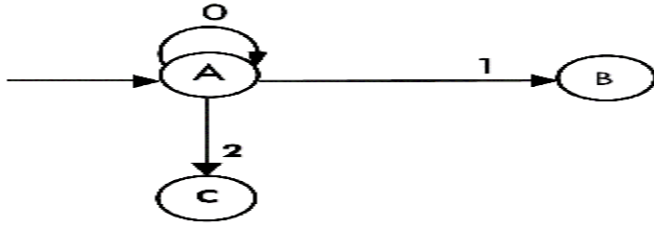
$$\begin{aligned}
 \delta'(A, 1) &= \epsilon\text{-closure}\{\delta((q_0, q_1, q_2), 1)\} \\
 &= \epsilon\text{-closure}\{\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)\} \\
 &= \epsilon\text{-closure}\{q_1\} \\
 &= \{q_1, q_2\} \quad \text{call it as state B}
 \end{aligned}$$

$$\begin{aligned}
 \delta'(A, 2) &= \epsilon\text{-closure}\{\delta((q_0, q_1, q_2), 2)\} \\
 &= \epsilon\text{-closure}\{\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2)\} \\
 &= \epsilon\text{-closure}\{q_2\} \\
 &= \{q_2\} \quad \text{call it state C}
 \end{aligned}$$

Thus we have obtained

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

The partial DFA will be:



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

Hence

$$\begin{aligned}
 \delta'(B, 0) &= \varepsilon\text{-closure}\{\delta((q1, q2), 0)\} \\
 &= \varepsilon\text{-closure}\{\delta(q1, 0) \cup \delta(q2, 0)\} \\
 &= \varepsilon\text{-closure}\{\phi\} \\
 &= \phi
 \end{aligned}$$

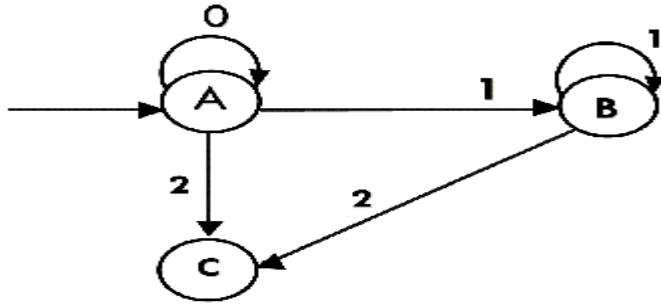
$$\begin{aligned}
 \delta'(B, 1) &= \varepsilon\text{-closure}\{\delta((q1, q2), 1)\} \\
 &= \varepsilon\text{-closure}\{\delta(q1, 1) \cup \delta(q2, 1)\} \\
 &= \varepsilon\text{-closure}\{q1\} \\
 &= \{q1, q2\} \quad \textbf{i.e. state B itself}
 \end{aligned}$$

$$\begin{aligned}
 \delta'(B, 2) &= \varepsilon\text{-closure}\{\delta((q1, q2), 2)\} \\
 &= \varepsilon\text{-closure}\{\delta(q1, 2) \cup \delta(q2, 2)\} \\
 &= \varepsilon\text{-closure}\{q2\} \\
 &= \{q2\} \quad \textbf{i.e. state C itself}
 \end{aligned}$$

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



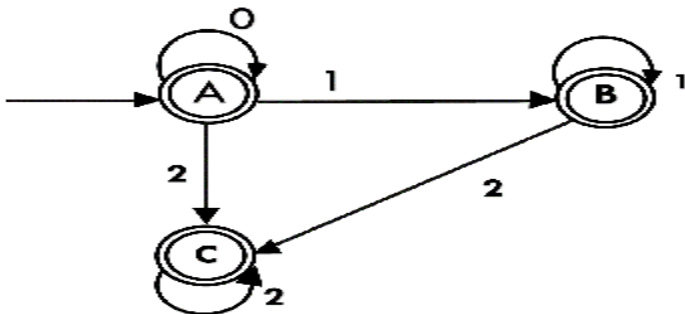
Now we will obtain transitions for C:

$$\begin{aligned}\delta'(C, 0) &= \varepsilon\text{-closure}\{\delta(q_2, 0)\} \\ &= \varepsilon\text{-closure}\{\phi\} \\ &= \phi\end{aligned}$$

$$\begin{aligned}\delta'(C, 1) &= \varepsilon\text{-closure}\{\delta(q_2, 1)\} \\ &= \varepsilon\text{-closure}\{\phi\} \\ &= \phi\end{aligned}$$

$$\begin{aligned}\delta'(C, 2) &= \varepsilon\text{-closure}\{\delta(q_2, 2)\} \\ &= \{q_2\}\end{aligned}$$

Hence the DFA is



As $A = \{q_0, q_1, q_2\}$ in which final state q_2 lies hence A is final state. $B = \{q_1, q_2\}$ in which the state q_2 lies hence B is also final state. $C = \{q_2\}$, the state q_2 lies hence C is also a final state.

Minimization of DFA

Minimization of DFA means reducing the number of states from given FA. Thus, we get the FSM(finite state machine) with redundant states after minimizing the FSM.

We have to follow the various steps to minimize the DFA. These are as follows:

Step 1: Remove all the states that are unreachable from the initial state via any set of the transition of DFA.

Step 2: Draw the transition table for all pair of states.

Step 3: Now split the transition table into two tables T1 and T2. T1 contains all final states, and T2 contains non-final states.

Step 4: Find similar rows from T1 such that:

1. $\delta(q, a) = p$
2. $\delta(r, a) = p$

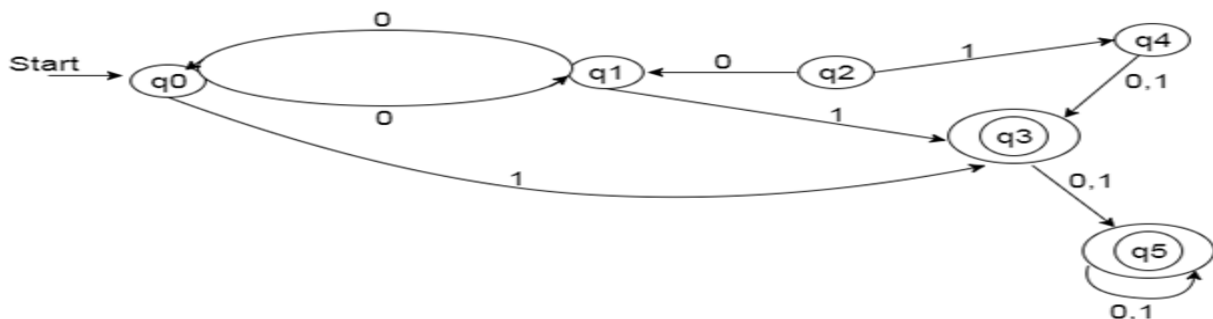
That means, find the two states which have the same value of a and b and remove one of them.

Step 5: Repeat step 3 until we find no similar rows available in the transition table T1.

Step 6: Repeat step 3 and step 4 for table T2 also.

Step 7: Now combine the reduced T1 and T2 tables. The combined transition table is the transition table of minimized DFA.

Example:



Solution:

Step 1: In the given DFA, q2 and q4 are the unreachable states so remove them.

Step 2: Draw the transition table for the rest of the states.

State	0	1
→q0	q1	q3
q1	q0	q3
*q3	q5	q5
*q5	q5	q5

Step 3: Now divide rows of transition table into two sets as:

1. One set contains those rows, which start from non-final states:

State	0	1
q0	q1	q3
q1	q0	q3

2. Another set contains those rows, which starts from final states.

State	0	1
q3	q5	q5
q5	q5	q5

Step 4: Set 1 has no similar rows so set 1 will be the same.

Step 5: In set 2, row 1 and row 2 are similar since q3 and q5 transit to the same state on 0 and 1. So skip q5 and then replace q5 by q3 in the rest.

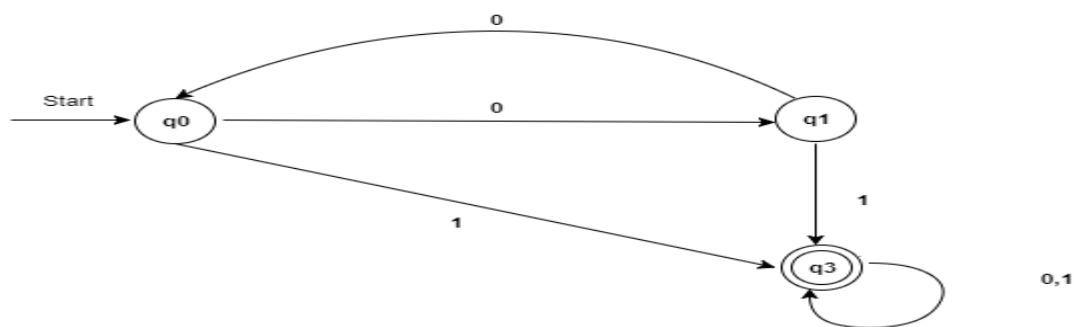
State	0	1
-------	---	---

q3	q3	q3
----	----	----

Step 6: Now combine set 1 and set 2 as:

State	0	1
→q0	q1	q3
q1	q0	q3
*q3	q3	q3

Now it is the transition table of minimized DFA.



Regular Expression

- The language accepted by finite automata can be easily described by simple expressions called Regular Expressions. It is the most effective way to represent any language.
- The languages accepted by some regular expression are referred to as Regular languages.
- A regular expression can also be described as a sequence of pattern that defines a string.
- Regular expressions are used to match character combinations in strings. String searching algorithm used this pattern to find the operations on a string.

For instance:

In a regular expression, x^* means zero or more occurrence of x . It can generate $\{e, x, xx, xxx, xxxx, \dots\}$

In a regular expression, x^+ means one or more occurrence of x . It can generate $\{x, xx, xxx, xxxx, \dots\}$

Operations on Regular Language

The various operations on regular language are:

Union: If L and M are two regular languages then their union $L \cup M$ is also a union.

1. $L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$

Intersection: If L and M are two regular languages then their intersection is also an intersection.

1. $L \cap M = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$

Kleen closure: If L is a regular language then its Kleen closure L^* will also be a regular language.

1. $L^* = \text{Zero or more occurrence of language } L.$

Example 1:

Write the regular expression for the language accepting all combinations of a 's, over the set $\Sigma = \{a\}$

Solution:

All combinations of a 's means a may be zero, single, double and so on. If a is appearing zero times, that means a null string. That is we expect the set of $\{\epsilon, a, aa, aaa, \dots\}$. So we give a regular expression for this as:

1. $R = a^*$

That is Kleen closure of a .

Example 2:

Write the regular expression for the language accepting all combinations of a 's except the null string, over the set $\Sigma = \{a\}$

Solution:

The regular expression has to be built for the language

$$1. L = \{a, aa, aaa, \dots\}$$

This set indicates that there is no null string. So we can denote regular expression as:

$$R = a^+$$

Example 3:

Write the regular expression for the language accepting all the string containing any number of a's and b's.

Solution:

The regular expression will be:

$$1. \text{ r.e. } = (a + b)^*$$

This will give the set as $L = \{\epsilon, a, aa, b, bb, ab, ba, aba, bab, \dots\}$, any combination of a and b.

The $(a + b)^*$ shows any combination with a and b even a null string.

Examples of Regular Expression

Example 1:

Write the regular expression for the language accepting all the string which are starting with 1 and ending with 0, over $\Sigma = \{0, 1\}$.

Solution:

In a regular expression, the first symbol should be 1, and the last symbol should be 0. The r.e. is as follows:

$$1. R = 1(0+1)^*0$$

Example 2:

Write the regular expression for the language starting and ending with a and having any having any combination of b's in between.

Solution:

The regular expression will be:

1. $R = a b^* a$

Example 3:

Write the regular expression for the language starting with a but not having consecutive b's.

Solution: The regular expression has to be built for the language:

1. $L = \{a, aba, aab, aba, aaa, abab, \dots\}$

The regular expression for the above language is:

1. $R = \{a + ab\}^*$

Example 4:

Write the regular expression for the language accepting all the string in which any number of a's is followed by any number of b's is followed by any number of c's.

Solution: As we know, any number of a's means a^* any number of b's means b^* , any number of c's means c^* . Since as given in problem statement, b's appear after a's and c's appear after b's. So the regular expression could be:

1. $R = a^* b^* c^*$

Example 5:

Write the regular expression for the language over $\Sigma = \{0\}$ having even length of the string.

Solution:

The regular expression has to be built for the language:

$$1. L = \{\epsilon, 00, 0000, 000000, \dots\}$$

The regular expression for the above language is:

$$1. R = (00)^*$$

Example 6:

Write the regular expression for the language having a string which should have atleast one 0 and atleast one 1.

Solution:

The regular expression will be:

$$1. R = [(0 + 1)^* 0 (0 + 1)^* 1 (0 + 1)^*] + [(0 + 1)^* 1 (0 + 1)^* 0 (0 + 1)^*]$$

Example 7:

Describe the language denoted by following regular expression

$$1. \text{r.e.} = (b^* (aaa)^* b^*)^*$$

Solution:

The language can be predicted from the regular expression by finding the meaning of it. We will first split the regular expression as:

$$\text{r.e.} = (\text{any combination of b's}) (aaa)^* (\text{any combination of b's})$$

$L = \{\text{The language consists of the string in which a's appear triples, there is no restriction on the number of b's}\}$

Example 8:

Write the regular expression for the language L over $\Sigma = \{0, 1\}$ such that all the string do not contain the substring 01.

Solution:

The Language is as follows:

$$1. L = \{\epsilon, 0, 1, 00, 11, 10, 100, \dots\}$$

The regular expression for the above language is as follows:

$$1. R = (1^* 0^*)$$

Example 9:

Write the regular expression for the language containing the string over $\{0, 1\}$ in which there are at least two occurrences of 1's between any two occurrences of 1's between any two occurrences of 0's.

Solution: At least two 1's between two occurrences of 0's can be denoted by $(0111^*0)^*$.

Similarly, if there is no occurrence of 0's, then any number of 1's are also allowed. Hence the r.e. for required language is:

$$1. R = (1 + (0111^*0))^*$$

Example 10:

Write the regular expression for the language containing the string in which every 0 is immediately followed by 11.

Solution:

The regular expectation will be:

$$1. R = (011 + 1)^*$$

Conversion of RE to FA

To convert the RE to FA, we are going to use a method called the subset method. This method is used to obtain FA from the given regular expression. This method is given below:

Step 1: Design a transition diagram for given regular expression, using NFA with ϵ moves.

Step 2: Convert this NFA with ϵ to NFA without ϵ .

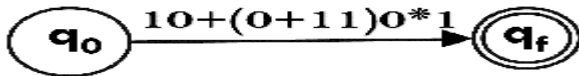
Step 3: Convert the obtained NFA to equivalent DFA.

Example 1:

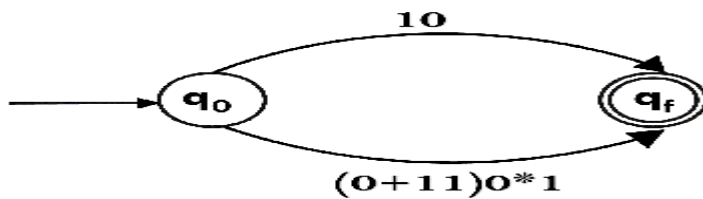
Design a FA from given regular expression $10 + (0 + 11)0^*1$.

Solution: First we will construct the transition diagram for a given regular expression.

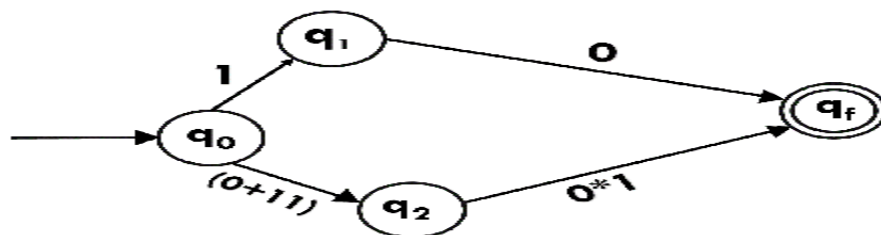
Step 1:



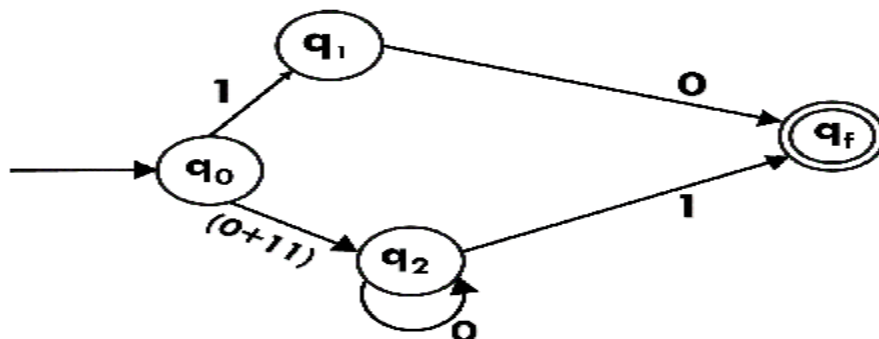
Step 2:



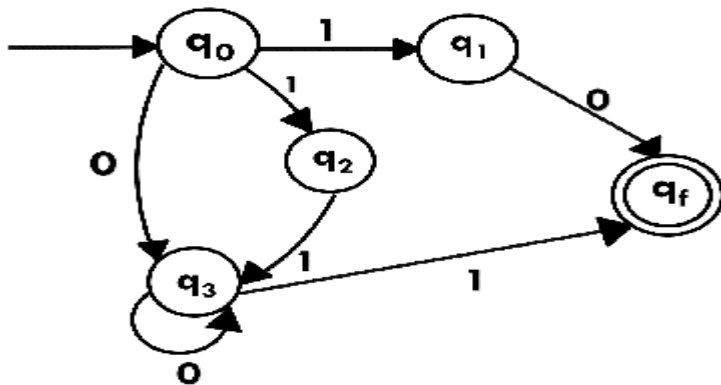
Step 3:



Step 4:



Step 5:



Now we have got NFA without ϵ . Now we will convert it into required DFA for that, we will first write a transition table for this NFA.

State	0	1
$\rightarrow q_0$	q_3	$\{q_1, q_2\}$
q_1	q_f	\emptyset
q_2	\emptyset	q_3
q_3	q_3	q_f
$*q_f$	\emptyset	\emptyset

The equivalent DFA will be:

State	0	1
$\rightarrow [q_0]$	$[q_3]$	$[q_1, q_2]$
$[q_1]$	$[q_f]$	\emptyset

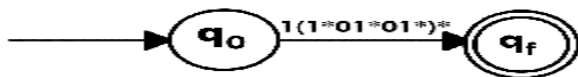
[q2]	Φ	[q3]
[q3]	[q3]	[qf]
[q1, q2]	[qf]	[qf]
*[qf]	Φ	Φ

Example 2:

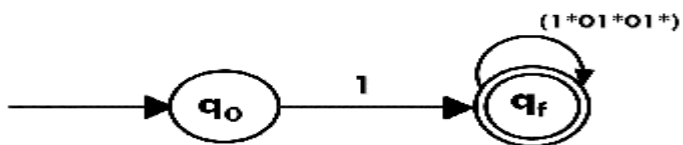
Design a NFA from given regular expression $1(1^*01^*01^*)^*$.

Solution: The NFA for the given regular expression is as follows:

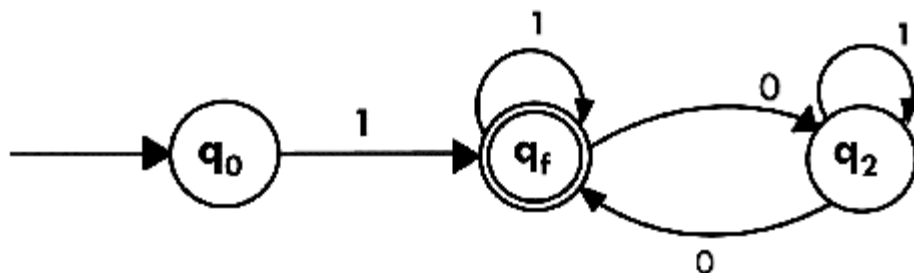
Step 1:



Step 2:



Step 3:



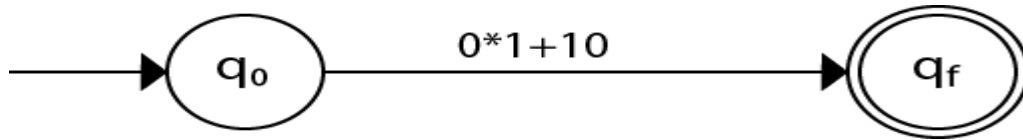
Example 3:

Construct the FA for regular expression $0^*1 + 10$.

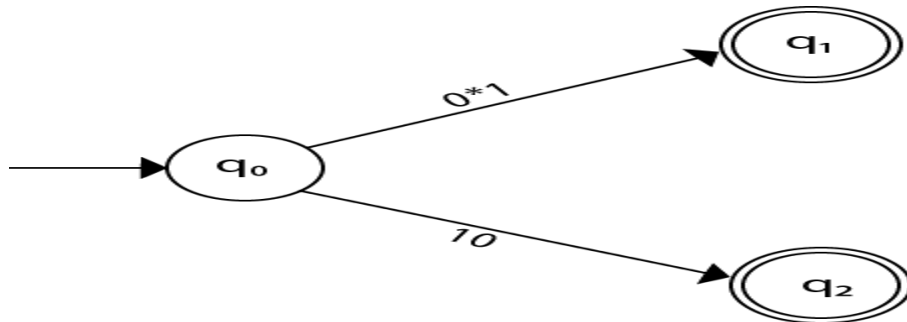
Solution:

We will first construct FA for $R = 0^*1 + 10$ as follows:

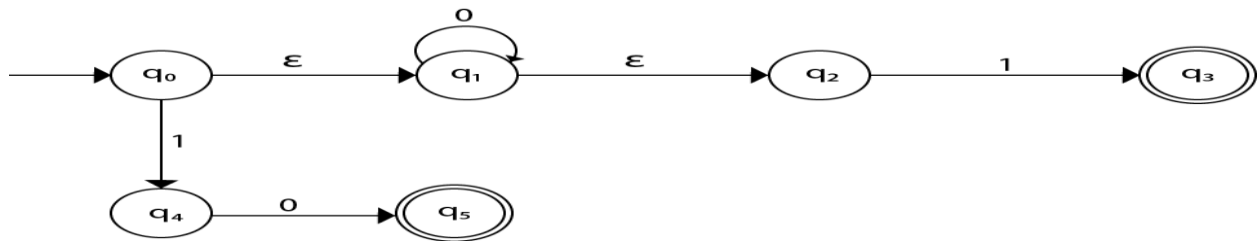
Step 1:



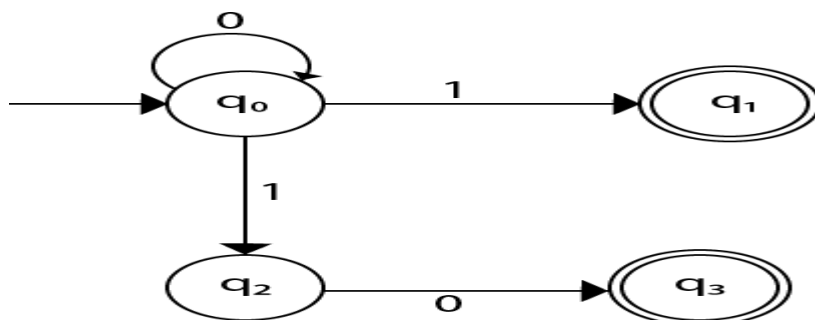
Step 2:



Step 3:



Step 4:

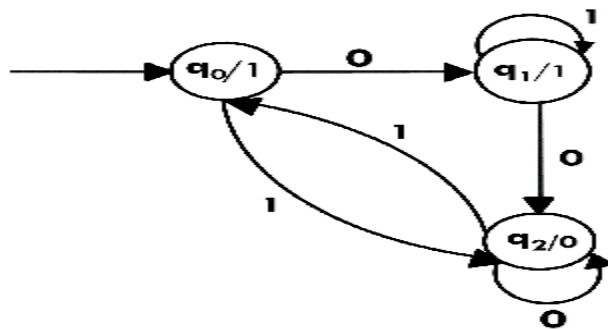


Moore Machine

Moore machine is a finite state machine in which the next state is decided by the current state and current input symbol. The output symbol at a given time depends only on the present state of the machine. Moore machine can be described by 6 tuples $(Q, q_0, \Sigma, O, \delta, \lambda)$ where,

1. Q : finite set of states
2. q_0 : initial state of machine
3. Σ : finite set of input symbols
4. O : output alphabet
5. δ : transition function where $Q \times \Sigma \rightarrow Q$
6. λ : output function where $Q \rightarrow O$

Example 1: The state diagram for Moore Machine is



Transition table for Moore Machine is:

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

In the above Moore machine, the output is represented with each input state separated by /. The output length for a Moore machine is greater than input by 1.

Input: 010

Transition: $\delta(q_0, 0) \Rightarrow \delta(q_1, 1) \Rightarrow \delta(q_1, 0) \Rightarrow q_2$

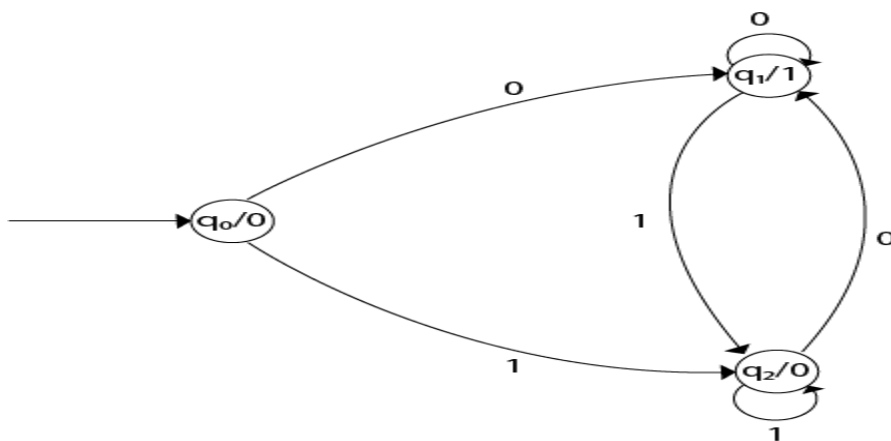
Output: 1110 (1 for q_0 , 1 for q_1 , again 1 for q_1 , 0 for q_2)

Example 2:

Design a Moore machine to generate 1's complement of a given binary number.

Solution: To generate 1's complement of a given binary number the simple logic is that if the input is 0 then the output will be 1 and if the input is 1 then the output will be 0. That means there are three states. One state is start state. The second state is for taking 0's as input and produces output as 1. The third state is for taking 1's as input and producing output as 0.

Hence the Moore machine will be,



For instance, take one binary number 1011 then

Input		1	0	1	1
State	q0	q2	q1	q2	q2
Output	0	0	1	0	0

Thus we get 00100 as 1's complement of 1011, we can neglect the initial 0 and the output which we get is 0100 which is 1's complement of 1011. The transaction table is as follows:

Current State	δ		λ
	0	1	
$\rightarrow q_0$	q_1	q_2	0
q_1	q_1	q_2	1
q_2	q_1	q_2	0

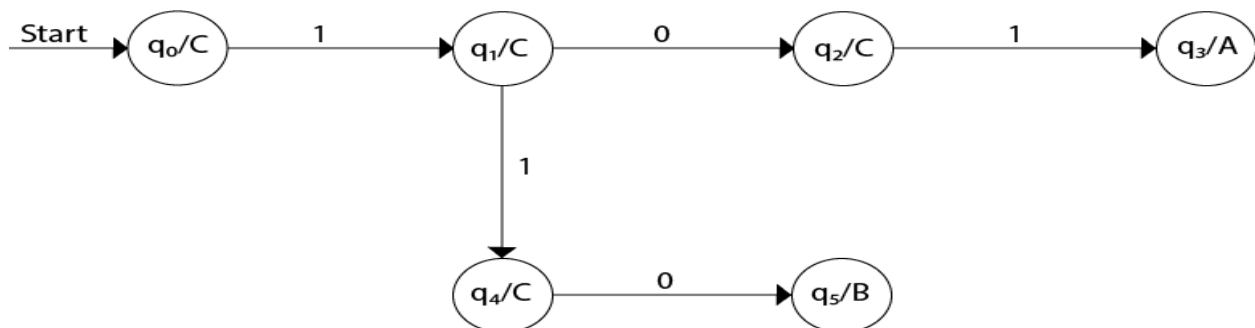
Thus Moore machine $M = (Q, q_0, \Sigma, O, \delta, \lambda)$; where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, $O = \{0, 1\}$. the transition table shows the δ and λ functions.

Example 3:

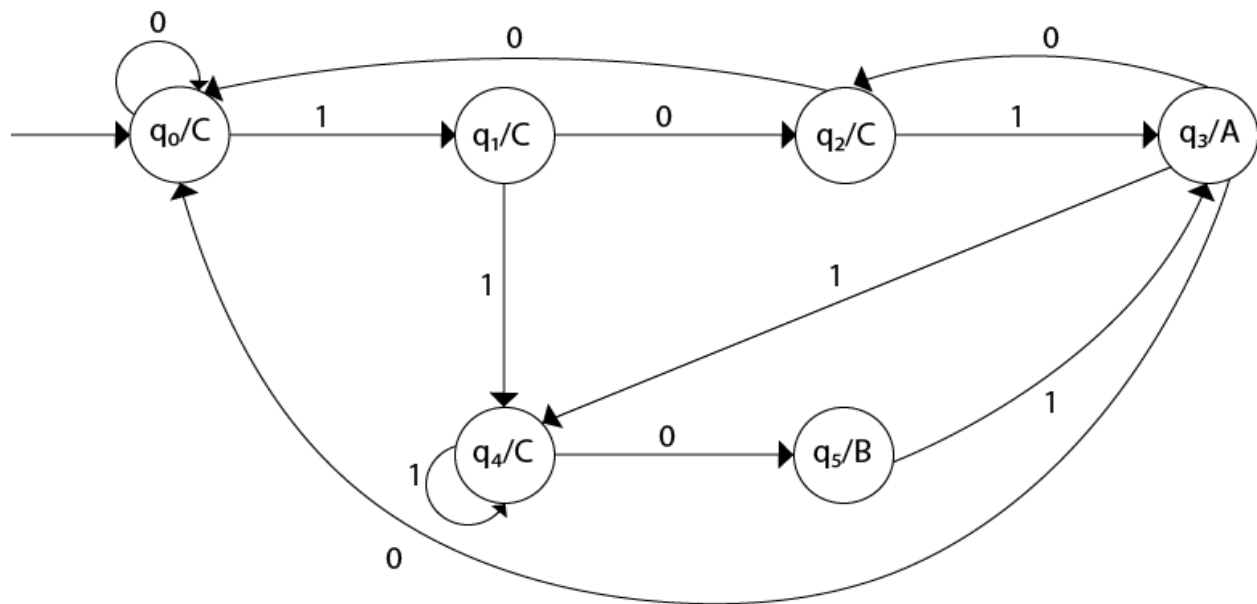
Design a Moore machine for a binary input sequence such that if it has a substring 101, the machine output A, if the input has substring 110, it outputs B otherwise it outputs C.

Solution: For designing such a machine, we will check two conditions, and those are 101 and 110. If we get 101, the output will be A, and if we recognize 110, the output will be B. For other strings, the output will be C.

The partial diagram will be:



Now we will insert the possibilities of 0's and 1's for each state. Thus the Moore machine becomes:

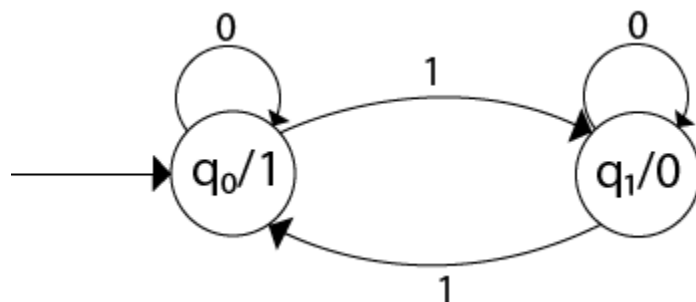


Example 4:

Construct a Moore machine that determines whether an input string contains an even or odd number of 1's. The machine should give 1 as output if an even number of 1's are in the string and 0 otherwise.

Solution:

The Moore machine will be:



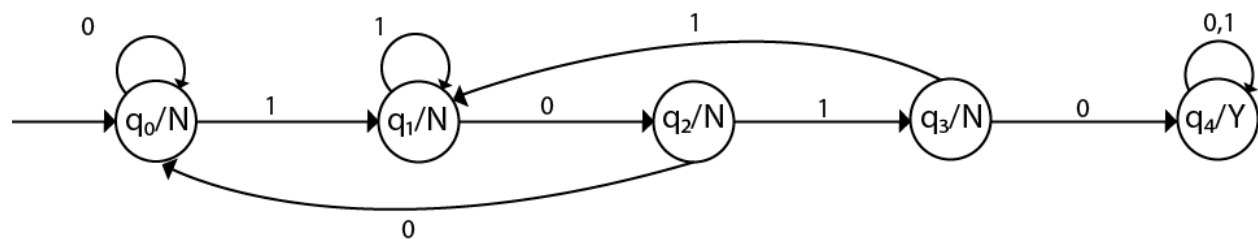
This is the required Moore machine. In this machine, state q_1 accepts an odd number of 1's and state q_0 accepts even number of 1's. There is no restriction on a number of zeros. Hence for 0 input, self-loop can be applied on both the states.

Example 5:

Design a Moore machine with the input alphabet $\{0, 1\}$ and output alphabet $\{Y, N\}$ which produces Y as output if input sequence contains 1010 as a substring otherwise, it produces N as output.

Solution:

The Moore machine will be:



Mealy Machine

A Mealy machine is a machine in which output symbol depends upon the present input symbol and present state of the machine. In the Mealy machine, the output is represented with each input symbol for each state separated by /. The Mealy machine can be described by 6 tuples $(Q, q_0, \Sigma, O, \delta, \lambda')$ where

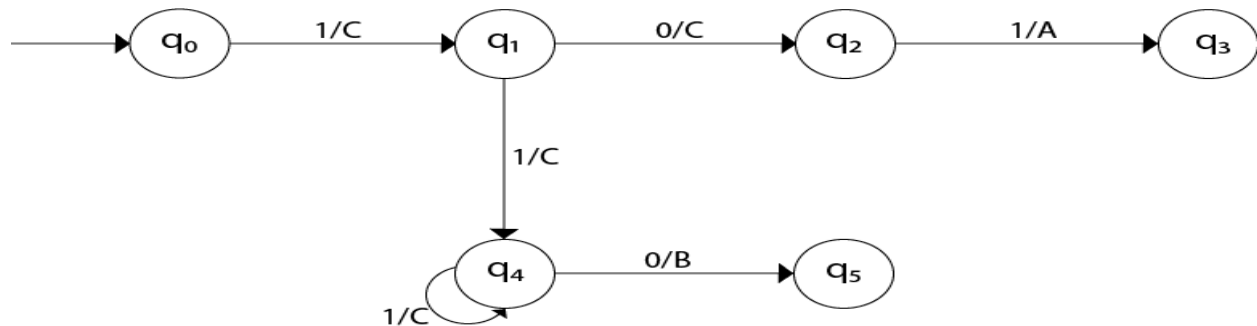
1. Q : finite set of states
2. q_0 : initial state of machine
3. Σ : finite set of input alphabet
4. O : output alphabet
5. δ : transition function where $Q \times \Sigma \rightarrow Q$
6. λ' : output function where $Q \times \Sigma \rightarrow O$

Example 1:

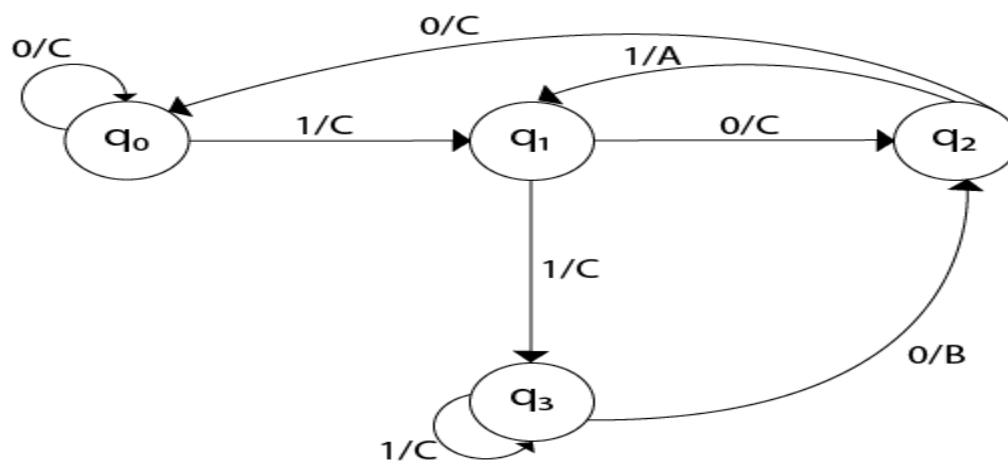
Design a Mealy machine for a binary input sequence such that if it has a substring 101, the machine output A, if the input has substring 110, it outputs B otherwise it outputs C.

Solution: For designing such a machine, we will check two conditions, and those are 101 and 110. If we get 101, the output will be A. If we recognize 110, the output will be B. For other strings the output will be C.

The partial diagram will be:



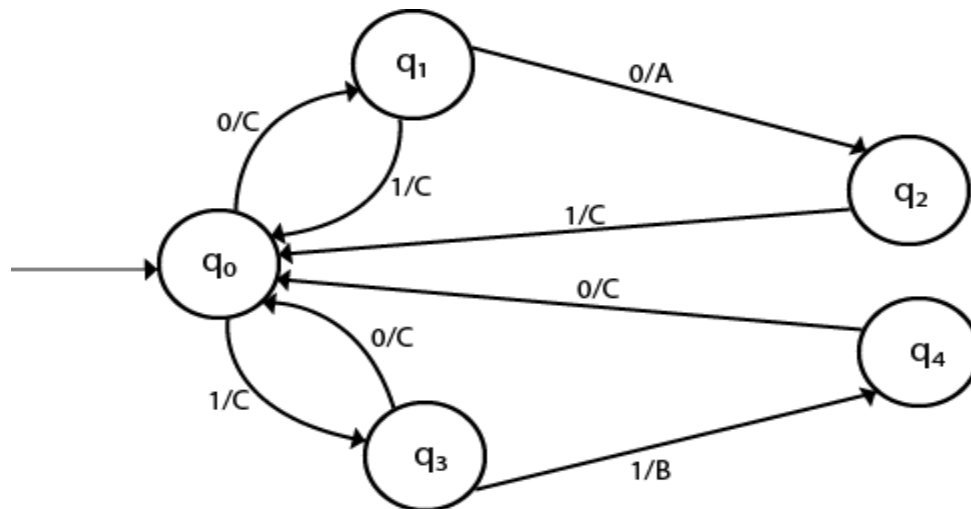
Now we will insert the possibilities of 0's and 1's for each state. Thus the Mealy machine becomes:



Example 2:

Design a mealy machine that scans sequence of input of 0 and 1 and generates output 'A' if the input string terminates in 00, output 'B' if the string terminates in 11, and output 'C' otherwise.

Solution: The mealy machine will be:



Conversion from Mealy machine to Moore Machine

In Moore machine, the output is associated with every state, and in Mealy machine, the output is given along the edge with input symbol. To convert Moore machine to Mealy machine, state output symbols are distributed to input symbol paths. But while converting the Mealy machine to Moore machine, we will create a separate state for every new output symbol and according to incoming and outgoing edges are distributed.

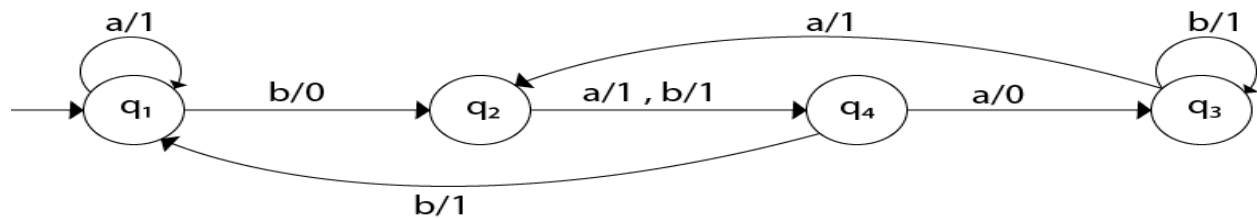
The following steps are used for converting Mealy machine to the Moore machine:

Step 1: For each state(Q_i), calculate the number of different outputs that are available in the transition table of the Mealy machine.

Step 2: Copy state Q_i , if all the outputs of Q_i are the same. Break q_i into n states as Q_{in} , if it has n distinct outputs where $n = 0, 1, 2..$

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

Example 1: Convert the following Mealy machine into equivalent Moore machine.



Solution:

Transition table for above Mealy machine is as follows:

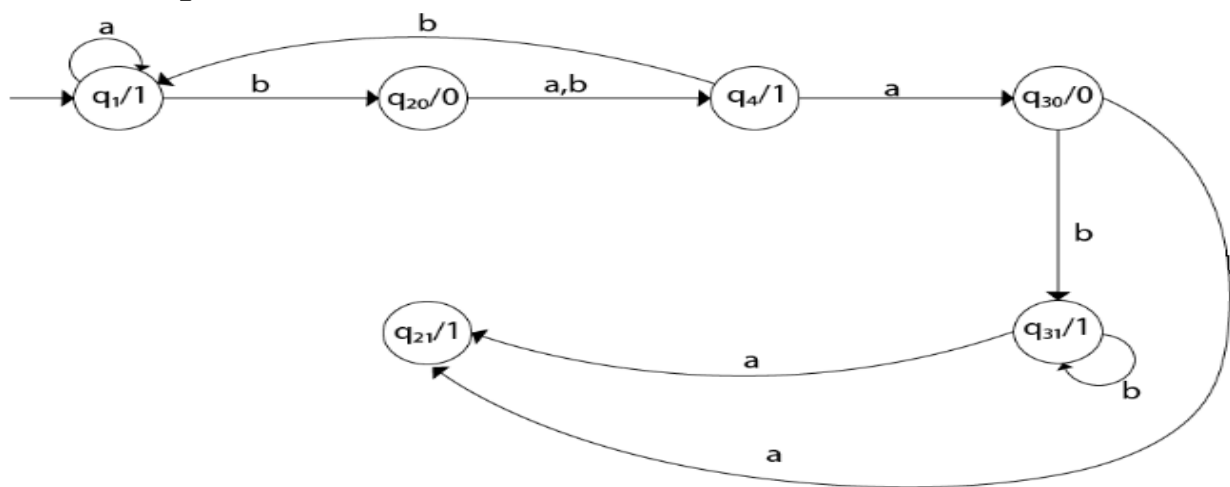
Present State	Next State			
	a		b	
	State	O/P	State	O/P
q ₁	q ₁	1	q ₂	0
q ₂	q ₄	1	q ₄	1
q ₃	q ₂	1	q ₃	1
q ₄	q ₃	0	q ₁	1

- For state q₁, there is only one incident edge with output 0. So, we don't need to split this state in Moore machine.
- For state q₂, there is 2 incident edge with output 0 and 1. So, we will split this state into two states q₂₀(state with output 0) and q₂₁(with output 1).
- For state q₃, there is 2 incident edge with output 0 and 1. So, we will split this state into two states q₃₀(state with output 0) and q₃₁(state with output 1).
- For state q₄, there is only one incident edge with output 0. So, we don't need to split this state in Moore machine.

Transition table for Moore machine will be:

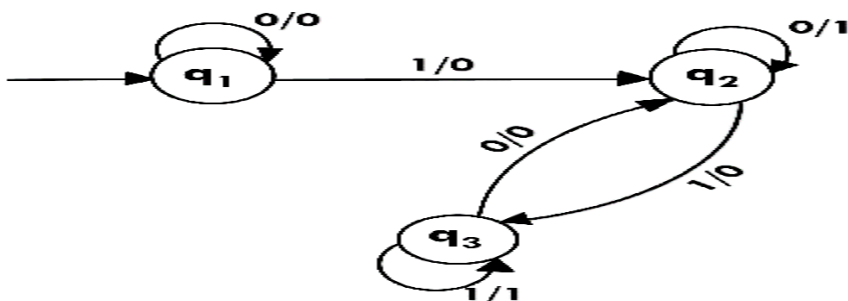
Present State	Next State		Output
	a=0	a=1	
q ₁	q ₁	q ₂	1
q ₂₀	q ₄	q ₄	0
q ₂₁	∅	∅	1
q ₃₀	q ₂₁	q ₃₁	0
q ₃₁	q ₂₁	q ₃₁	1
q ₄	q ₃	q ₄	1

Transition diagram for Moore machine will be:



Example 2:

Convert the following Mealy machine into equivalent Moore machine.



Solution:

Transition table for above Mealy machine is as follows:

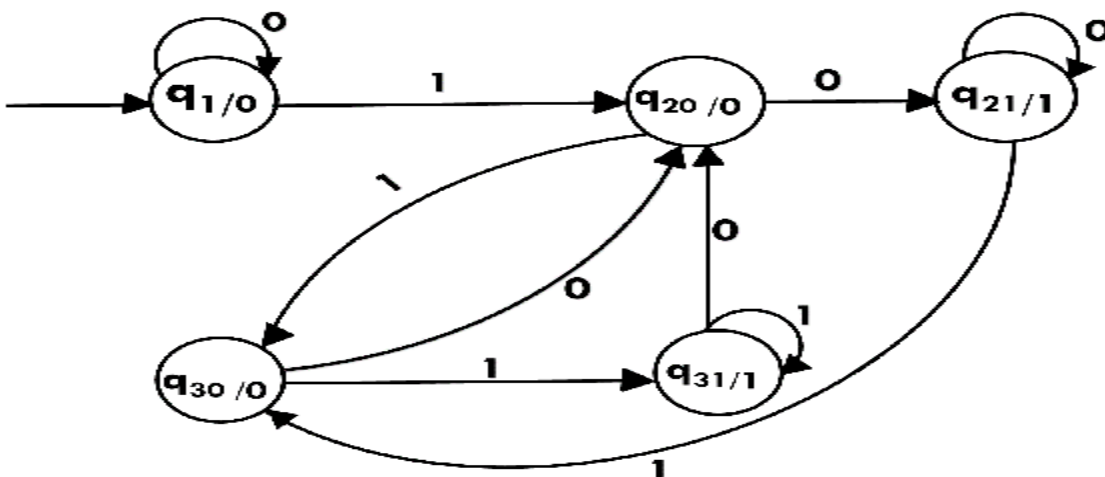
Present State	Next State 0		Next State 1	
	State	o/P	State	o/P
q₁	q₁	0	q₂	0
q₂	q₂	1	q₃	0
q₃	q₂	0	q₃	1

The state q₁ has only one output. The state q₂ and q₃ have both output 0 and 1. So we will create two states for these states. For q₂, two states will be q₂₀(with output 0) and q₂₁(with output 1). Similarly, for q₃ two states will be q₃₀(with output 0) and q₃₁(with output 1).

Transition table for Moore machine will be:

Present State	Next State 0	Next State 1	o/P
q₁	q₁	q₂₀	0
q₂₀	q₂₁	q₃₀	0
q₂₁	q₂₁	q₃₀	1
q₃₀	q₂₀	q₃₁	0
q₃₁	q₂₀	q₃₁	1

Transition diagram for Moore machine will be:



Conversion from Moore machine to Mealy Machine

In the Moore machine, the output is associated with every state, and in the mealy machine, the output is given along the edge with input symbol. The equivalence of the Moore machine and Mealy machine means both the machines generate the same output string for same input string.

We cannot directly convert Moore machine to its equivalent Mealy machine because the length of the Moore machine is one longer than the Mealy machine for the given input. To convert Moore machine to Mealy machine, state output symbols are distributed into input symbol paths. We are going to use the following method to convert the Moore machine to Mealy machine.

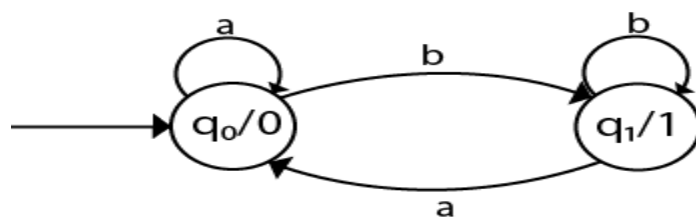
Method for conversion of Moore machine to Mealy machine

Let $M = (Q, \Sigma, \delta, \lambda, q_0)$ be a Moore machine. The equivalent Mealy machine can be represented by $M' = (Q, \Sigma, \delta, \lambda', q_0)$. The output function λ' can be obtained as:

$$1. \lambda'(q, a) = \lambda(\delta(q, a))$$

Example 1:

Convert the following Moore machine into its equivalent Mealy machine.



Solution:

The transition table of given Moore machine is as follows:

Q	A	b	Output(λ)
q0	q0	q1	0
q1	q0	q1	1

The equivalent Mealy machine can be obtained as follows:

1. $\lambda'(q_0, a) = \lambda(\delta(q_0, a))$
2. $= \lambda(q_0)$
3. $= 0$
- 4.
5. $\lambda'(q_0, b) = \lambda(\delta(q_0, b))$
6. $= \lambda(q_1)$
7. $= 1$

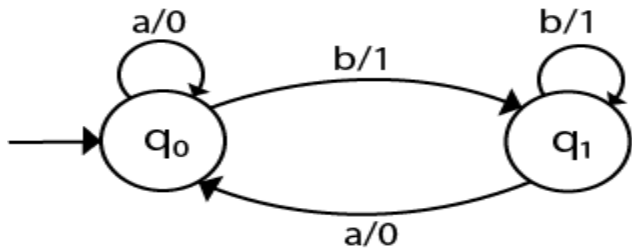
The λ for state q1 is as follows:

1. $\lambda'(q_1, a) = \lambda(\delta(q_1, a))$
2. $= \lambda(q_0)$
3. $= 0$
- 4.
5. $\lambda'(q_1, b) = \lambda(\delta(q_1, b))$
6. $= \lambda(q_1)$
7. $= 1$

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

Q \ Σ	Input 0		Input 1	
	State	O/P	State	O/P
q ₀	q ₀	0	q ₁	1
q ₁	q ₀	0	q ₁	1

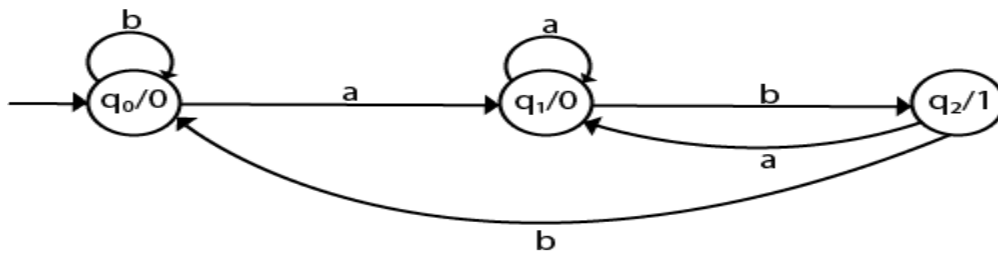
The equivalent Mealy machine will be,



Note: The length of output sequence is 'n+1' in Moore machine and is 'n' in the Mealy machine.

Example 2:

Convert the given Moore machine into its equivalent Mealy machine.



Solution:

The transition table of given Moore machine is as follows:

Q	A	B	Output(λ)
q0	q1	q0	0
q1	q1	q2	0
q2	q1	q0	1

The equivalent Mealy machine can be obtained as follows:

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

2. $= \lambda(q_1)$
3. $= 0$
- 4.
5. $\lambda'(q_0, b) = \lambda(\delta(q_0, b))$
6. $= \lambda(q_0)$
7. $= 0$

The λ for state q_1 is as follows:

1. $\lambda'(q_1, a) = \lambda(\delta(q_1, a))$
2. $= \lambda(q_1)$
3. $= 0$
- 4.
5. $\lambda'(q_1, b) = \lambda(\delta(q_1, b))$
6. $= \lambda(q_2)$
7. $= 1$

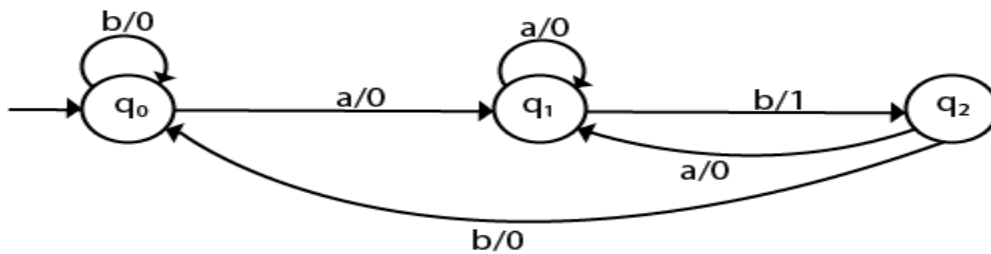
The λ for state q_2 is as follows:

1. $\lambda'(q_2, a) = \lambda(\delta(q_2, a))$
2. $= \lambda(q_1)$
3. $= 0$
- 4.
5. $\lambda'(q_2, b) = \lambda(\delta(q_2, b))$
6. $= \lambda(q_0)$
7. $= 0$

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

<div> <div>Σ</div> <div>Q</div> </div>	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 will be,



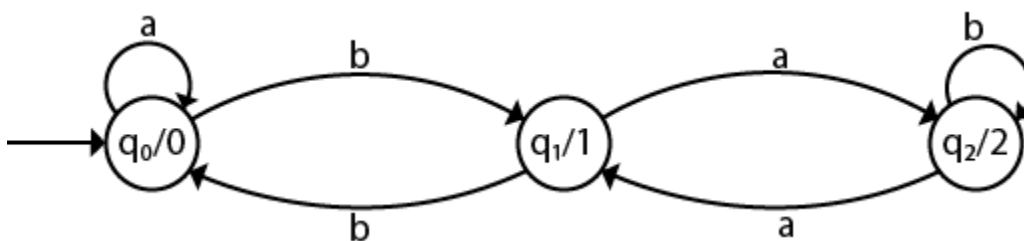
Example 3:

Convert the given Moore machine into its equivalent Mealy machine.

Q		B	Output(λ)
a			
q0	q0	q1	0
q1	q2	q0	1
q2	q1	q2	2

Solution:

The transaction diagram for the given problem can be drawn as:



The equivalent Mealy machine can be obtained as follows:

1. $\lambda' (q_0, a) = \lambda(\delta(q_0, a))$
2. $= \lambda(q_0)$
3. $= 0$
- 4.
5. $\lambda' (q_0, b) = \lambda(\delta(q_0, b))$
6. $= \lambda(q_1)$
7. $= 1$

The λ for state q_1 is as follows:

1. $\lambda' (q_1, a) = \lambda(\delta(q_1, a))$
2. $= \lambda(q_2)$
3. $= 2$
- 4.
5. $\lambda' (q_1, b) = \lambda(\delta(q_1, b))$
6. $= \lambda(q_0)$
7. $= 0$

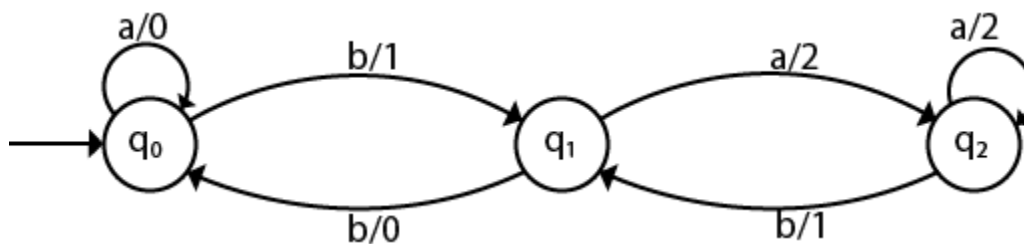
The λ for state q_2 is as follows:

1. $\lambda' (q_2, a) = \lambda(\delta(q_2, a))$
2. $= \lambda(q_1)$
3. $= 1$
- 4.
5. $\lambda' (q_2, b) = \lambda(\delta(q_2, b))$
6. $= \lambda(q_2)$
7. $= 2$

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

<div> <div>Σ</div> <div>Q</div> </div>	Input a		Input b	
	State	O/P	State	O/P
q ₀	q ₀	0	q ₁	1
q ₁	q ₂	2	q ₀	0
q ₂	q ₁	1	q ₂	2

The equivalent Mealy machine will be,



Pumping Lemma For Regular Grammars

Theorem

Let L be a regular language. Then there exists a constant ' c ' such that for every string w in L –

$$|w| \geq c$$

We can break w into three strings, $w = xyz$, such that –

- $|y| > 0$
- $|xy| \leq c$
- For all $k \geq 0$, the string xy^kz is also in L .

Applications of Pumping Lemma

Pumping Lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

- If **L** is regular, it satisfies Pumping Lemma.
- If **L** does not satisfy Pumping Lemma, it is non-regular.

Method to prove that a language **L** is not regular

- At first, we have to assume that **L** is regular.
- So, the pumping lemma should hold for **L**.
- Use the pumping lemma to obtain a contradiction –
 - Select **w** such that $|w| \geq c$
 - Select **y** such that $|y| \geq 1$
 - Select **x** such that $|xy| \leq c$
 - Assign the remaining string to **z**.
 - Select **k** such that the resulting string is not in **L**.

Hence **L** is not regular.

Problem

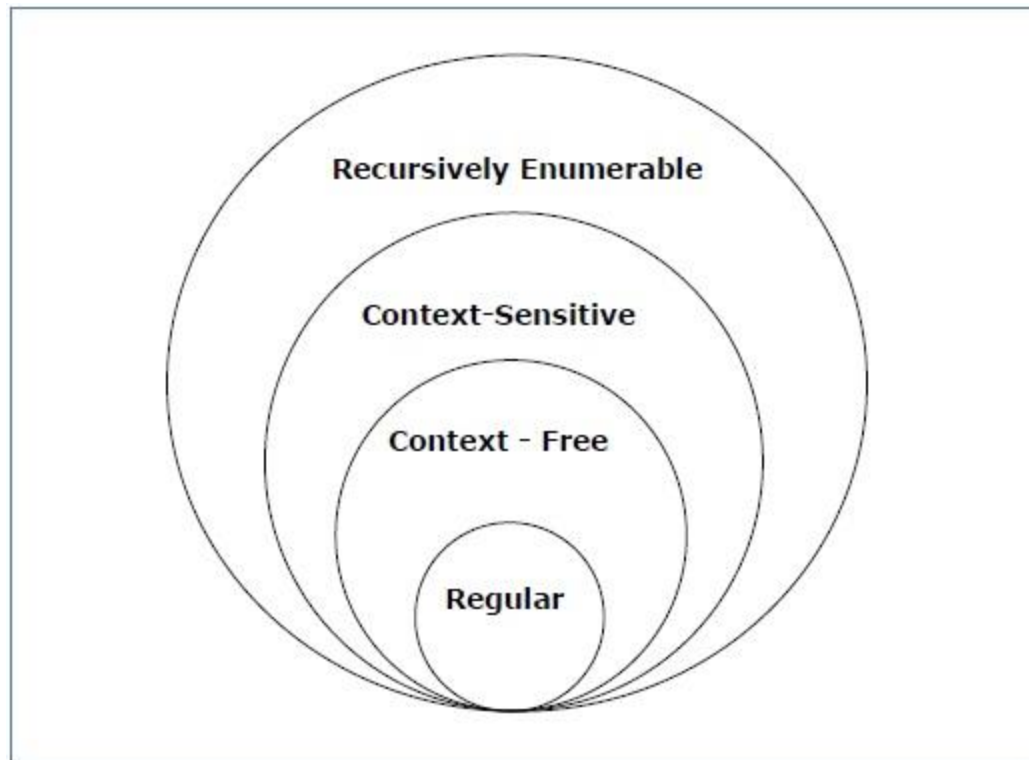
Prove that **L** = $\{a^i b^i \mid i \geq 0\}$ is not regular.

Chomsky Classification of Grammars

According to Noam Chomsky, there are four types of grammars – Type 0, Type 1, Type 2, and Type 3. The following table shows how they differ from each other –

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammar	Recursively enumerable language	Turing Machine
Type 1	Context-sensitive grammar	Context-sensitive language	Linear-bounded automaton
Type 2	Context-free grammar	Context-free language	Pushdown automaton
Type 3	Regular grammar	Regular language	Finite state automaton

Take a look at the following illustration. It shows the scope of each type of grammar –



Type - 3 Grammar

Type-3 grammars generate regular languages. Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.

The productions must be in the form $X \rightarrow a$ or $X \rightarrow aY$

where $X, Y \in N$ (Non terminal)

and $a \in T$ (Terminal)

The rule $S \rightarrow \epsilon$ is allowed if S does not appear on the right side of any rule.

Example

```
X → ε  
X → a | aY  
Y → b
```

Type - 2 Grammar

Type-2 grammars generate context-free languages.

The productions must be in the form $A \rightarrow \gamma$

where $A \in N$ (Non terminal)

and $\gamma \in (T \cup N)^*$ (String of terminals and non-terminals).

These languages generated by these grammars are recognized by a non-deterministic pushdown automaton.

Example

```
S → X a
X → a
X → aX
X → abc
X → ε
```

Type - 1 Grammar

Type-1 grammars generate context-sensitive languages. The productions must be in the form

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

where $A \in N$ (Non-terminal)

and $\alpha, \beta, \gamma \in (T \cup N)^*$ (Strings of terminals and non-terminals)

The strings α and β may be empty, but γ must be non-empty.

The rule $S \rightarrow \epsilon$ is allowed if S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.

Example

```
AB → AbBc
A → bcA
B → b
```

Type - 0 Grammar

Type-0 grammars generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars.

They generate the languages that are recognized by a Turing machine.

The productions can be in the form of $\alpha \rightarrow \beta$ where α is a string of terminals and nonterminals with at least one non-terminal and α cannot be null. β is a string of terminals and non-terminals.

Example

```
S → ACaB
```

Bc → acB
CB → DB
aD → Db