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. An automaton with a finite number of states is called a Finite automaton.

In this tutorial, we are going to learn how to construct deterministic finite automata, non-deterministic finite automata, Regular expression, context-free grammar, context-free language, Push down automata, Turning machines, etc.

Prerequisite

Before learning Automata, you should have a basic understanding of string, language, alphabets, symbols.

Audience

Our Automata Tutorial is designed to help beginners and professionals.

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 ∑.

Examples:

1. ∑ = {a, b}

2. ∑ = {A, B, C, D}

3. ∑ = {0, 1, 2}

4. ∑ = {0, 1, ..., 5]

5. ∑ = {#, β, Δ}

String:

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

Example 1:

If ∑ = {a, b}, various string that can be generated from ∑ are {ab, aa, aaa, bb, bbb, ba, aba.....}.

- 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. 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 = {Set of string of length 2}

= {aa, bb, ba, bb} **Finite Language**

Example: 2

L2 = {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, ∑, δ, q0, F), where:

1. Q: finite set of states
2. ∑: finite set of the input symbol
3. q0: 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.

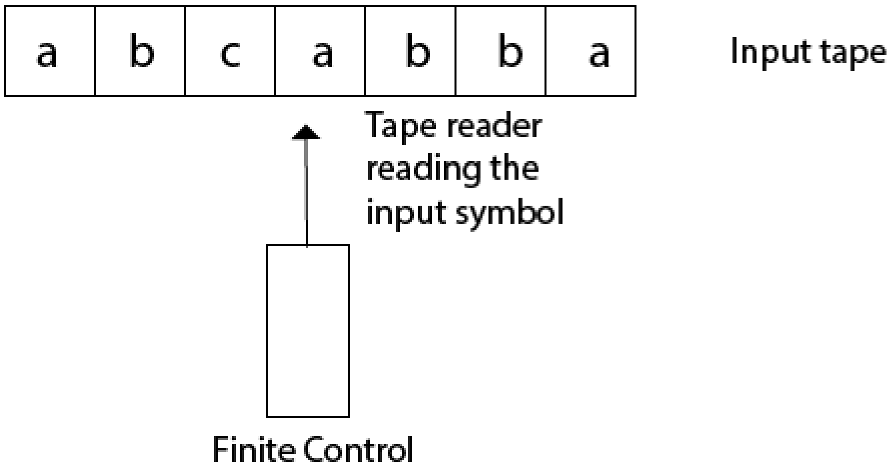
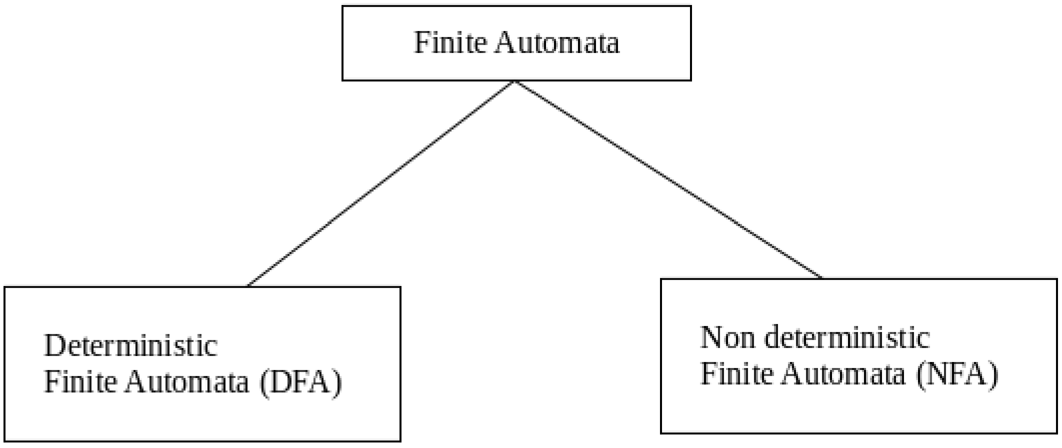


Fig :- Finite automata model

Types of Automata:

There are two types of finite automata:

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


Finite Automata
Deterministic Finite Automata (DFA)
Non deterministic Finite Automata (NFA)

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.

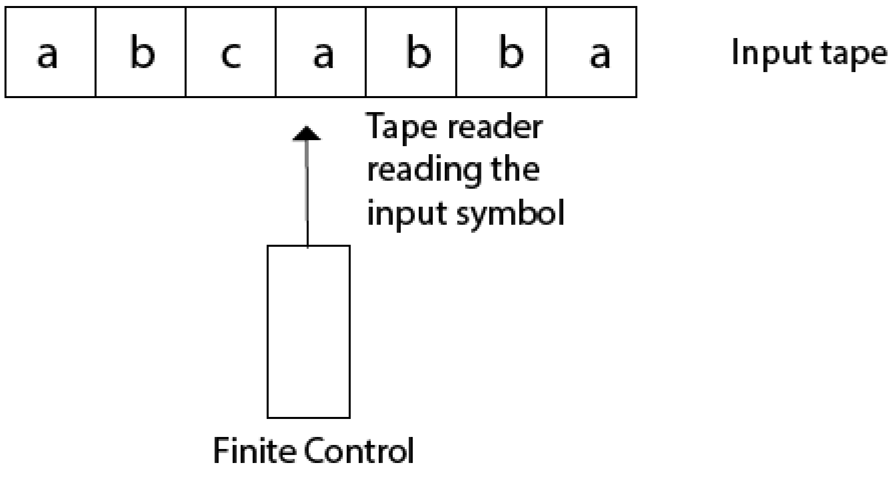


Fig :- Finite automata model

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 δ(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:

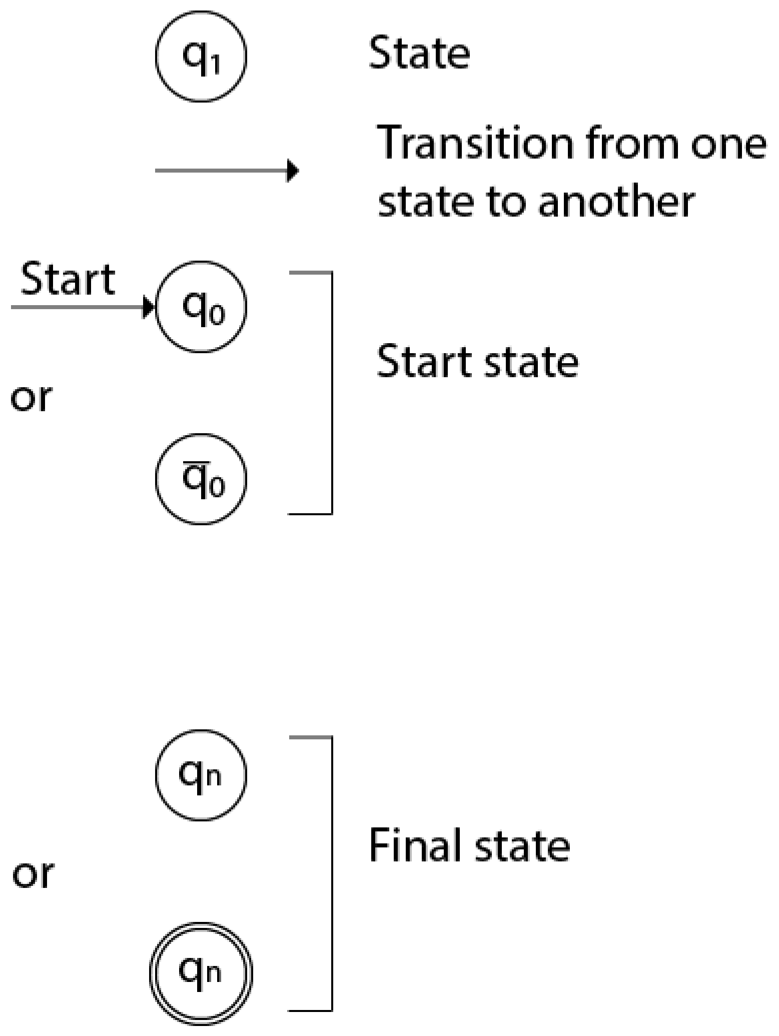


Fig:- Notations

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 δ(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 ∈ 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 ∉ F.

Example 1:

DFA with ∑ = {0, 1} accepts all strings starting with 1.

Solution:

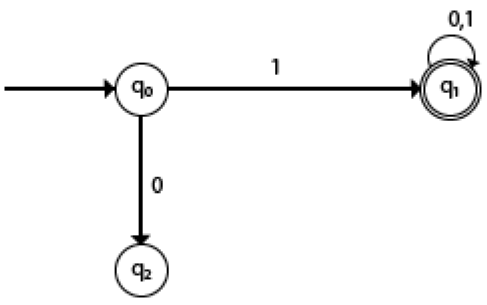


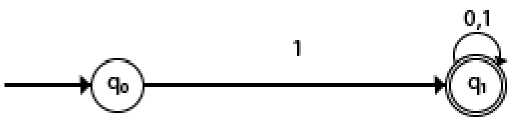
Fig: Transition diagram

The finite automata can be represented using a transition graph. In the above diagram, the machine initially is in start state q0 then on receiving input 1 the machine changes its state to q1. From q0 on receiving 0, the machine changes its state to q2, which is the dead state. From q1 on receiving input 0, 1 the machine changes its state to q1, 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 ∑ = {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 q0 then on receiving input 1 the machine changes its state to q1. From q1 on receiving input 0, 1 the machine changes its state to q1. The possible input string that can be generated is 10, 11, 110, 101, 111......, that means all string starts with 1.

Transition Table

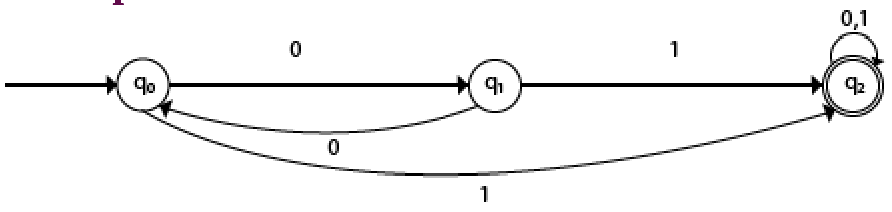
The transition table is basically a tabular representation of the transition function. It takes two arguments (a state and a symbol) and returns a state (the "next state").

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:

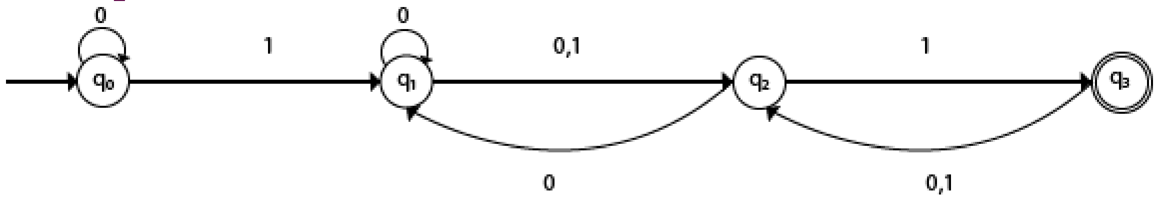
Transition table of given DFA is as follows:

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 q0, on input 0 the next state will be q1 and on input 1 the next state will be q2.
- In the second row, when the current state is q1, on input 0, the next state will be q0, and on 1 input the next state will be q2.
- In the third row, when the current state is q2 on input 0, the next state will be q2, and on 1 input the next state will be q2.
- The arrow marked to q0 indicates that it is a start state and circle marked to q2 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
→q0	q0	q1
q1	q1, q2	q2
q2	q1	q3
*q3	q2	q2

Explanation:

- The first row of the transition table can be read as, when the current state is q0, on input 0 the next state will be q0 and on input 1 the next state will be q1.
- In the second row, when the current state is q1, on input 0 the next state will be either q1 or q2, and on 1 input the next state will be q2.
- In the third row, when the current state is q2 on input 0, the next state will be q1, and on 1 input the next state will be q3.
- In the fourth row, when the current state is q3 on input 0, the next state will be q2, and on 1 input the next state will be q2.

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 q0 for input a, there is only one path which is going to q1. Similarly, from q0, there is only one path for input b going to q2.

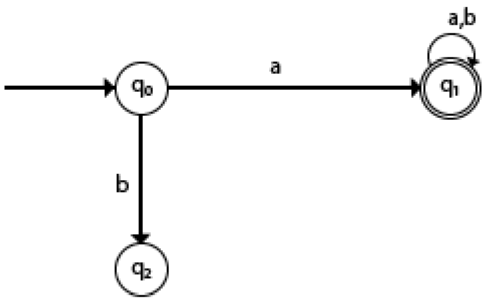


Fig:- DFA

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. ∑: finite set of the input symbol
3. q0: initial state
4. F: **final** state
5. δ: Transition function

Transition function can be defined as:

1. δ: Q x ∑→Q

Graphical Representation of DFA

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

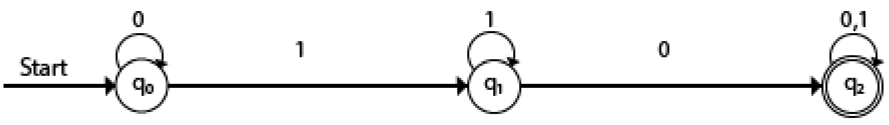
1. The state is represented by vertices.
2. The arc labeled with an input character show the transitions.
3. The initial state is marked with an arrow.
4. The final state is denoted by a double circle.

Example 1:

1. Q = {q0, q1, q2}
2. ∑ = {0, 1}
3. q0 = {q0}
4. F = {q2}

Solution:

Transition Diagram:



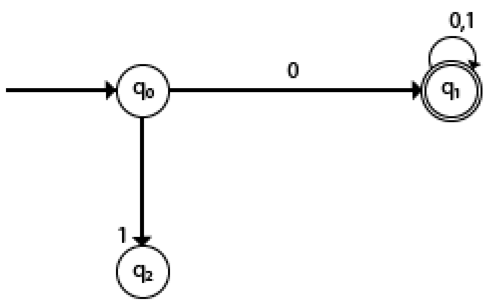
Transition Table:

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 ∑ = {0, 1} accepts all starting with 0.

Solution:



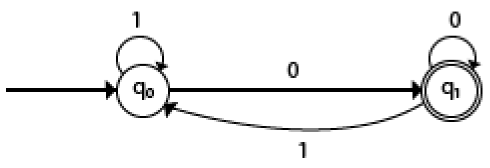
Explanation:

- In the above diagram, we can see that on given 0 as input to DFA in state q0 the DFA changes state to q1 and always go to final state q1 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 ∑ = {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 q0, the DFA changes state to q1. 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 q1 on 1 input, so the string ending with 1, will not be accepted or will be rejected.

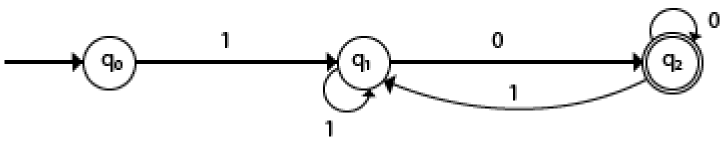
Examples of DFA

Example 1:

Design a FA with ∑ = {0, 1} accepts those string which starts with 1 and ends with 0.

Solution:

The FA will have a start state q0 from which only the edge with input 1 will go to the next state.



In state q1, if we read 1, we will be in state q1, but if we read 0 at state q1, we will reach to state q2 which is the final state. In state q2, if we read either 0 or 1, we will go to q2 state or q1 state respectively. Note that if the input ends with 0, it will be in the final state.

Example 2:

Design a FA with ∑ = {0, 1} accepts the only input 101.

Solution:

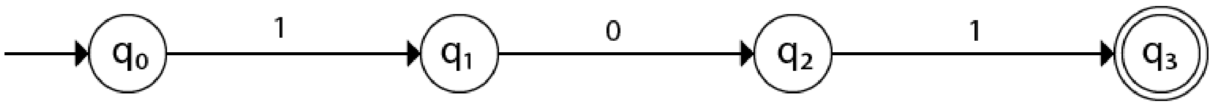


Fig: FA

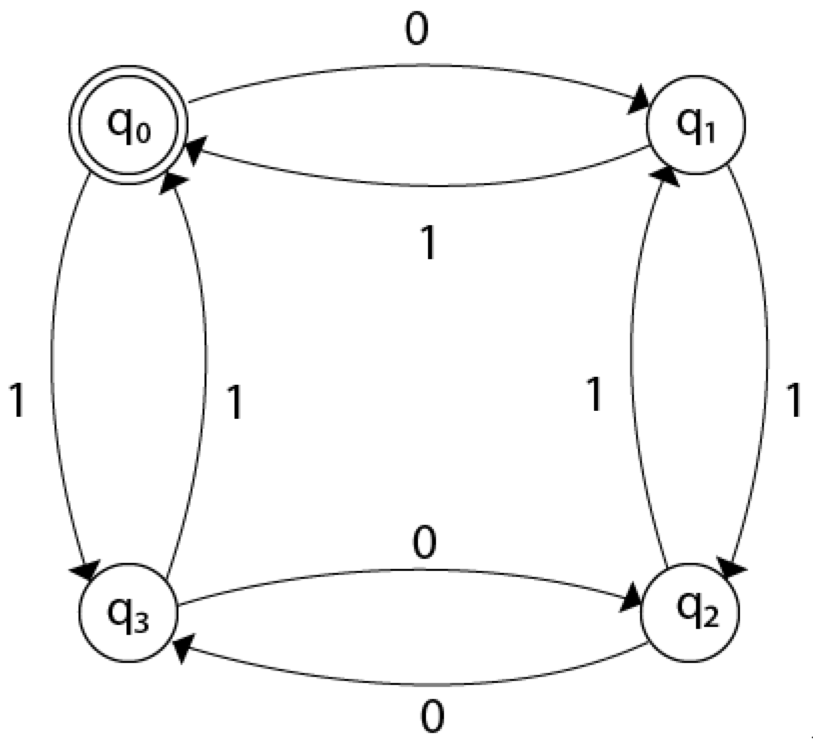
In the given solution, we can see that only input 101 will be accepted. Hence, for input 101, there is no other path shown for other input.

Example 3:

Design FA with ∑ = {0, 1} accepts even number of 0's and even number of 1's.

Solution:

This FA will consider four different stages for input 0 and input 1. The stages could be:



<<

Here q0 is a start state and the final state also. Note carefully that a symmetry of 0's and 1's is maintained. We can associate meanings to each state as:

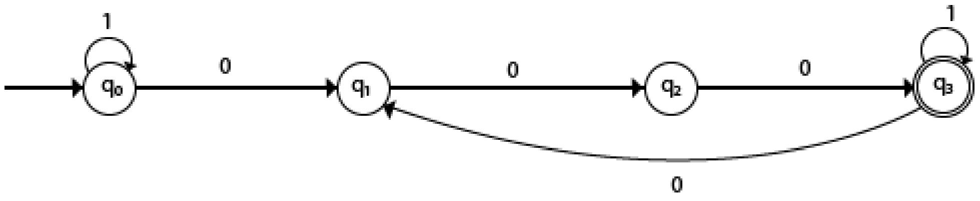
q0: state of even number of 0's and even number of 1's.
q1: state of odd number of 0's and even number of 1's.
q2: state of odd number of 0's and odd number of 1's.
q3: state of even number of 0's and odd number of 1's.

Example 4:

Design FA with ∑ = {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:



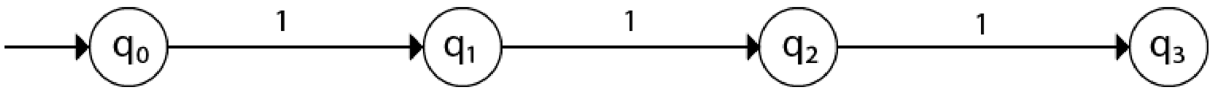
Note that the sequence of triple zeros is maintained to reach the final state.

Example 5:

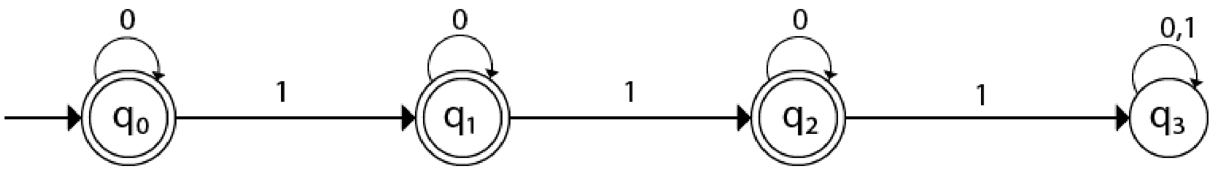
Design a DFA L(M) = {w | w ε {0, 1}*} and W is a string that does not contain consecutive 1's.

Solution:

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



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



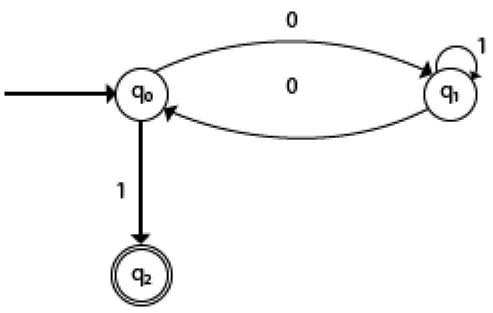
The stages q0, q1, q2 are the final states. The DFA will generate the strings that do not contain consecutive 1's like 10, 110, 101,..... etc.

Example 6:

Design a FA with ∑ = {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 q0 for input a, there are two next states q1 and q2, similarly, from q0 for input b, the next states are q0 and q1. 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.

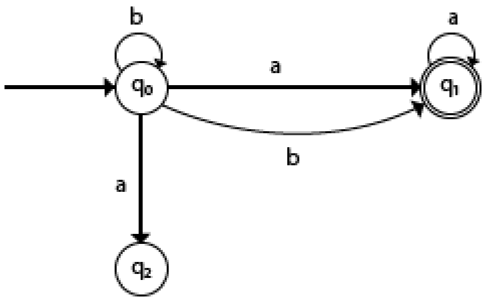


Fig:- NDFA

Formal definition of NFA:

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

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

where,

1. Q: finite set of states
2. ∑: finite set of the input symbol
3. q0: 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 = {q0, q1, q2}
2. ∑ = {0, 1}
3. q0 = {q0}
4. F = {q2}

Solution:

Transition diagram:

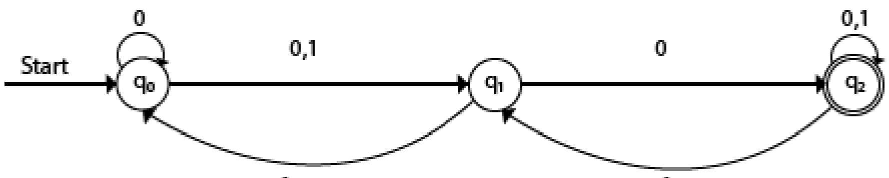


Fig: NFA

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 ∑ = {0, 1} accepts all strings with 01.

Solution:

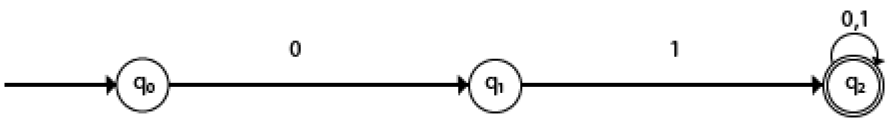


Fig: NFA

Transition Table:

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

Example 3:

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

Solution:

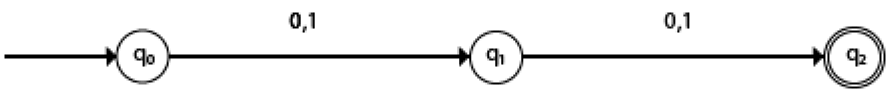


Fig: NFA

Transition Table:

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

Examples of NFA

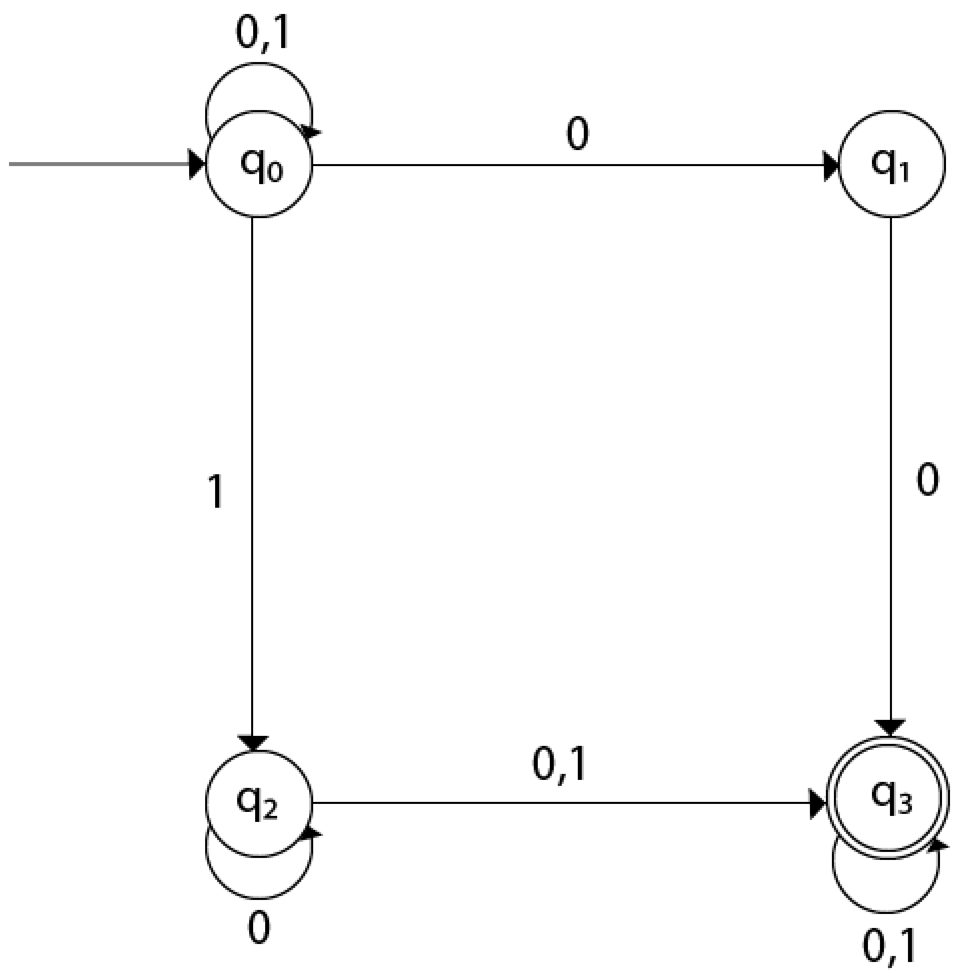
Example 1:

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
*q3	q3	q3

Solution:

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



Here,

1. δ(q0, 0) = {q0, q1}
2. δ(q0, 1) = {q0, q2}
3. Then, δ(q1, 0) = {q3}
4. Then, δ(q2, 0) = {q2, q3}
5. δ(q2, 1) = {q3}
6. Then, δ(q3, 0) = {q3}
7. δ(q3, 1) = {q3}

Example 2:

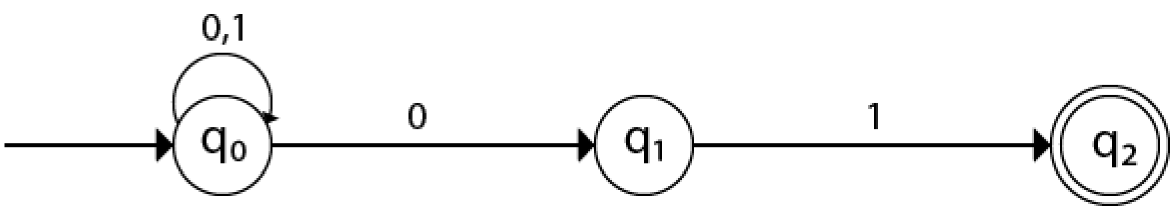
Design an NFA with ∑ = {0, 1} accepts all string ending with 01.

Solution:

Anything either 0 or 1

0 1

Hence, NFA would be:

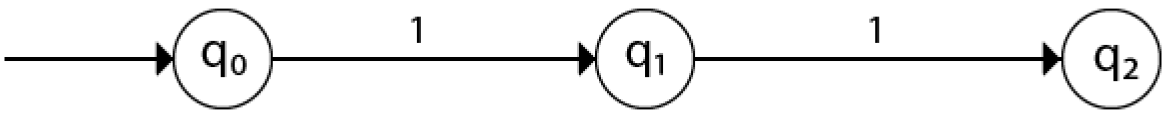


Example 3:

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

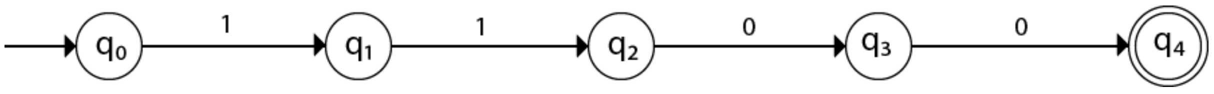
Solution:

The FA with double 1 is as follows:



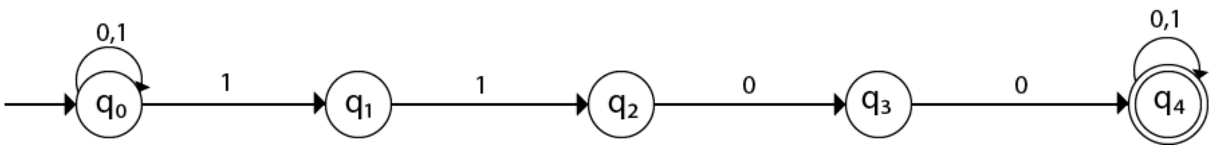
It should be immediately followed by double 0.

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

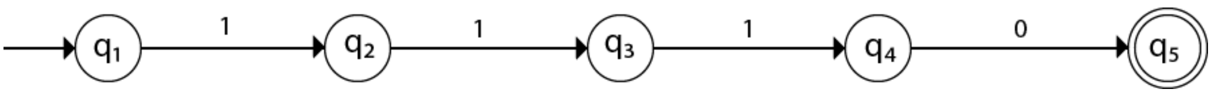
1. q0 → q1 → q2 → q3 → q4 → q4 → q4 → q4

Example 4:

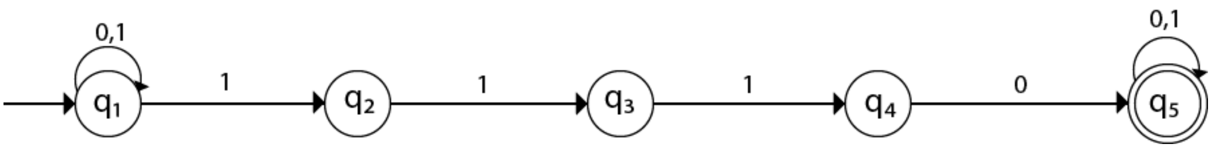
Design an NFA in which all the string contain a substring 1110.

Solution:

The language consists of all the string containing substring 1010. The partial transition diagram can be 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. δ(q1, 111010) = δ(q1, 1100)
2. = δ(q1, 100)

3. = δ(q2, 00)

Got stuck! As there is no path from q2 for input symbol 0. We can process string 111010 in another way.

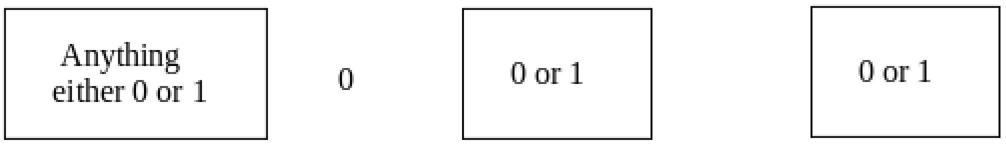
1. δ(q1, 111010) = δ(q2, 1100)
2. = δ(q3, 100)
3. = δ(q4, 00)
4. = δ(q5, 0)
5. = δ(q5, ε)

As state q5 is the accept state. We get the complete scanned, and we reached to the final state.

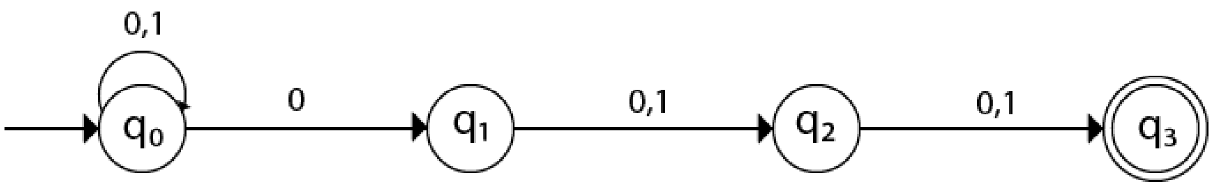
Example 5:

Design an NFA with ∑ = {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 q0 with input 0, we can either go to state q0 or q1.

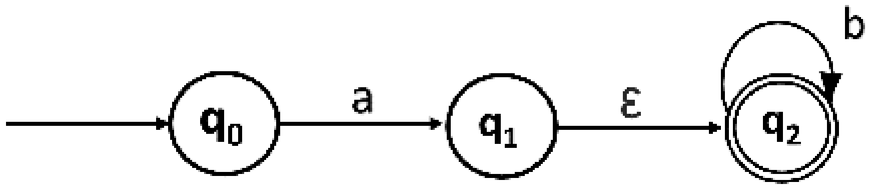
Eliminating ε Transitions

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{q1} where qi ∈ 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 q0, q1 and q2 as follows:

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

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

1. δ'(q0, a) = ε-closure(δ(δ^(q0, ε),a))
2. = ε-closure(δ(ε-closure(q0),a))
3. = ε-closure(δ(q0, a))
4. = ε-closure(q1)
5. = {q1, q2}
6.
7. δ'(q0, b) = ε-closure(δ(δ^(q0, ε),b))
8. = ε-closure(δ(ε-closure(q0),b))
9. = ε-closure(δ(q0, b))
10. = Φ

Now the δ' transition on q1 is obtained as:

1. δ'(q1, a) = ε-closure(δ(δ^(q1, ε),a))
2. = ε-closure(δ(ε-closure(q1),a))
3. = ε-closure(δ(q1, q2), a)

4. = ε-closure(δ(q1, a) ∪ δ(q2, a))
5. = ε-closure(Φ ∪ Φ)
6. = Φ
7.
8. δ'(q1, b) = ε-closure(δ(δ^(q1, ε),b))
9. = ε-closure(δ(ε-closure(q1),b))
10. = ε-closure(δ(q1, q2), b)
11. = ε-closure(δ(q1, b) ∪ δ(q2, b))
12. = ε-closure(Φ ∪ q2)
13. = {q2}

The δ' transition on q2 is obtained as:

1. δ'(q2, a) = ε-closure(δ(δ^(q2, ε),a))
2. = ε-closure(δ(ε-closure(q2),a))
3. = ε-closure(δ(q2, a))
4. = ε-closure(Φ)
5. = Φ
6.
7. δ'(q2, b) = ε-closure(δ(δ^(q2, ε),b))
8. = ε-closure(δ(ε-closure(q2),b))
9. = ε-closure(δ(q2, b))
10. = ε-closure(q2)
11. = {q2}

Now we will summarize all the computed δ' transitions:

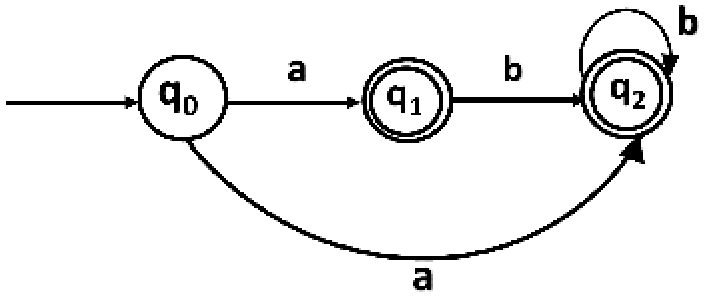
1. δ'(q0, a) = {q0, q1}
2. δ'(q0, b) = Φ
3. δ'(q1, a) = Φ
4. δ'(q1, b) = {q2}
5. δ'(q2, a) = Φ
6. δ'(q2, b) = {q2}

The transition table can be:

States	**a**	**b**

→q0	{q1, q2}	Φ
*q1	Φ	{q2}
*q2	Φ	{q2}

State q1 and q2 become the final state as ε-closure of q1 and q2 contain the final state q2. 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, ∑, δ, q0, F) is an NFA which accepts the language L(M). There should be equivalent DFA denoted by M' = (Q', ∑', q0', δ', F') such that L(M) = L(M').

Steps for converting NFA to DFA:

Step 1: Initially Q' = ϕ

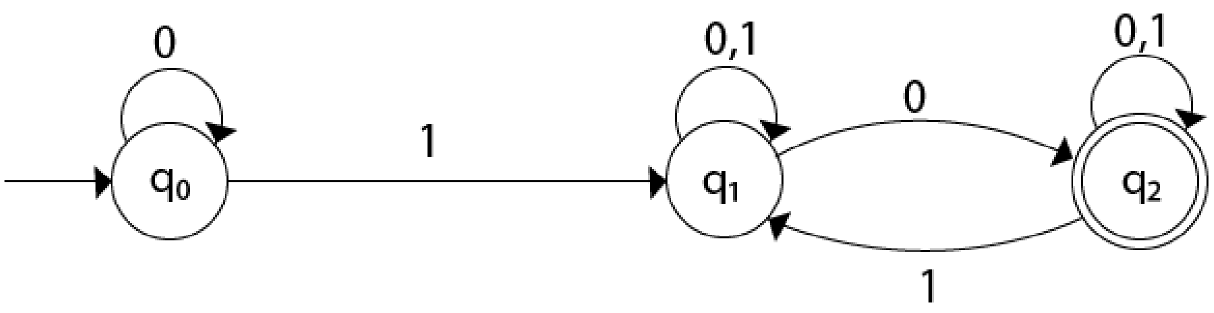
Step 2: Add q0 of NFA to Q'. Then find the transitions from this start state.

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. δ'([q0], 0) = [q0]
2. δ'([q0], 1) = [q1]

The δ' transition for state q1 is obtained as:

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

The δ' transition for state q2 is obtained as:

1. δ'([q2], 0) = [q2]
2. δ'([q2], 1) = [q1, q2]

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

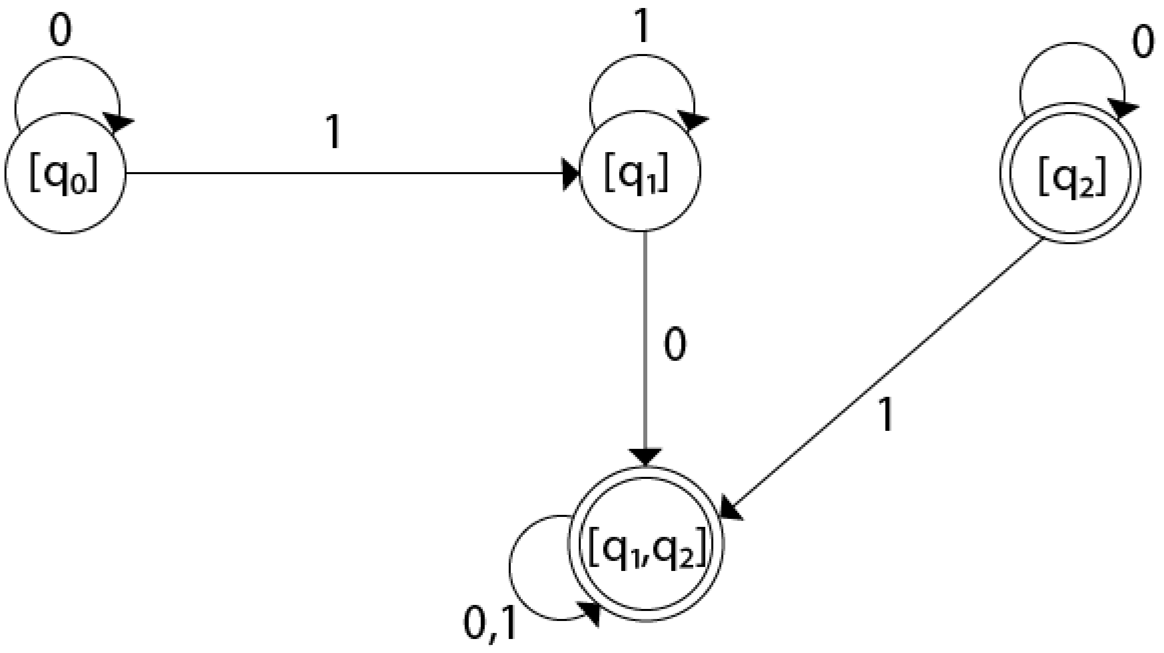
1. δ'([q1, q2], 0) = δ(q1, 0) ∪ δ(q2, 0)
2.   = {q1, q2} ∪ {q2}
3.   = [q1, q2]
4. δ'([q1, q2], 1) = δ(q1, 1) ∪ δ(q2, 1)

5. = {q1} ∪ {q1, q2}
6. = {q1, q2}
7. = [q1, q2]

The state [q1, q2] is the final state as well because it contains a final state q2. The transition table for the constructed DFA will be:

State	0	1
→[q0]	[q0]	[q1]
[q1]	[q1, q2]	[q1]
*[q2]	[q2]	[q1, q2]
*[q1, q2]	[q1, q2]	[q1, q2]

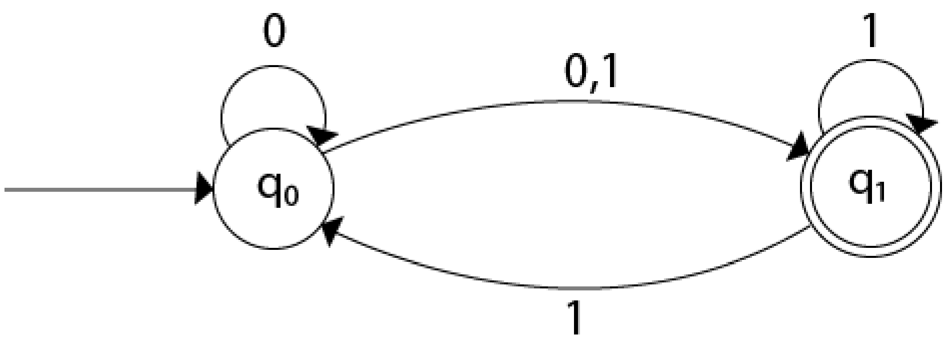
The Transition diagram will be:



The state q2 can be eliminated because q2 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**
→q0	{q0, q1}	{q1}
*q1	ϕ	{q0, q1}

Now we will obtain δ' transition for state q0.

1. δ'([q0], 0) = {q0, q1}
2. = [q0, q1] (**new** state generated)
3. δ'([q0], 1) = {q1} = [q1]

The δ' transition for state q1 is obtained as:

1. δ'([q1], 0) = ϕ
2. δ'([q1], 1) = [q0, q1]

Now we will obtain δ' transition on [q0, q1].

1. δ'([q0, q1], 0) = δ(q0, 0) ∪ δ(q1, 0)
2. = {q0, q1} ∪ ϕ
3. = {q0, q1}
4. = [q0, q1]

Similarly,

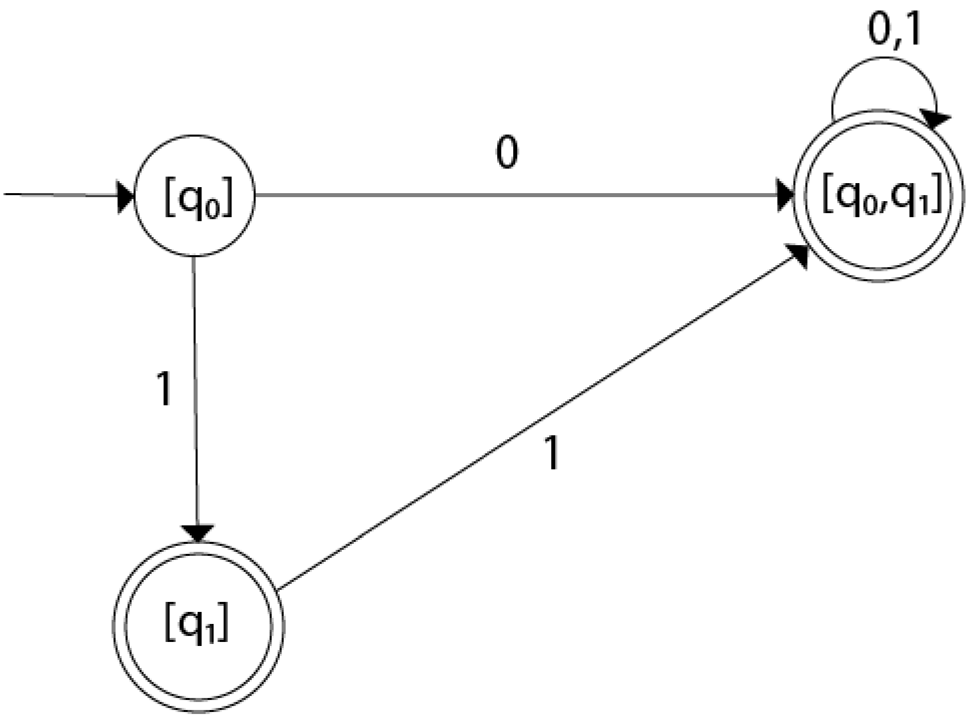
1. δ'([q0, q1], 1) = δ(q0, 1) ∪ δ(q1, 1)
2. = {q1} ∪ {q0, q1}
3. = {q0, q1}
4. = [q0, q1]

As in the given NFA, q1 is a final state, then in DFA wherever, q1 exists that state becomes a final state. Hence in the DFA, final states are [q1] and [q0, q1]. Therefore set of final states F = {[q1], [q0, q1]}.

The transition table for the constructed DFA will be:

State	**0**	**1**
→[q0]	[q0, q1]	[q1]
*[q1]	ϕ	[q0, q1]
*[q0, q1]	[q0, q1]	[q0, q1]

The Transition diagram will be:

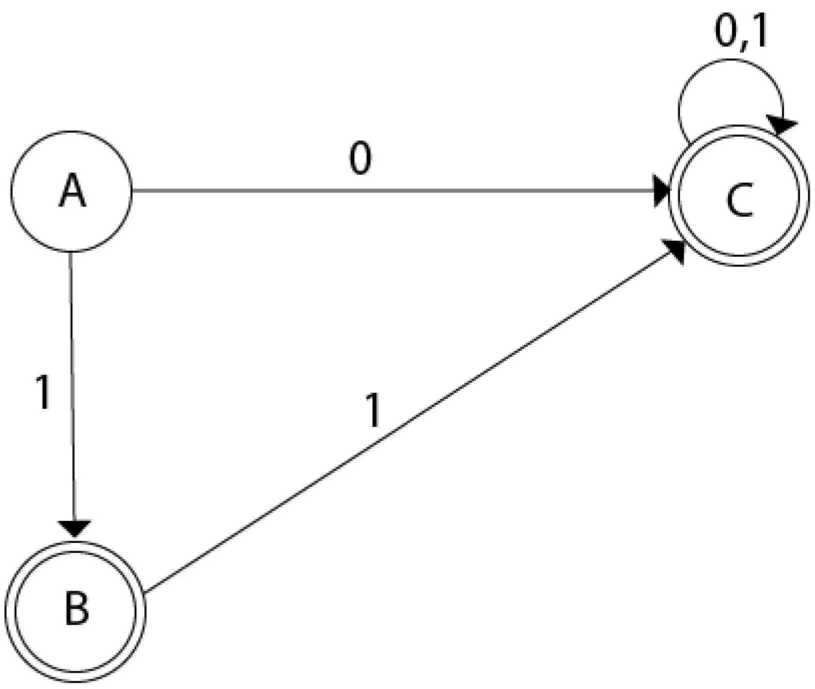


Even we can change the name of the states of DFA.

Suppose

1. A = [q0]
2. B = [q1]
3. C = [q0, q1]

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, ∑, δ, q0, F}.

Where

1. Q: finite set of states
2. ∑: finite set of the input symbol
3. q0: initial state
4. F: **final** state
5. δ: Transition function

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

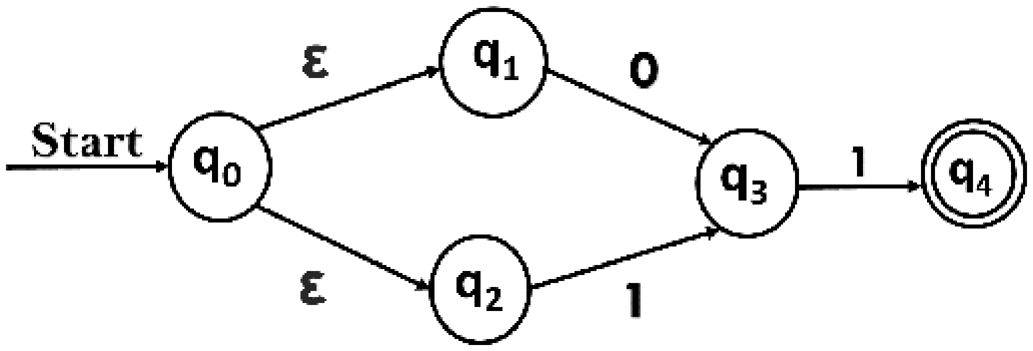
Step 3: If we found a new state, take it as current state and repeat step 2.

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

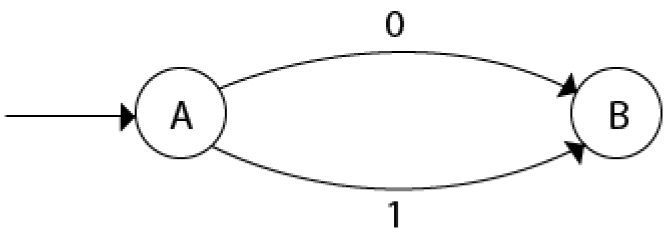
Now, let ε-closure {q0} = {q0, q1, q2} be state A.

Hence

δ'(A, 0) = ε-closure {δ((q0, q1, q2), 0) }
= ε-closure {δ(q0, 0) ∪ δ(q1, 0) ∪ δ(q2, 0) }
= ε-closure {q3}
= {q3} **call it as state B**.

δ'(A, 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



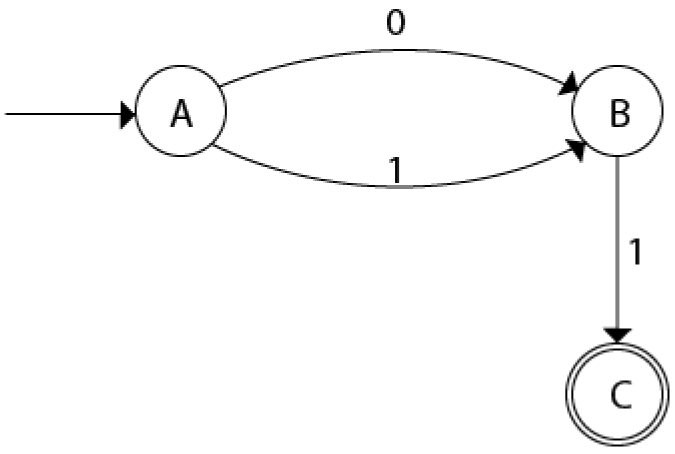
Now,

δ'(B, 0) = ε-closure {δ(q3, 0) }
= ϕ
δ'(B, 1) = ε-closure {δ(q3, 1) }
= ε-closure {q4}
= {q4} **i.e. state C**

For state C:

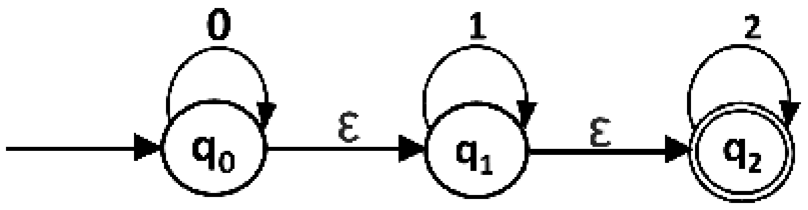
1. δ'(C, 0) = ε-closure {δ(q4, 0) }
2. = ϕ
3. δ'(C, 1) = ε-closure {δ(q4, 1) }
4. = ϕ

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} call 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}

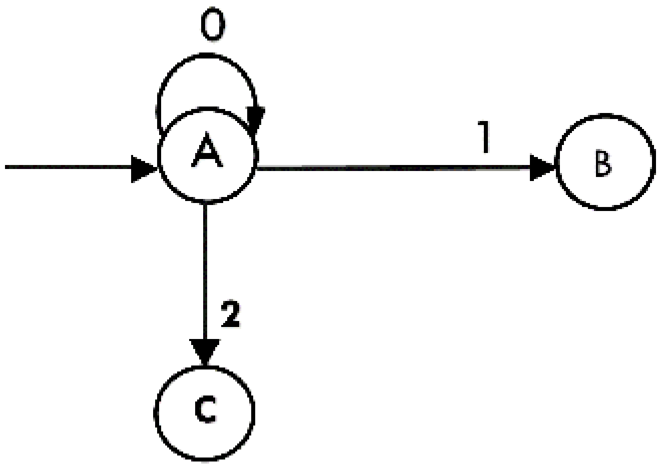
δ'(A, 1) = ε-closure{δ((q0, q1, q2), 1)}
= ε-closure{δ(q0, 1) ∪ δ(q1, 1) ∪ δ(q2, 1)}
= ε-closure{q1}
= {q1, q2} **call it as state B**

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

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

δ'(B, 1) = ε-closure{δ((q1, q2), 1)}
    = ε-closure{δ(q1, 1) ∪ δ(q2, 1)}
    = ε-closure{q1}
    = {q1, q2}   **i.e. state B itself**

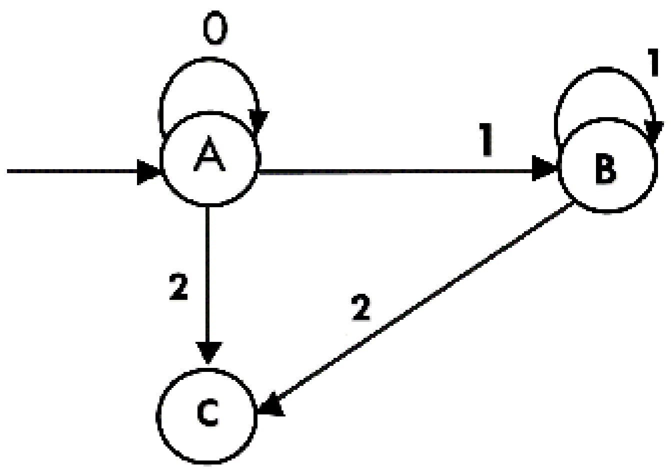
δ'(B, 2) = ε-closure{δ((q1, q2), 2)}
    = ε-closure{δ(q1, 2) ∪ δ(q2, 2)}
    = ε-closure{q2}
    = {q2}   **i.e. state C itself**

Thus we have obtained

1. δ'(B, 0) = ϕ
2. δ'(B, 1) = B

3. δ'(B, 2) = C

The partial transition diagram will be



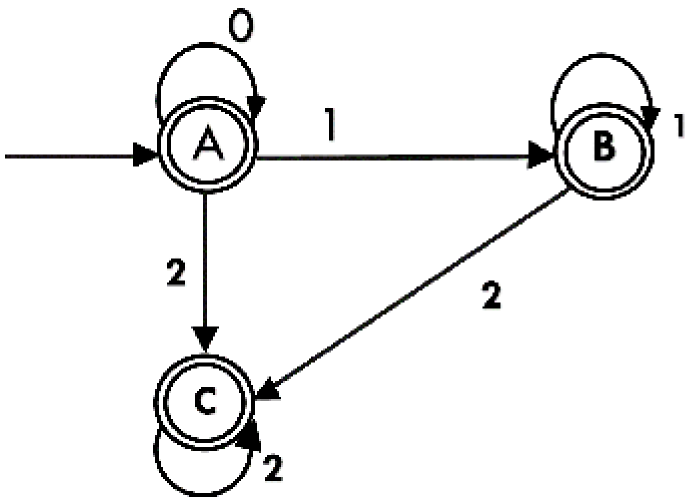
Now we will obtain transitions for C:

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.

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. 1. δ (q, a) = p
2. 2. δ (r, a) = p

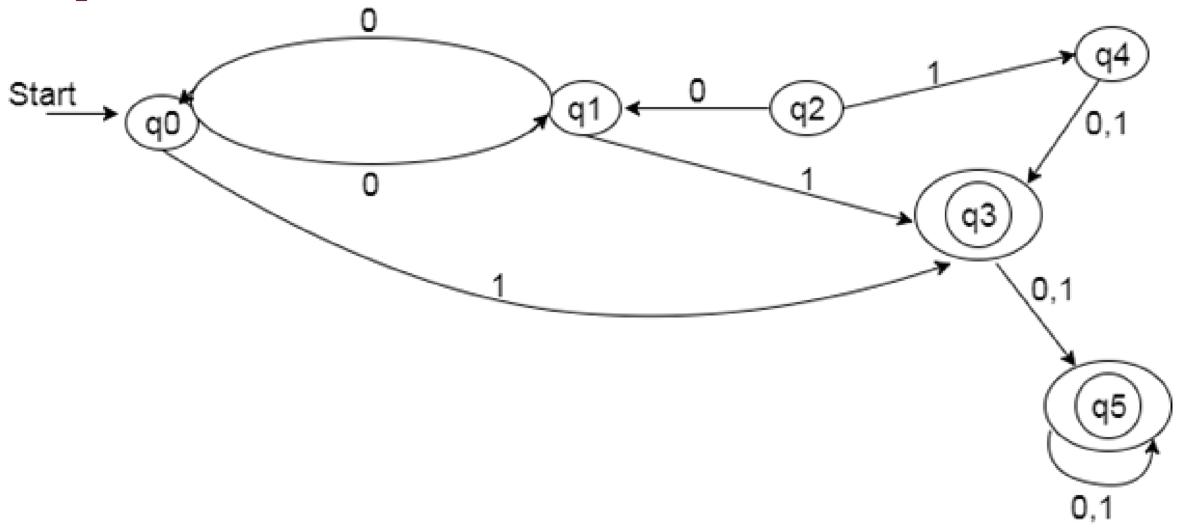
That means, find the two states which have the same value of a and b and remove one of them.

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.

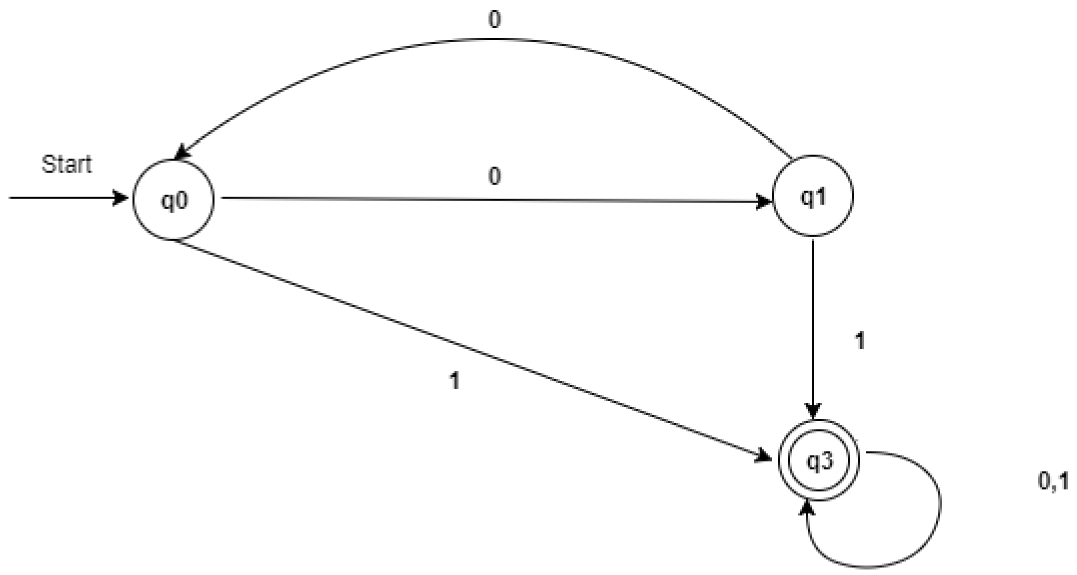
State	0	1
q3	q3	q3

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

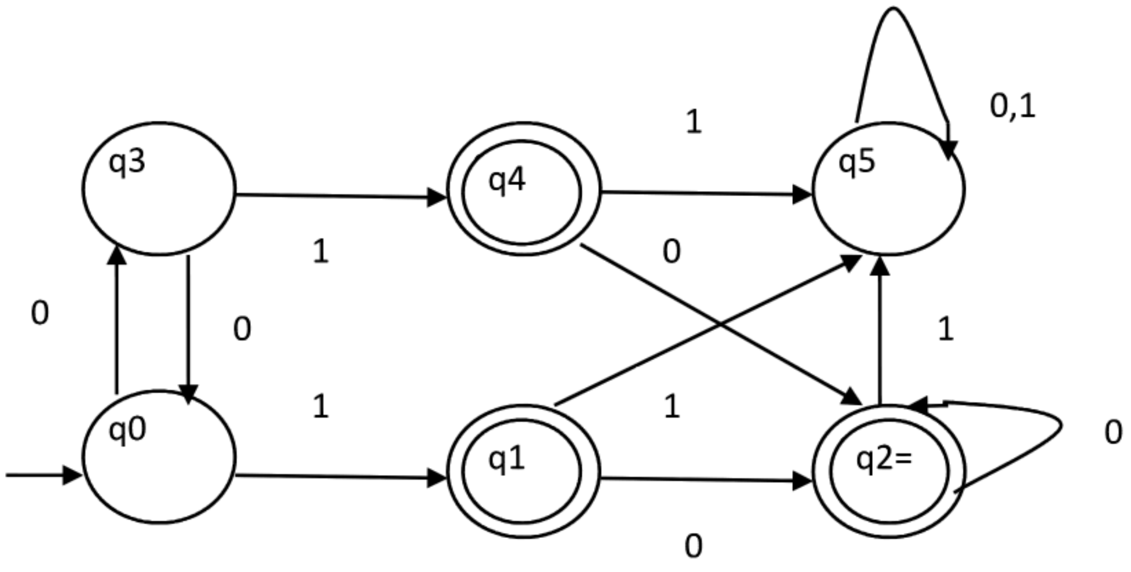
State	0	1

→q0	q1	q3
q1	q0	q3
*q3	q3	q3

Now it is the transition table of minimized DFA.



Example
Consider the following DFA shown in figure.



Step 1. P0 will have two sets of states. One set will contain q1, q2, q4 which are final states of DFA and another set will contain remaining states. So P0 = { { q1, q2, q4 }, { q0, q3, q5 } }.
Step 2. To calculate P1, we will check whether sets of partition P0 can be partitioned or not:

i) For set { q1, q2, q4 } :
δ (q1, 0) = δ (q2, 0) = q2 and δ (q1, 1) = δ (q2, 1) = q5, So q1 and q2 are not distinguishable.
Similarly, δ (q1, 0) = δ (q4, 0) = q2 and δ (q1, 1) = δ (q4, 1) = q5, So q1 and q4 are not distinguishable.
Since, q1 and q2 are not distinguishable and q1 and q4 are also not distinguishable, So q2 and q4 are not distinguishable. So, { q1, q2, q4 } set will not be partitioned in P1.

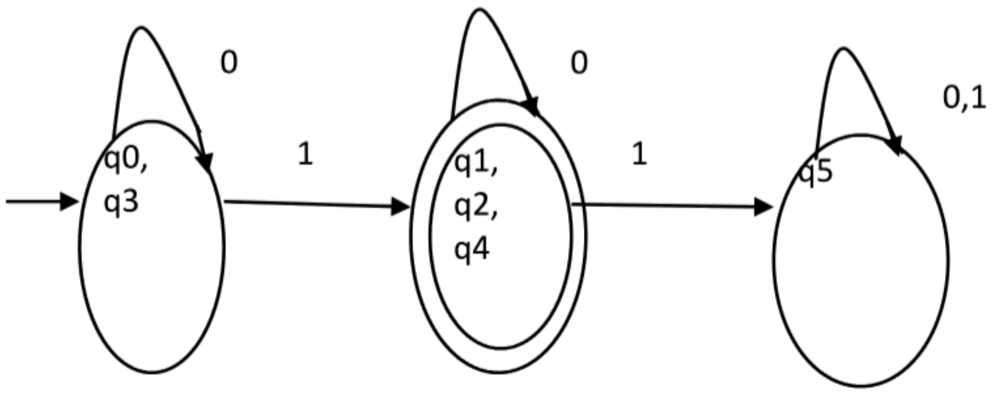
ii) For set { q0, q3, q5 } :
δ (q0, 0) = q3 and δ (q3, 0) = q0
δ (q0, 1) = q1 and δ(q3, 1) = q4
Moves of q0 and q3 on input symbol 0 are q3 and q0 respectively which are in same set in partition P0.
Similarly, Moves of q0 and q3 on input symbol 1 are q1 and q4 which are in same set in partition P0. So, q0 and q3 are not distinguishable.

δ (q0, 0) = q3 and δ (q5, 0) = q5 and δ (q0, 1) = q1 and δ (q5, 1) = q5
Moves of q0 and q5 on input symbol 1 are q1 and q5 respectively which are in different set in partition P0.
So, q0 and q5 are distinguishable. So, set { q0, q3, q5 } will be partitioned into { q0, q3 } and { q5 }. So,
P1 = { { q1, q2, q4 }, { q0, q3}, { q5 } }

To calculate P2, we will check whether sets of partition P1 can be partitioned or not:
iii)For set { q1, q2, q4 } :
δ (q1, 0) = δ (q2, 0) = q2 and δ (q1, 1) = δ (q2, 1) = q5, So q1 and q2 are not distinguishable.
Similarly, δ (q1, 0) = δ (q4, 0) = q2 and δ (q1, 1) = δ (q4, 1) = q5, So q1 and q4 are not distinguishable.
Since, q1 and q2 are not distinguishable and q1 and q4 are also not distinguishable, So q2 and q4 are not distinguishable. So, { q1, q2, q4 } set will not be partitioned in P2.

iv)For set { q0, q3 } :
δ (q0, 0) = q3 and δ (q3, 0) = q0
δ (q0, 1) = q1 and δ (q3, 1) = q4
Moves of q0 and q3 on input symbol 0 are q3 and q0 respectively which are in same set in partition P1.
Similarly, Moves of q0 and q3 on input symbol 1 are q1 and q4 which are in same set in partition P1. So, q0 and q3 are not distinguishable.

v) For set { q5 }:
Since we have only one state in this set, it can’t be further partitioned. So,
P2 = { { q1, q2, q4 }, { q0, q3 }, { q5 } }
Since, P1=P2. So, this is the final partition. Partition P2 means that q1, q2 and q4 states are merged into one.
Similarly, q0 and q3 are merged into one. Minimized DFA corresponding to DFA of Figure 1 is shown in Figure 2 as:



Moore Machines are finite state machines with output value and its output depends only on the present state. It can be defined as (Q, q0, ∑, O, δ, λ) where:

- Q is a finite set of states.
- q0 is the initial state.
- ∑ is the input alphabet.
- O is the output alphabet.
- δ is transition function which maps Q×∑ → Q.
- λ is the output function which maps Q → O.

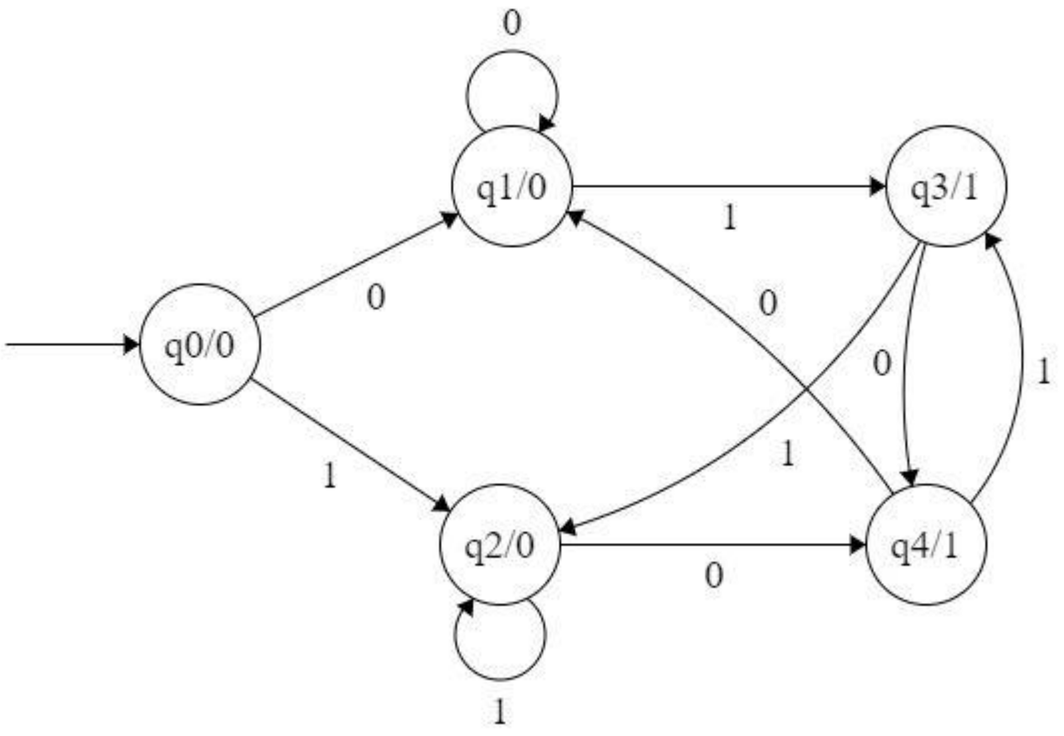


Figure 1

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

- **Input:** 11
- **Transition:** δ (q0,11)=> δ(q2,1)=>q2
- **Output:** 000 (0 for q0, 0 for q2 and again 0 for q2)

Mealy Machines: Mealy machines are also finite state machines with output value and its output depends on the present state and current input symbol. It can be defined as (Q, q0, ∑, O, δ, λ') where:

- Q is a finite set of states.
- q0 is the initial state.
- ∑ is the input alphabet.
- O is the output alphabet.
- δ is the transition function which maps Q×∑ → Q.
- 'λ' is the output function that maps Q×∑→ O.

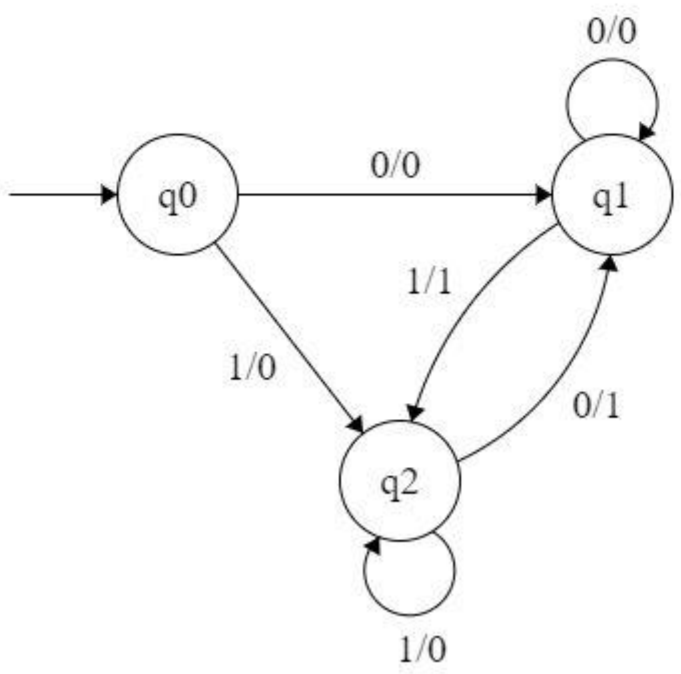


Figure 2

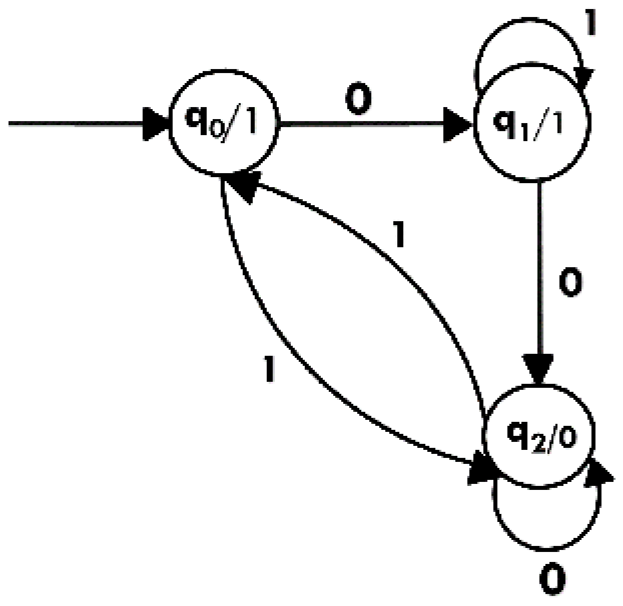
Moore Machine

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, q0, ∑, O, δ, λ) where,

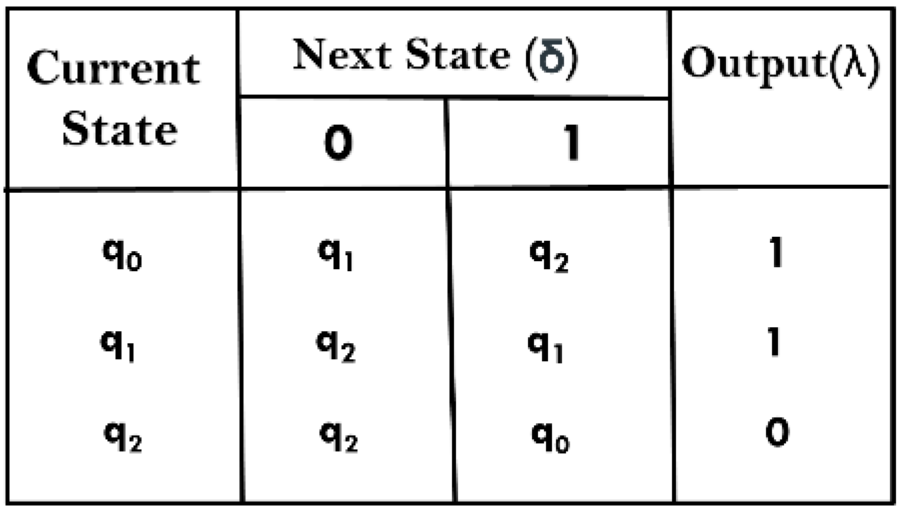
1. Q: finite set of states
2. q0: initial state of machine
3. ∑: finite set of input symbols
4. O: output alphabet
5. δ: transition function where Q × ∑ → Q
6. λ: output function where Q → O

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: δ (q0,0) => δ(q1,1) => δ(q1,0) => q2

Output: 1110(1 for q0, 1 for q1, again 1 for q1, 0 for q2)

Thus Moore machine M = (Q, q0, ∑, O, δ, λ); where Q = {q0, q1, q2}, ∑ = {0, 1}, O = {0, 1}. the transition table shows the δ and λ functions.

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, q0, ∑, O, δ, λ') where

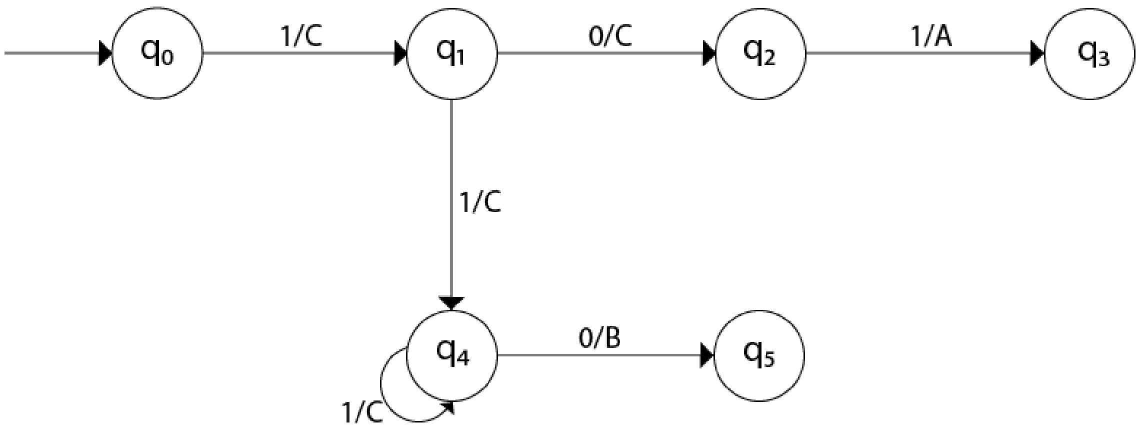
1. Q: finite set of states
2. q0: initial state of machine
3. ∑: finite set of input alphabet
4. O: output alphabet
5. δ: transition function where Q × ∑ → Q
6. λ': output function where Q × ∑ →O

Example 1:

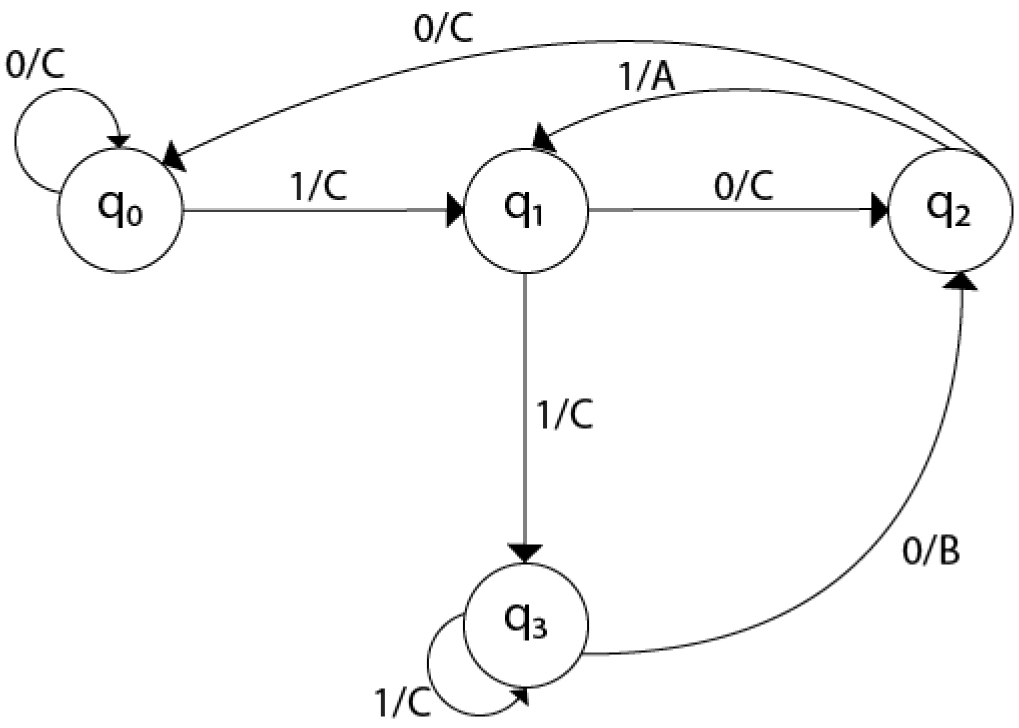
Design a Mealy machine for a binary input sequence such that if it has a substring 101, the machine output A, if the input has substring 110, it outputs B otherwise it outputs C.

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:


$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[r, "1/C"] & q_1 \arrow[r, "0/C"] \arrow[d, "1/C"] & q_2 \arrow[r, "1/A"] & q_3 \\
 & & q_4 \arrow[r, "0/B"] \arrow[loop left, "1/C"] & q_5 &
\end{tikzcd}$$

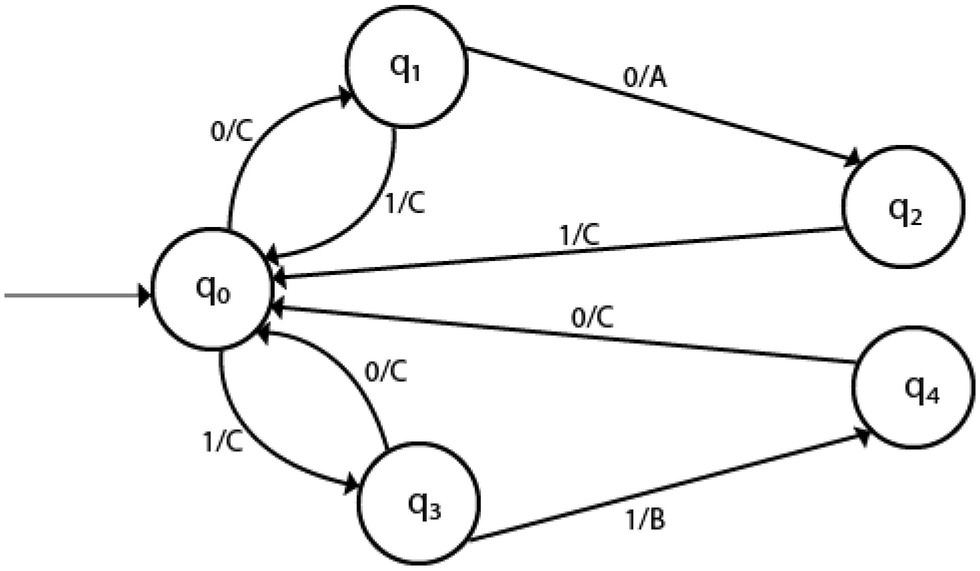
Now we will insert the possibilities of 0's and 1's for each state. Thus the Mealy machine becomes:



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:

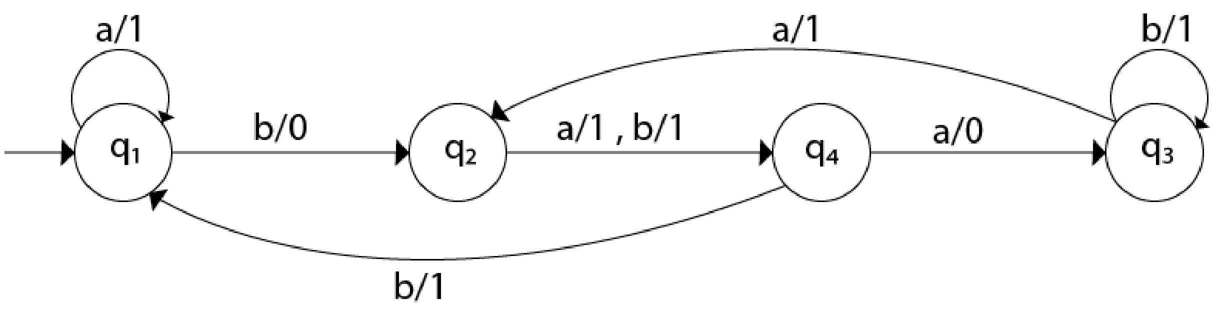
Step 1: For each state(Qi), calculate the number of different outputs that are available in the transition table of the Mealy machine.

Step 2: Copy state Qi, if all the outputs of Qi are the same. Break qi into n states as Qin, if it has n distinct outputs where n = 0, 1, 2....

tep 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:

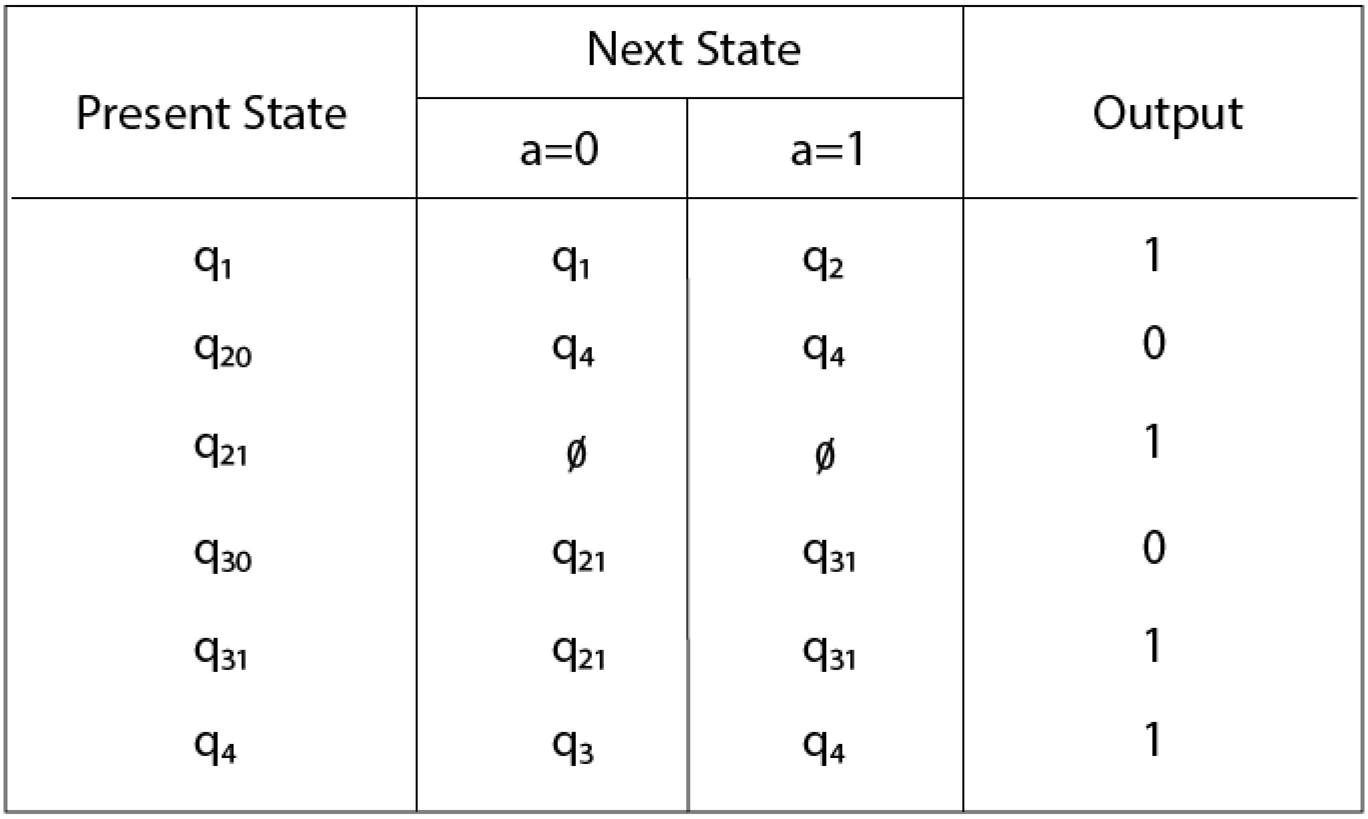
Transition table for above Mealy machine is as follows:

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).

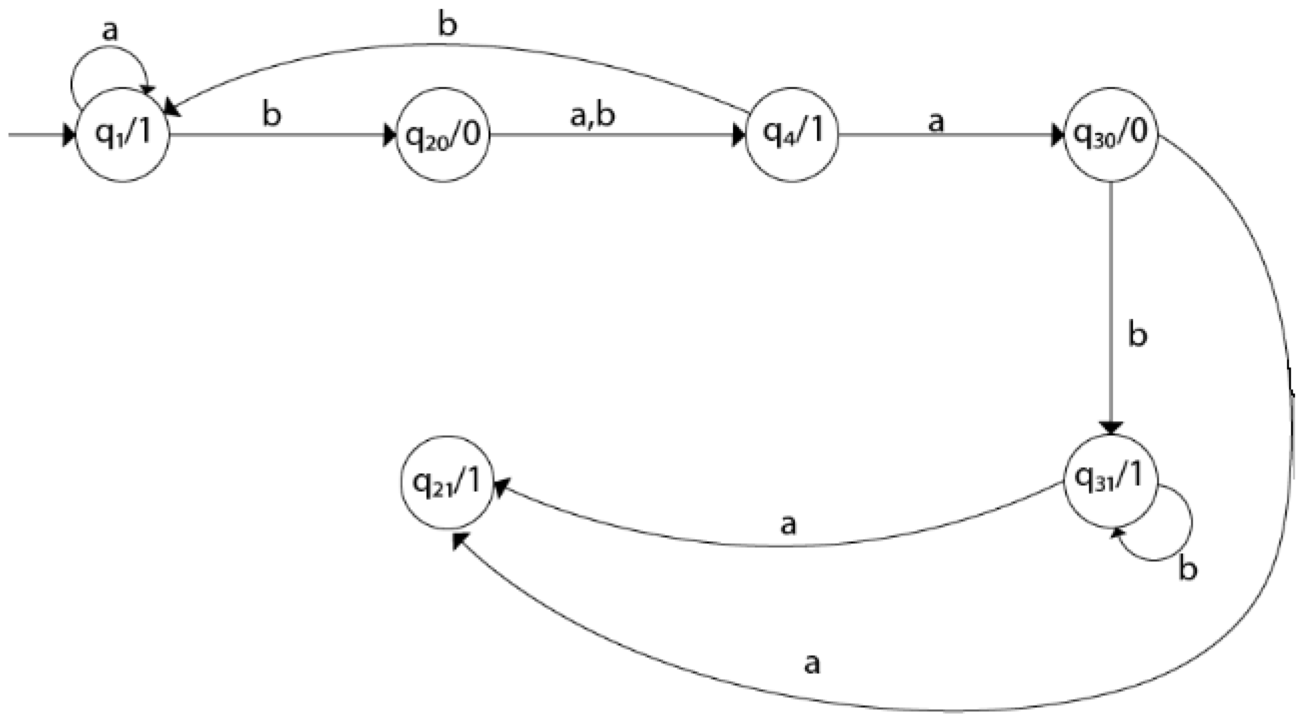
- For state q3, there is 2 incident edge with output 0 and 1. So, we will split this state into two states q30(state with output 0) and q31(state with output 1).
- For state q4, there is only one incident edge with output 0. So, we don't need to split this state in Moore machine.

Transition table for Moore machine will be:



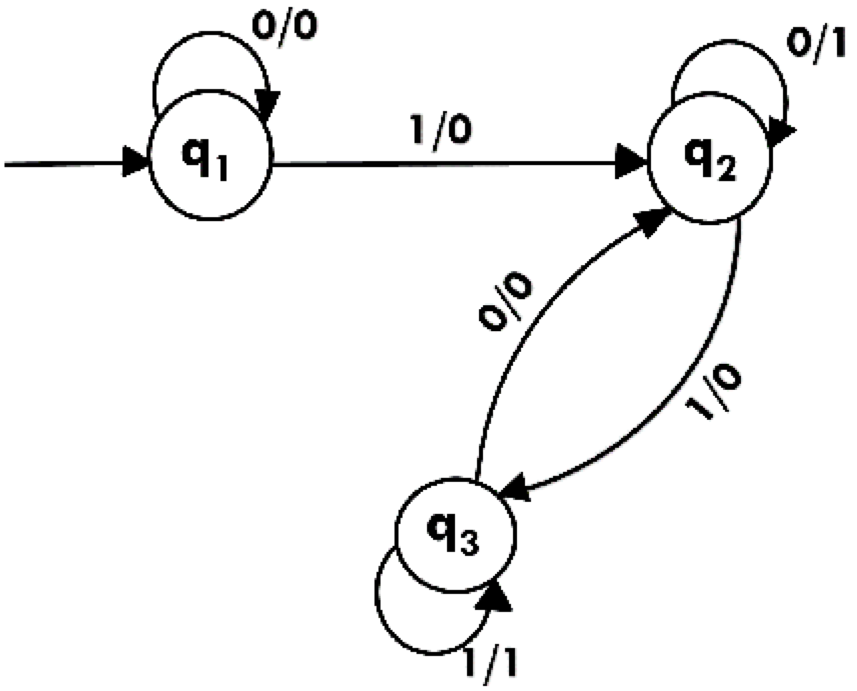
Present State	Next State		Output
	a=0	a=1	
q_1	q_1	q_2	1
q_{20}	q_4	q_4	0
q_{21}	∅	∅	1
q_{30}	q_{21}	q_{31}	0
q_{31}	q_{21}	q_{31}	1
q_4	q_3	q_4	1

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:

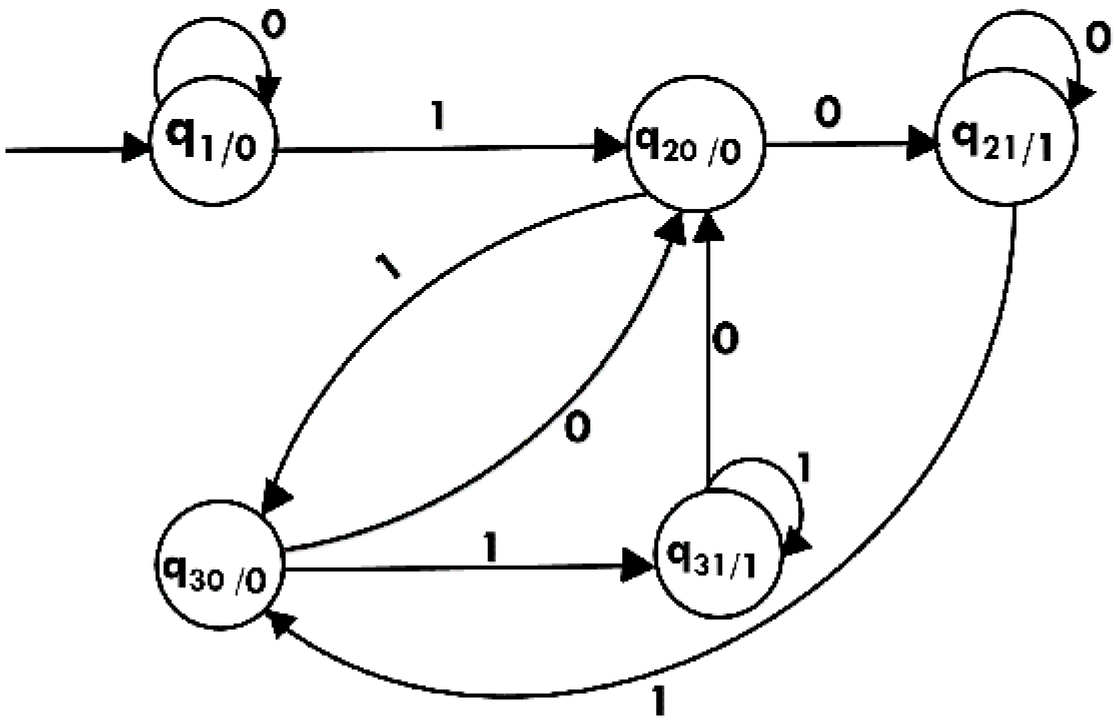
Present State	Next State 0		Next State 1	
	State	O/P	State	O/P
q_1	q_1	0	q_2	0
q_2	q_2	1	q_3	0
q_3	q_2	0	q_3	1

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
q_1	q_1	q_{20}	0
q_{20}	q_{21}	q_{30}	0
q_{21}	q_{21}	q_{30}	1
q_{30}	q_{20}	q_{31}	0
q_{31}	q_{20}	q_{31}	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.

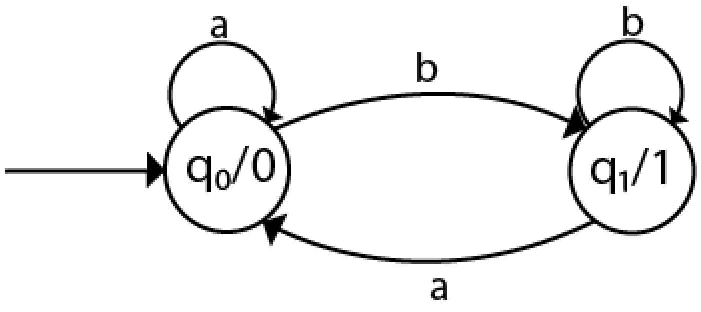
Method for conversion of Moore machine to Mealy machine

Let M = (Q, ∑, δ, λ, q0) be a Moore machine. The equivalent Mealy machine can be represented by M' = (Q, ∑, δ, λ', q0). The output function λ' can be obtained as:

1. λ' (q, a) = λ(δ(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. λ' (q0, a) = λ(δ(q0, a))
2. = λ(q0)
3. = 0
4.
5. λ' (q0, b) = λ(δ(q0, b))
6. = λ(q1)
7. = 1

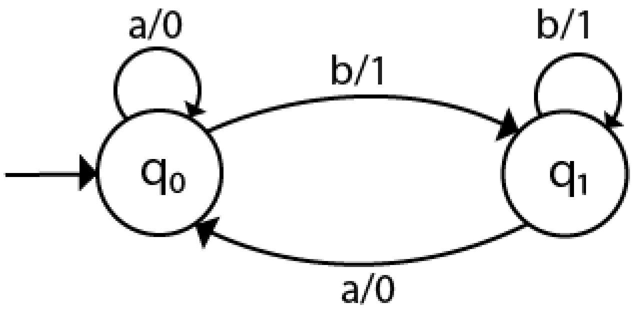
The λ for state q1 is as follows:

1. λ' (q1, a) = λ(δ(q1, a))
2. = λ(q0)
3. = 0
4.
5. λ' (q1, b) = λ(δ(q1, b))
6. = λ(q1)
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

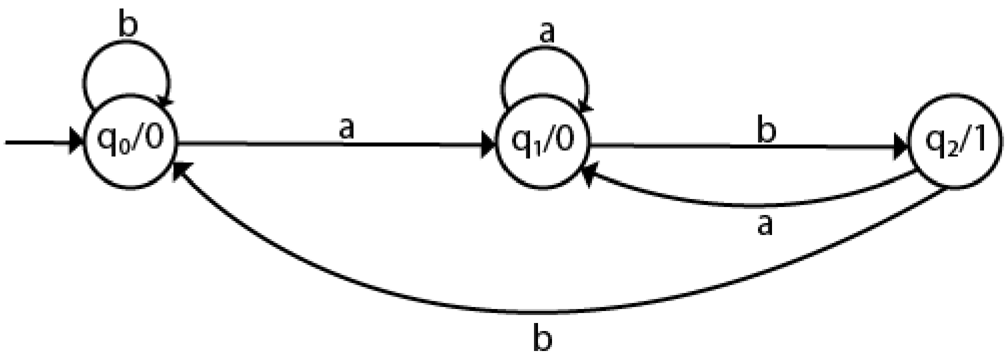
The equivalent Mealy machine will be,



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:

```
1. λ' (q0, a) = λ(δ(q0, a))
2.            = λ(q1)
3.            = 0
4.
5. λ' (q0, b) = λ(δ(q0, b))
6.            = λ(q0)
7.            = 0
```

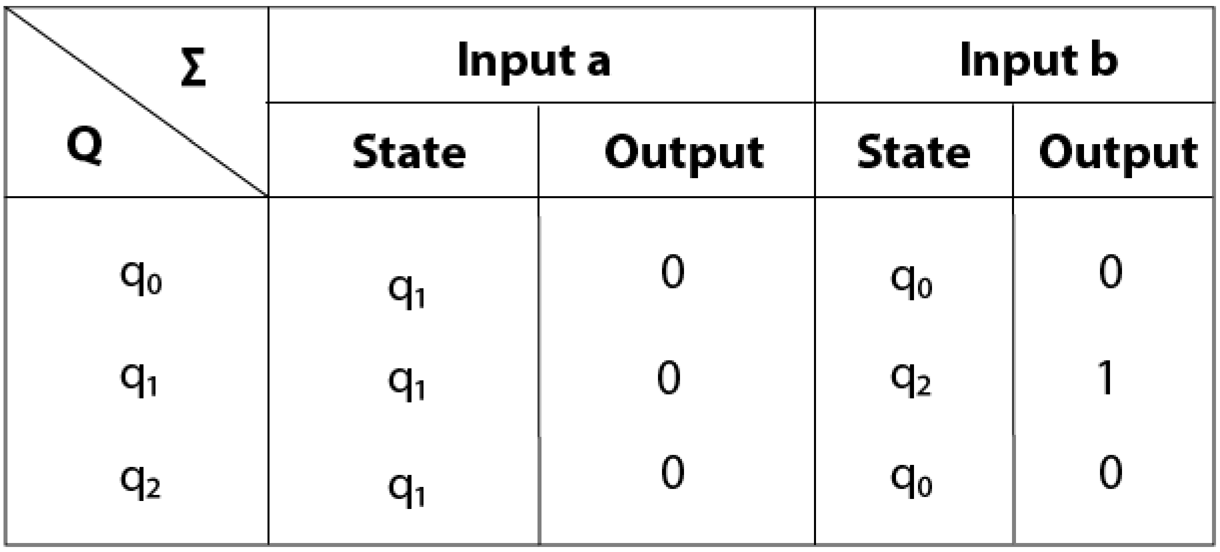
The λ for state q1 is as follows:

```
1. λ' (q1, a) = λ(δ(q1, a))
2.            = λ(q1)
3.            = 0
4.
5. λ' (q1, b) = λ(δ(q1, b))
6.            = λ(q2)
7.            = 1
```

The λ for state q2 is as follows:

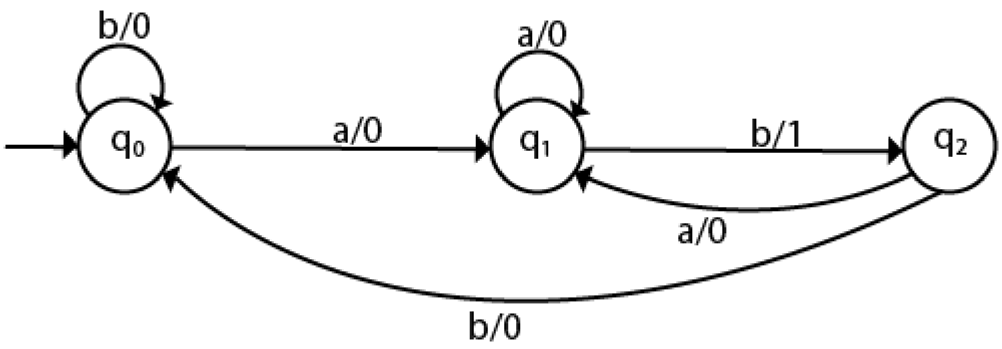
```
1. λ' (q2, a) = λ(δ(q2, a))
2.            = λ(q1)
3.            = 0
4.
5. λ' (q2, b) = λ(δ(q2, b))
6.            = λ(q0)
7.            = 0
```

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



Q \ Σ	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,



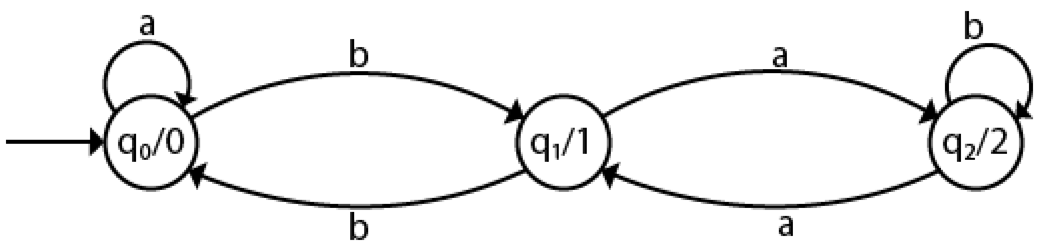
Example 3:

Convert the given Moore machine into its equivalent Mealy machine.

Q	**a**	**b**	**Output(λ)**
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. λ' (q0, a) = λ(δ(q0, a))
2. = λ(q0)
3. = 0
4.
5. λ' (q0, b) = λ(δ(q0, b))
6. = λ(q1)
7. = 1

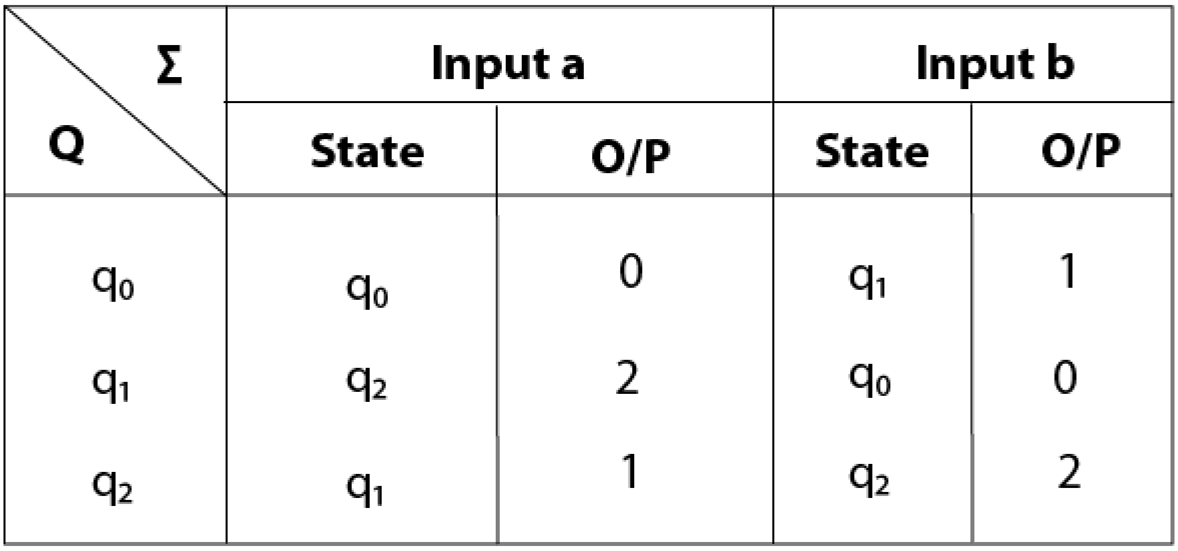
The λ for state q1 is as follows:

1. λ' (q1, a) = λ(δ(q1, a))
2. = λ(q2)
3. = 2
4.
5. λ' (q1, b) = λ(δ(q1, b))
6. = λ(q0)
7. = 0

The λ for state q2 is as follows:

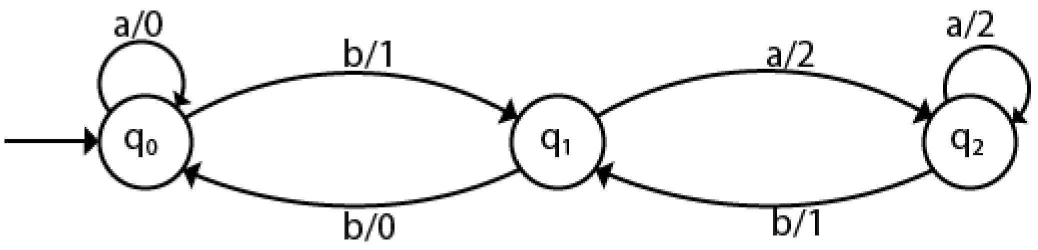
1. λ' (q2, a) = λ(δ(q2, a))
2. = λ(q1)
3. = 1
4.
5. λ' (q2, b) = λ(δ(q2, b))
6. = λ(q2)
7. = 2

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



Q \ Σ	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,



Mealy and Moore Machines in TOC

In the mealy machine shown in Figure 1, the output is represented with each input symbol for each state separated by /. The length of output for a mealy machine is equal to the length of input.

- **Input:**11
- **Transition:** δ (q0,11)=> δ(q2,1)=>q2
- **Output:** 00 (q0 to q2 transition has Output 0 and q2 to q2 transition also has Output 0)

__NOTE:__ If there are n inputs in the Mealy machine then it generates n outputs while if there are n inputs in the Moore machine then it generates n + 1 outputs.

Conversion from Mealy to Moore Machine

Let us take the transition table of the mealy machine shown in Figure 2.

	Input=0		**Input=1**	
Present State	**Next State**	**Output**	**Next State**	**Output**
q0	q1	0	q2	0
q1	q1	0	q2	1
q2	q1	1	q2	0

Table 1

Step 1. First find out those states which have more than 1 output associated with them. q1 and q2 are the state s which have both output 0 and 1 associated with them.

Step 2. Create two states for these states. For q1, two states will be q10 (state with output 0) and q11 (state wi th output 1). Similarly, for q2, two states will be q20 and q21.

Step 3. Create an empty moore machine with newly generated state. For more machines, Output will be assoc iated to each state irrespective of inputs.

	Input=0	Input=1	
Present State	Next State	Next State	Output
q0			
q10			
q11			
q20			
q21			

Table 2

Step 4. Fill the entries of next state using mealy machine transition table shown in Table 1. For q0 on input 0, next
state is q10 (q1 with output 0). Similarly, for q0 on input 1, next state is q20 (q2 with output 0). For q1 (both q10 and q11) on input 0, next state is q10. Similarly, for q1(both q10 and q11), next state is q21. For q10, out put will be 0 and for q11, output will be 1. Similarly, other entries can be filled.

	Input=0	Input=1	
Present State	Next State	Next State	Output
q0	q10	q20	0
q10	q10	q21	0
q11	q10	q21	1
q20	q11	q20	0
q21	q11	q20	1

Table 3

This is the transition table of moore machine shown in Figure 1.

Conversion from Moore machine to mealy machine

Let us take the moore machine of Figure 1 and its transition table is shown in Table 3. **Step 1.** Construct an e mpty mealy machine using all states of moore machine as shown in Table 4.

	Input=0		**Input=1**	
Present State	**Next State**	**Output**	**Next State**	**Output**
q0				
q10				
q11				
q20				
q21				

Table 4

Step 2: Next state for each state can also be directly found from Moore machine transition Table as:

	Input=0		**Input=1**	
Present State	**Next State**	**Output**	**Next State**	**Output**
q0	q10		q20	
q10	q10		q21	
q11	q10		q21	
q20	q11		q20	
q21	q11		q20	

Table 5

Step 3: As we can see output corresponds to each input in Moore machine transition table. Use this to fill the Output entries. e.g.; Output corresponding to q10, q11, q20 and q21 are 0, 1, 0 and 1 respectively.

	Input=0		**Input=1**	
Present State	**Next State**	**Output**	**Next State**	**Output**

q0	q10	0	q20	0
q10	q10	0	q21	1
q11	q10	0	q21	1
q20	q11	1	q20	0
q21	q11	1	q20	0

Table 6

Step 4: As we can see from table 6, q10 and q11 are similar to each other (same value of next state and Outp ut for different Inputs). Similarly, q20 and q21 are also similar. So, q11 and q21 can be eliminated.

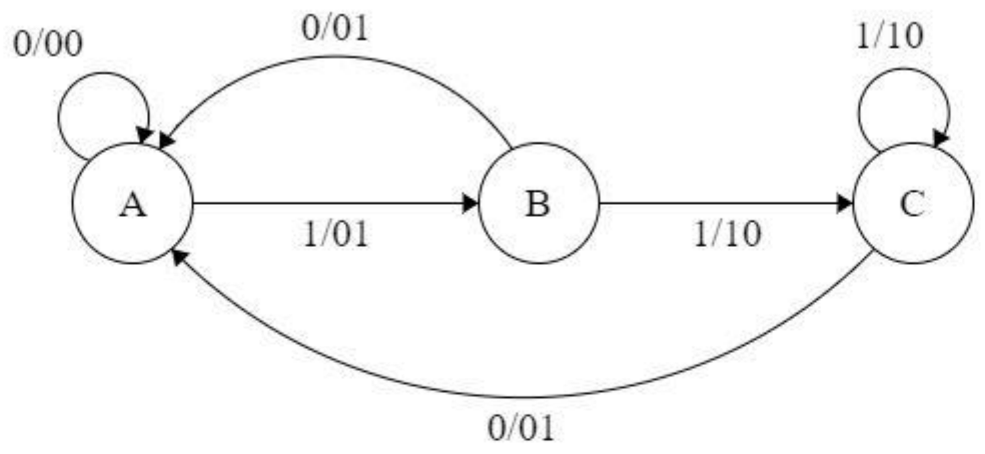
	Input=0		**Input=1**	
Present State	**Next State**	**Output**	**Next State**	**Output**
q0	q10	0	q20	0
q10	q10	0	q21	1
q20	q11	1	q20	0

Table 7

This is the same mealy machine shown in Table 1. So we have converted mealy to Moore machine and conve rted back moore to mealy.

Note: *Number of statAes in the mealy machine can't be greater than number of states in moore machine.*

Example: The Finite state machine is
described by the following state diagram with A as starting state, where an arc label is x / y and x stands for 1-bit input and y stands for 2-bit output?



Outputs the sum of the present and the previous bits of the input.

1. Outputs 01 whenever the input sequence contains 11.
2. Outputs 00 whenever the input sequence contains 10.
3. None of these.

Solution: Let us take different inputs and its output and check which option works:

Input: 01

Output: 00 01 (For 0, Output is 00 and state is A. Then, for 1, Output is 01 and state will be B)

Input: 11

Output: 01 10 (For 1, Output is 01 and state is B. Then, for 1, Output is 10 and state is C)

As we can see, it is giving the binary sum of the present and previous bit. For the first bit, the previous bit is taken as 0.

Difference between Mealy machine and Moore machine

Mealy Machine is defined as a machine in the theory of computation whose output values are determined by both its current state and current inputs. In this machine at most one transition is possible.
It has 6 tuples: (Q, q0, ∑, ▲, δ, λ')

1. Q is a finite set of states
2. q0 is the initial state
3. ∑ is the input alphabet
4. ▲ is the output alphabet
5. δ is the transition function that maps Q×∑ → Q
6. 'λ' is the output function that maps Q×∑→ ▲

Prerequisite – Mealy and Moore Machines

The diagram is as follows:

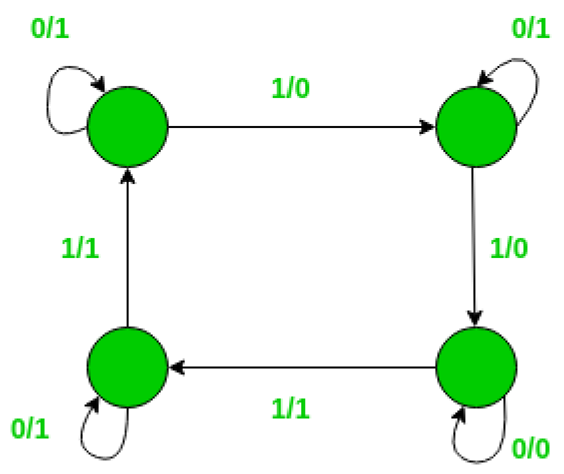


Figure - Mealy machine

Moore Machine:

Moore's machine is defined as a machine in the theory of computation whose output values are determined only by its current state. It has also 6 tuples

(Q, q0, ∑, ▲, δ, λ)

1. Q is a finite set of states
2. q0 is the initial state
3. ∑ is the input alphabet
4. ▲ is the output alphabet
5. δ is the transition function that maps Q×∑ → Q
6. λ is the output function that maps Q → ▲

Diagram:

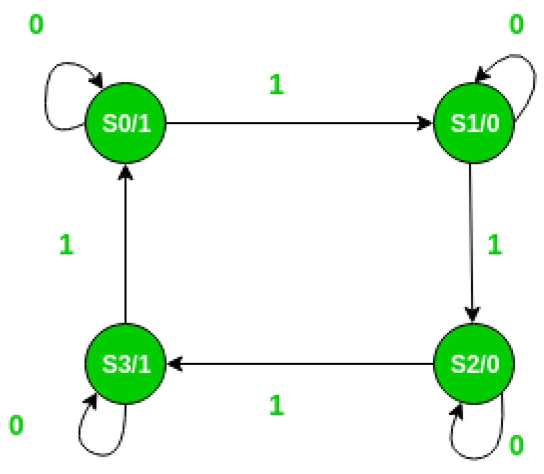


Figure - Moore machine

The difference between the Mealy machine and Moore machine is as follows:

Moore Machine	**Mealy Machine**
Output depends only upon the present state.	Output depends on the present state as well as present input.
Moore machine also places its output on the transition.	Mealy Machine places its output on the transition.
More states are required.	Less number of states are required.
There is less hardware requirement for circuit implementation.	There is more hardware requirement for circuit implementation.
They react slower to inputs(One clock cycle later).	They react faster to inputs.
Synchronous output and state generation.	Asynchronous output generation.
Output is placed on states.	Output is placed on transitions.
Easy to design.	It is difficult to design.