Chapter – 1

Fundamentals and Finite 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

∑ = {a, b}

∑ = {A, B, C, D}

∑ = {0, 1, 2}

∑ = {0, 1, ..., 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:

w = 010

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**

Chomsky Hierarchy Representation:

Chomsky Hierarchy represents the class of languages that are accepted by the different machine. The category of language in Chomsky's Hierarchy is as given below:

1. Type 0 known as Unrestricted Grammar.
2. Type 1 known as Context Sensitive Grammar.
3. Type 2 known as Context Free Grammar.
4. Type 3 Regular Grammar.

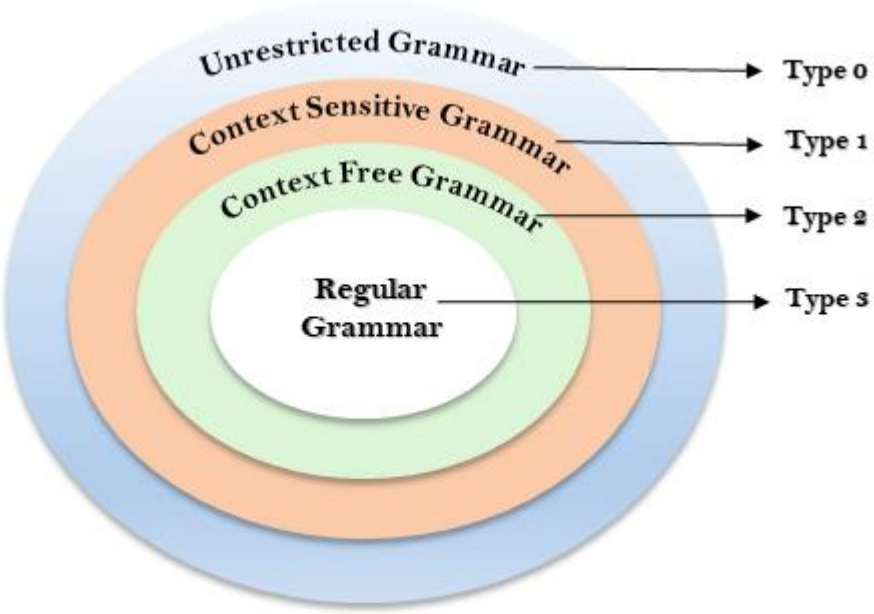


Fig: Chomsky Hierarchy

This is a hierarchy. Therefore every language of type 3 is also of type 2, 1 and 0. Similarly, every language of type 2 is also of type 1 and type 0, etc.

Type 0 Grammar:

Type 0 grammar is known as Unrestricted grammar. There is no restriction on the grammar rules of these types of languages. These languages can be efficiently modeled by Turing machines.

For example:

1. bAa → aa
2. S → s

Type 1 Grammar:

Type 1 grammar is known as Context Sensitive Grammar. The context sensitive grammar is used to represent context sensitive language. The context sensitive grammar follows the following rules:

- The context sensitive grammar may have more than one symbol on the left hand side of their production rules.
- The number of symbols on the left-hand side must not exceed the number of symbols on the right-hand side.
- The rule of the form A → ε is not allowed unless A is a start symbol. It does not occur on the right-hand side of any rule.

- The Type 1 grammar should be Type 0. In type 1, Production is in the form of V → T

Where the count of symbol in V is less than or equal to T.

For example:

S → AT

T → xy

A → a

Type 2 Grammar:

Type 2 Grammar is known as Context Free Grammar. Context free languages are the languages which can be represented by the context free grammar (CFG). Type 2 should be type 1. The production rule is of the form

A → α

Where A is any single non-terminal and is any combination of terminals and non-terminals.

For example:

A → aBb

A → b

B → a

Type 3 Grammar:

Type 3 Grammar is known as Regular Grammar. Regular languages are those languages which can be described using regular expressions. These languages can be modeled by NFA or DFA.

Type 3 is most restricted form of grammar. The Type 3 grammar should be Type 2 and Type 1. Type 3 should be in the form of

V → T*V / T*

For example:

A → xy

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.

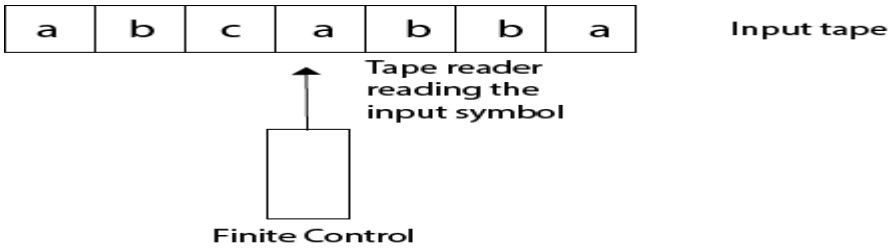
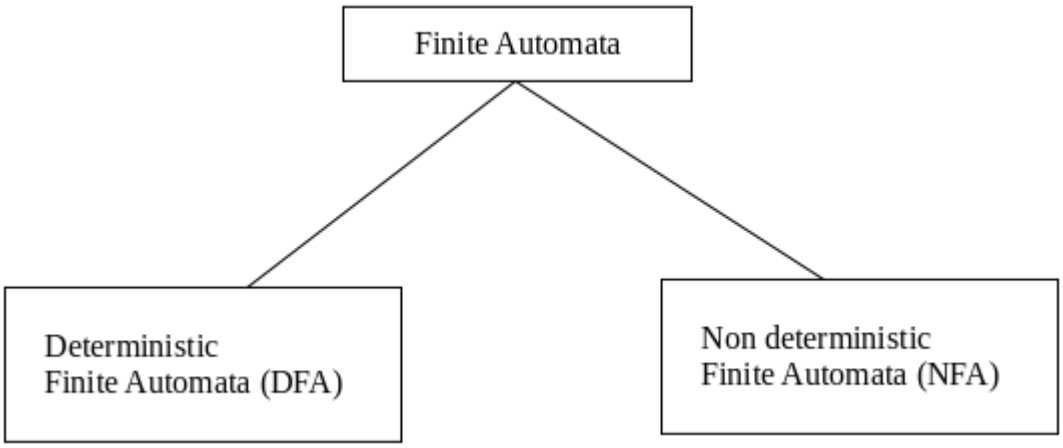


Fig :- Finite automata model

Types of Automata:

There are two types of finite automata:

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



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.

Acceptance of Strings and Languages:

- Language accepted by DFA

Contents

Here we are going to formally define what is meant by a DFA (deterministic finite automaton) accepting a string or a language.

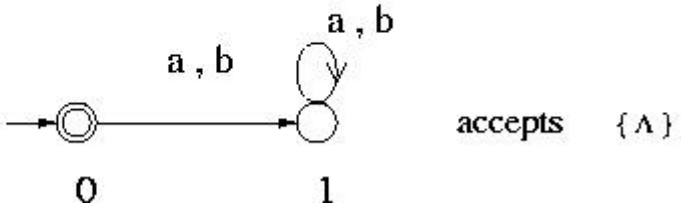
A string w is accepted by a DFA $< Q , \quad , q_0 , \quad , A >$, if and only if

$^{*}(q_0 , w) \quad A$. That is a string is accepted by a DFA if and only if the DFA starting at the initial state ends in an accepting state after reading the string.

A language L is accepted by a DFA $< Q , \quad , q_0 , \quad , A >$, if and only if

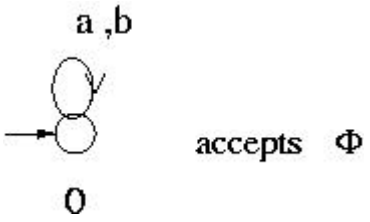
L = { w | *(q_0 , w) A } . That is, the language accepted by a DFA is the set of strings accepted by the DFA.

Example 1 :



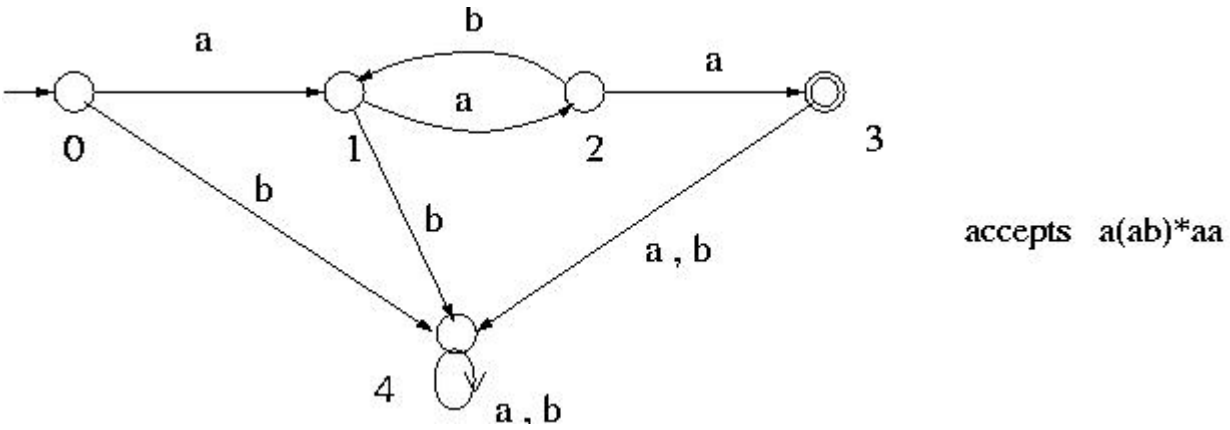
This DFA accepts {Lamda} because it can go from the initial state to the accepting state (also the initial state) without reading any symbol of the alphabet i.e. by reading an empty string . It accepts nothing else because any non-empty symbol would take it to state 1, which is not an accepting state, and it stays there.

Example 2 :



This DFA does not accept any string because it has no accepting state. Thus the language it accepts is the empty set ϕ .

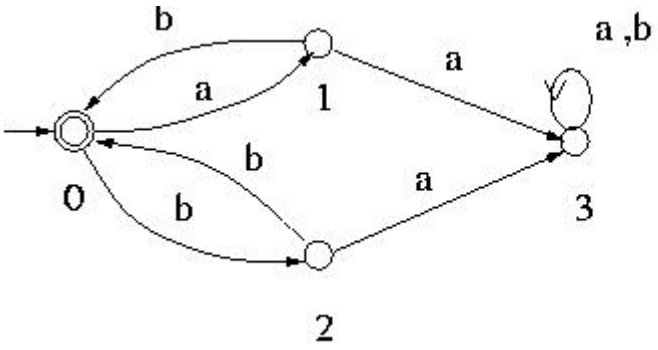
Example 3 : DFA with one cycle



This DFA has a cycle: 1 - 2 - 1 and it can go through this cycle any number of times by reading substring ab repeatedly.

To find the language it accepts, first from the initial state go to state 1 by reading one a. Then from state 1 go through the cycle 1 - 2 - 1 any number of times by reading substring ab any number of times to come back to state 1. This is represented by $(ab)^*$. Then from state 1 go to state 2 and then to state 3 by reading aa. Thus a string that is accepted by this DFA can be represented by $a(ab)^*aa$.

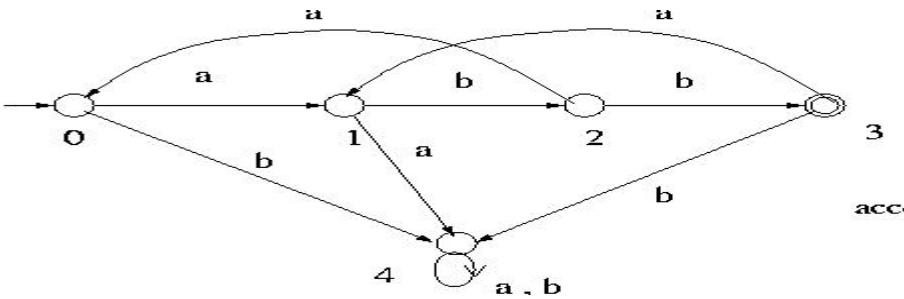
Example 4 : DFA with two independent cycles



accepts (ab + bb)*

This DFA has two independent cycles: 0 - 1 - 0 and 0 - 2 - 0 and it can move through these cycles any number of times in any order to reach the accepting state from the initial state such as 0 - 1 - 0 - 2 - 0 - 2 - 0. Thus a string that is accepted by this DFA can be represented by $(ab + bb)^*$.

Example 5 : DFA with two interleaved cycles



accepts a(bba + baa)*bb

This DFA has two cycles: 1 - 2 - 0 - 1 and 1 - 2 - 3 - 1. To find the language accepted by this DFA, first from state 0 go to state 1 by reading a (any other state which is common to these cycles such as state 2 can also be used instead of state 1). Then from state 1 go through the two cycles 1 - 2 - 0 - 1 and 1 - 2 - 3 - 1 any number of times in any order by reading substrings baa and bba, respectively. At this point a substring $a(baa + bba)^*$ will have been read. Then go from state 1 to state 2 and then to state 3 by reading bb. Thus altogether $a(baa + bba)^*bb$ will have been read when state 3 is reached from state 0.

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.

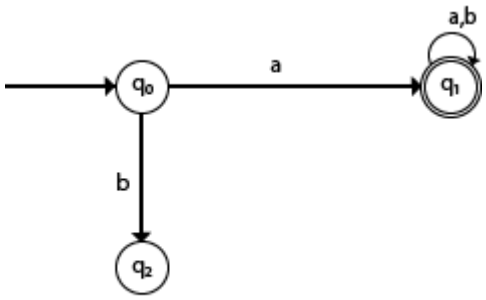


Fig:- DFA

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:

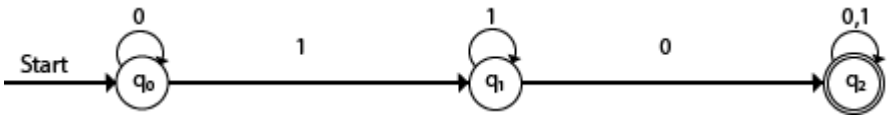
1. The state is represented by vertices.
2. The arc labeled with an input character show the transitions.
3. The initial state is marked with an arrow.
4. The final state is denoted by a double circle.

Example 1:

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

Solution:

Transition Diagram:



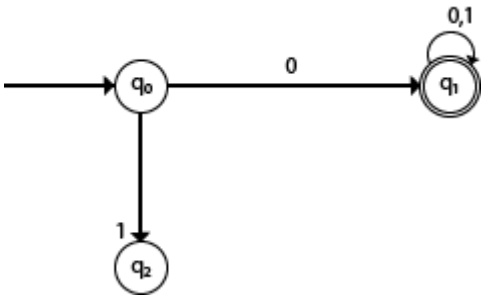
Transition Table:

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:



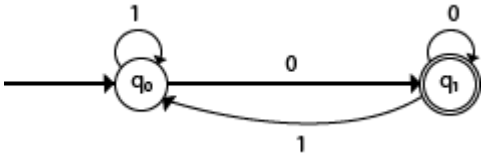
Explanation:

- 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.

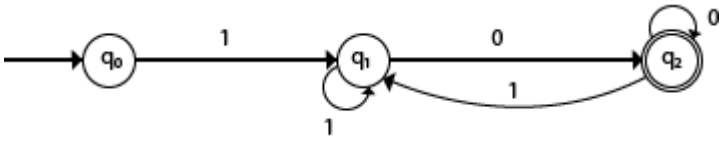
Examples of DFA

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:

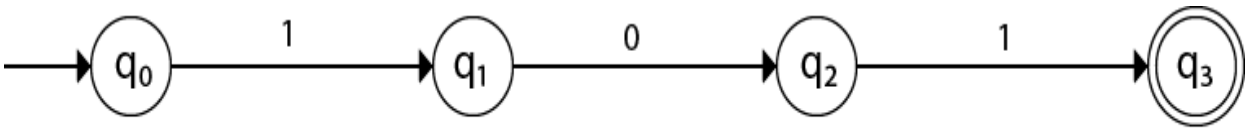


Fig: FA

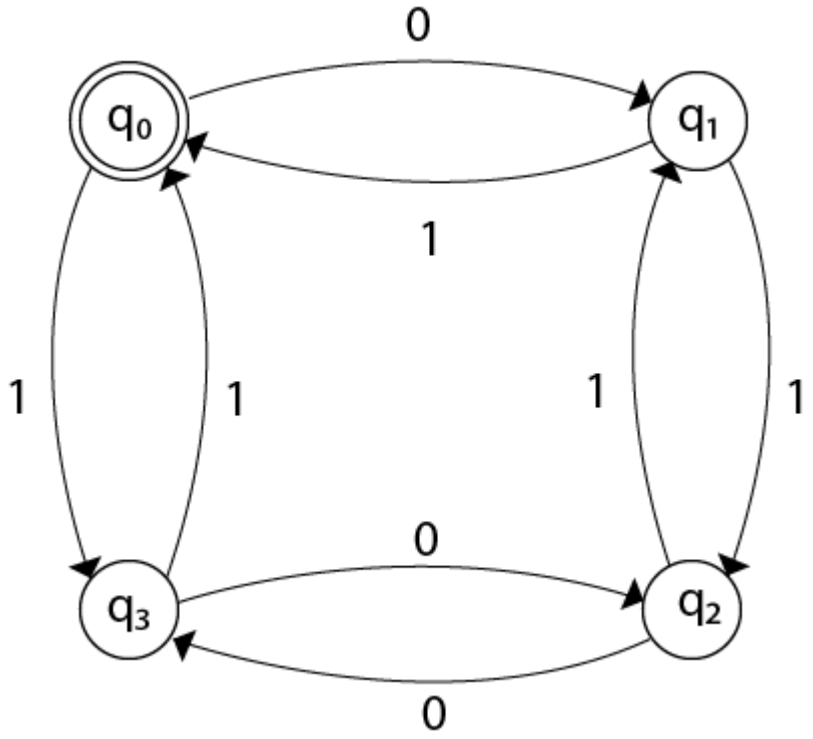
In the given solution, we can see that only input 101 will be accepted. Hence, for input 101, there is no other path shown for other input.

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:

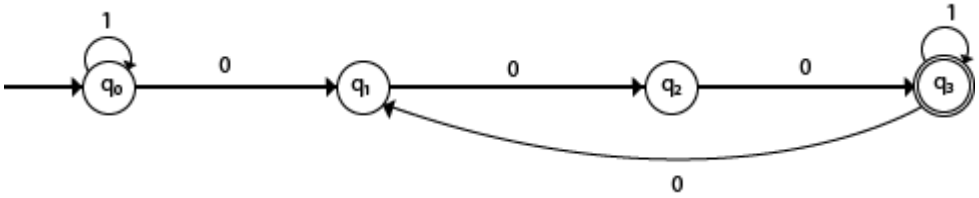
q0: state of even number of 0's and even number of 1's.
q1: state of odd number of 0's and even number of 1's.
q2: state of odd number of 0's and odd number of 1's.
q3: state of even number of 0's and odd number of 1's.

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:



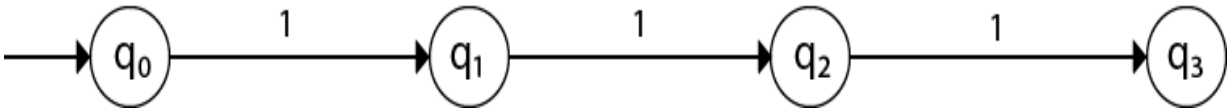
Note that the sequence of triple zeros is maintained to reach the final state.

Example 5:

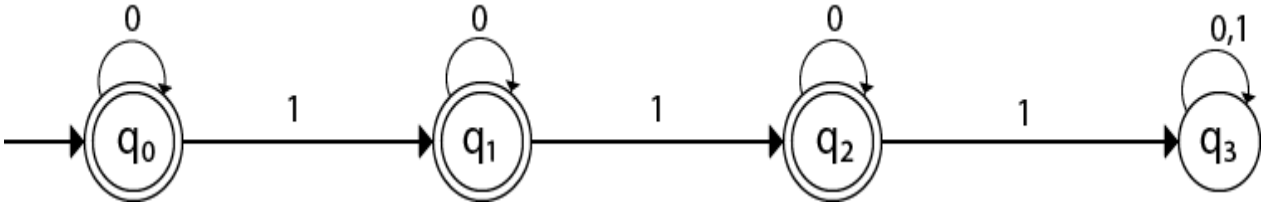
Design a DFA L(M) = {w | w ε {0, 1}*} and W is a string that does not contain consecutive 1's.

Solution:

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



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



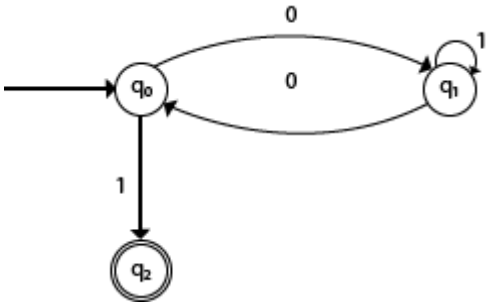
The stages q0, q1, q2 are the final states. The DFA will generate the strings that do not contain consecutive 1's like 10, 110, 101,..... etc.

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:



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.

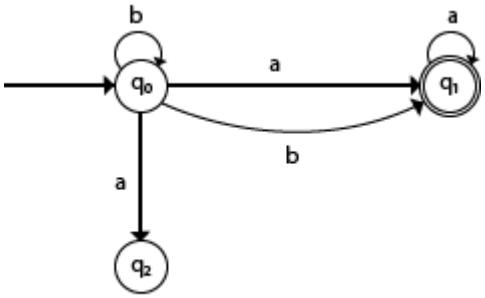


Fig:- NDFA

Formal definition of NFA:

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

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

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:

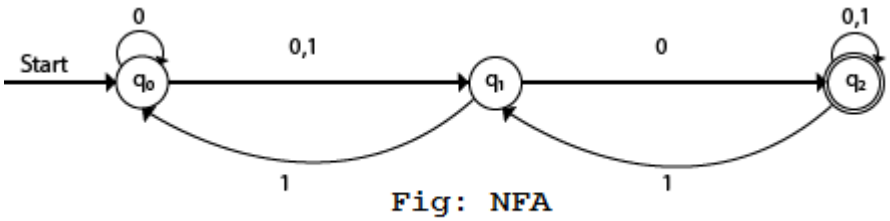


Fig: NFA

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:

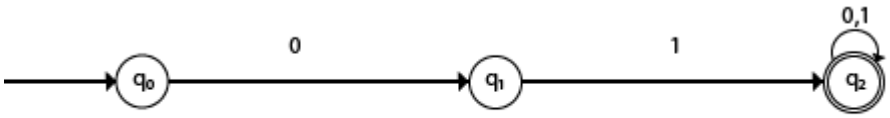


Fig: NFA

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:

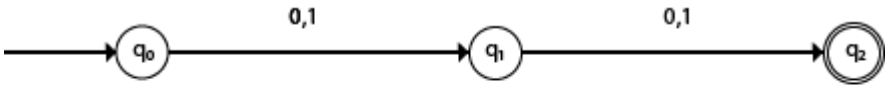


Fig: NFA

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:

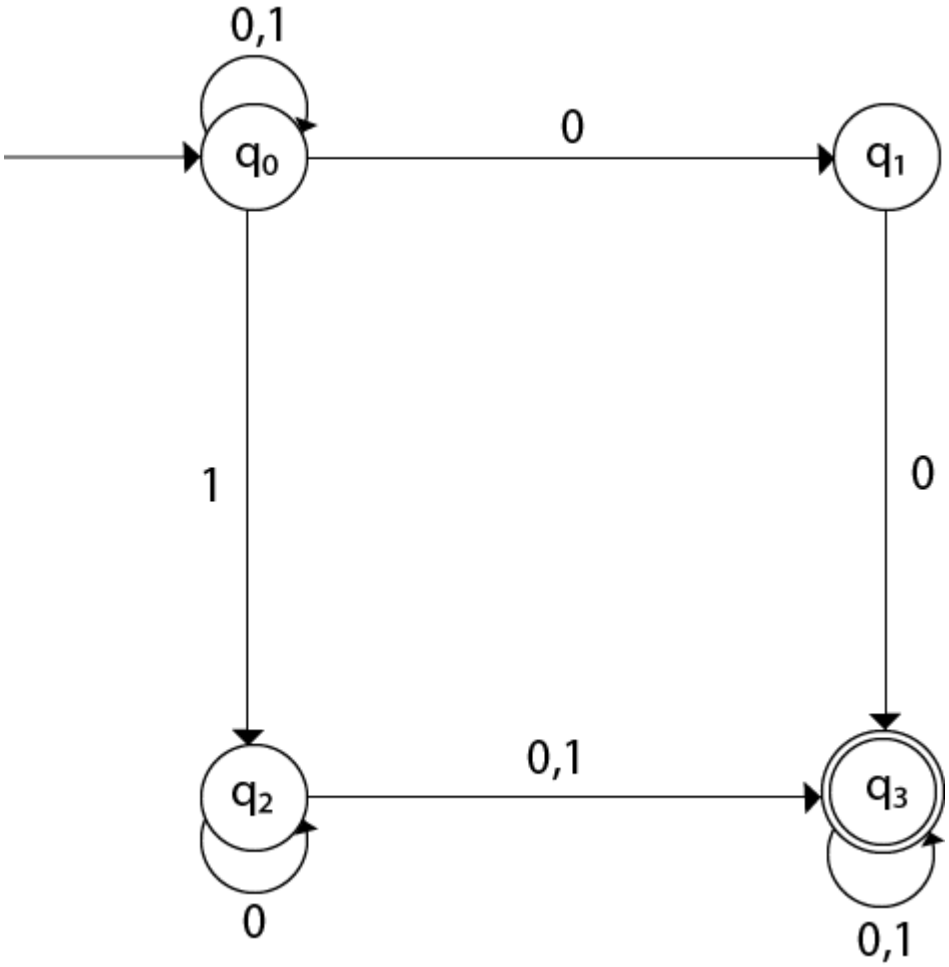
Example 1:

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}

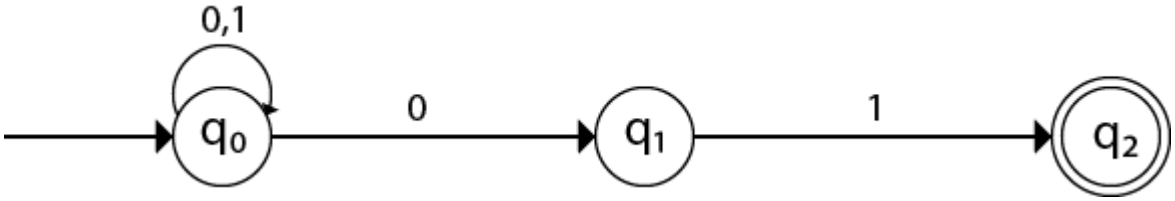
Example 2:

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

Solution:

Anything either 0 or 1	0 1

Hence, NFA would be:

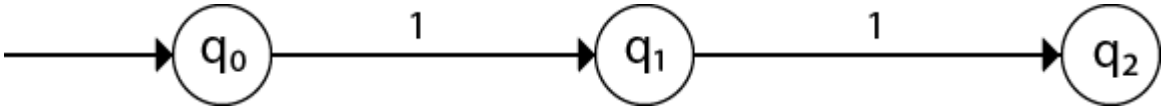


Example 3:

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

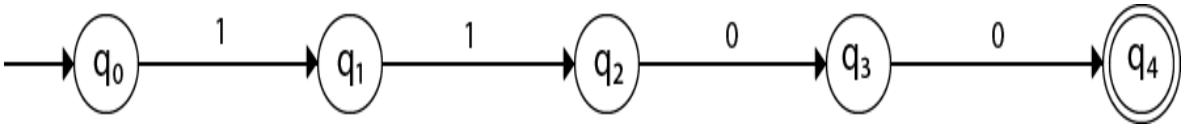
Solution:

The FA with double 1 is as follows:



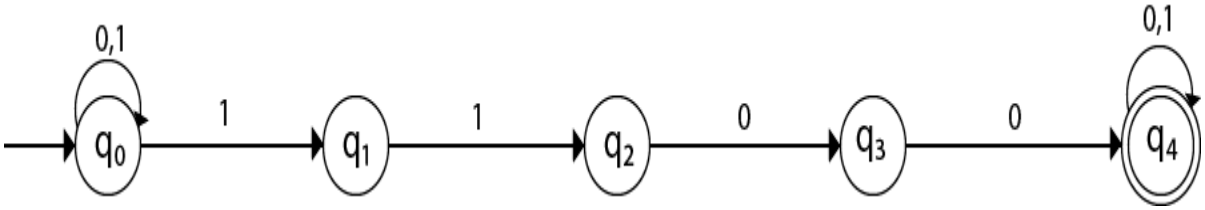
It should be immediately followed by double 0.

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

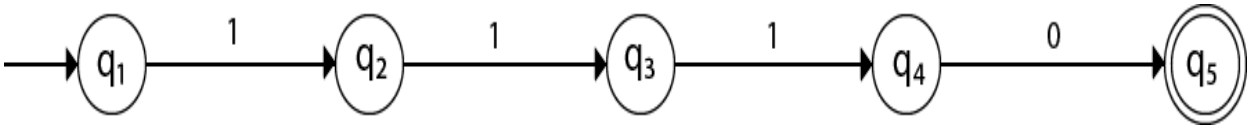
1. q0 → q1 → q2 → q3 → q4 → q4 → q4 → q4

Example 4:

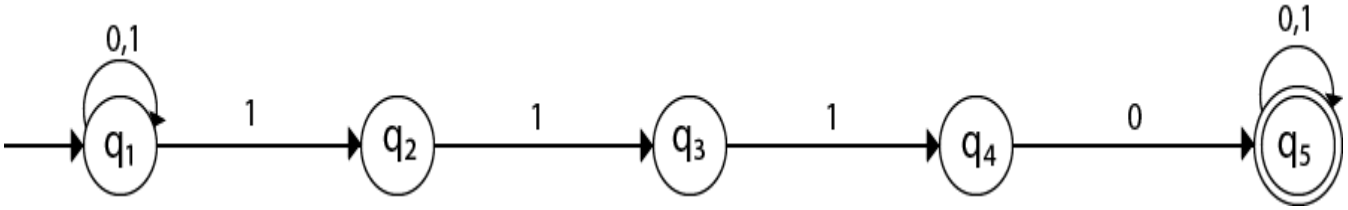
Design an NFA in which all the string contain a substring 1110.

Solution:

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



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.

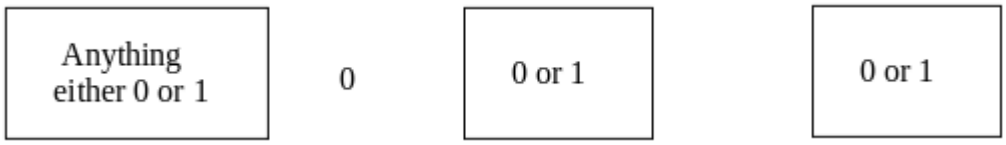
1. δ(q1, 111010) = δ(q2, 1100)
2. = δ(q3, 100)
3. = δ(q4, 00)
4. = δ(q5, 0)
5. = δ(q5, ε)

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

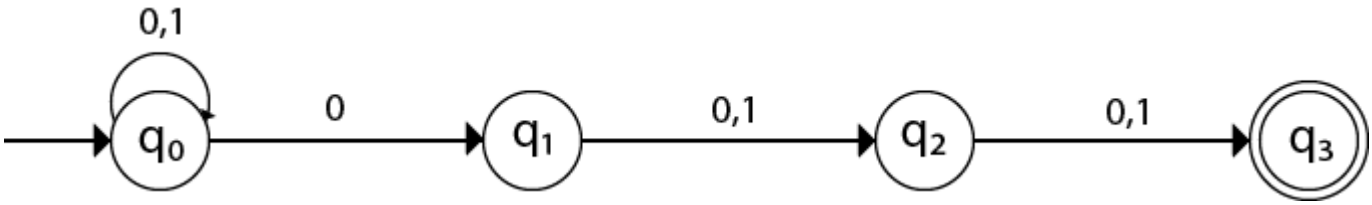
Example 5:

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.

Equivalence Of NFA and 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

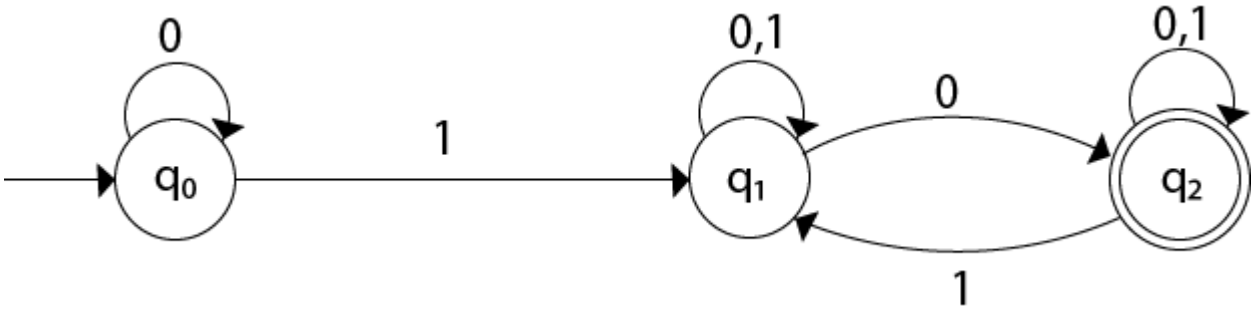
Step 2: Add q0 of NFA to Q'. Then find the transitions from this start state.

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].

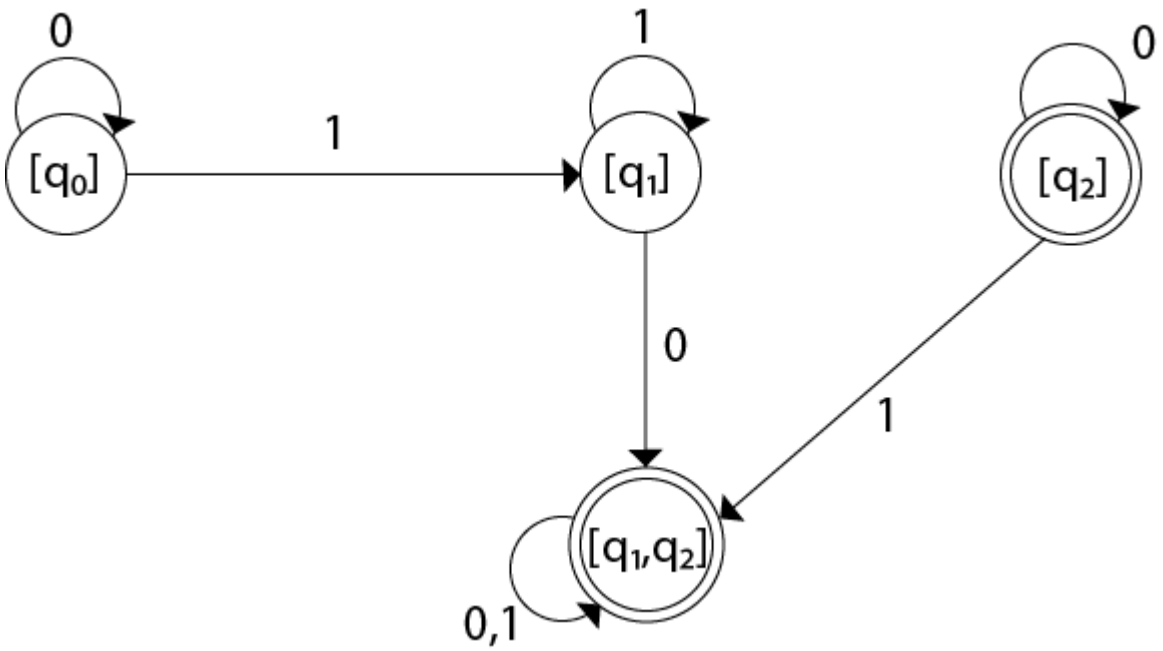
1. δ'([q1, q2], 0) = δ(q1, 0) ∪ δ(q2, 0)
2. = {q1, q2} ∪ {q2}
3. = [q1, q2]
4. δ'([q1, q2], 1) = δ(q1, 1) ∪ δ(q2, 1)
5. = {q1} ∪ {q1, q2}
6. = {q1, q2}
7. = [q1, q2]

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]

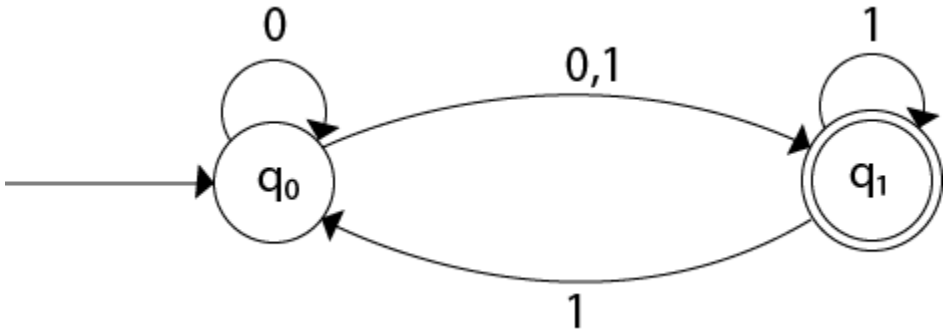
The Transition diagram will be:



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]

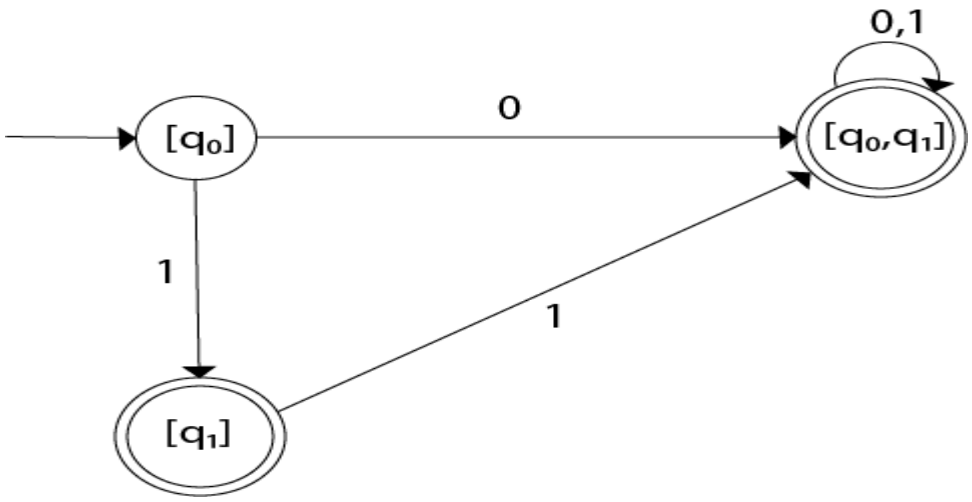
As in the given NFA, q1 is a final state, then in DFA wherever, q1 exists that state becomes a final state. Hence in the DFA, final states are [q1] and [q0, q1]. Therefore set of final states F = {[q1], [q0, q1]}.

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:

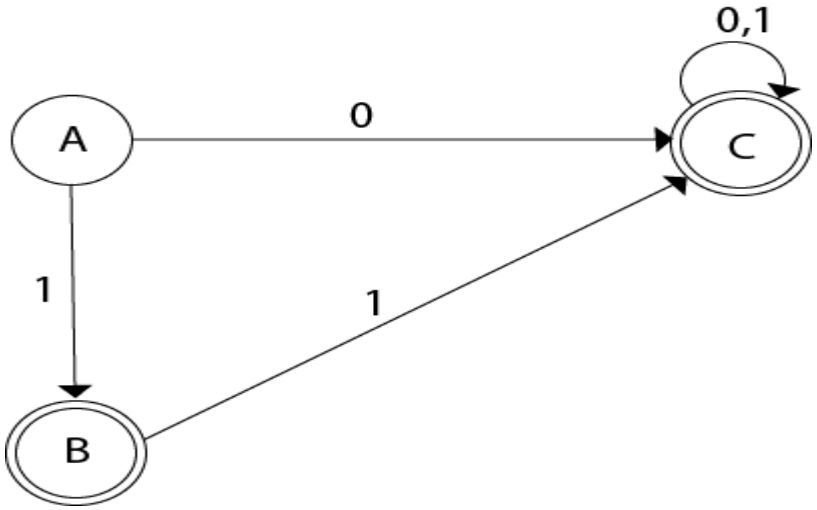


Even we can change the name of the states of DFA.

Suppose

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

With these new names the DFA will be as follows:



Equivalence of NFA with € to NFA without €:

The method is mentioned below stepwise −

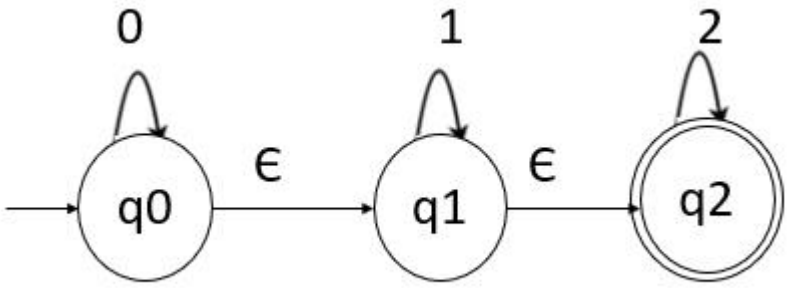
- **Step 1** − Find out all the ε-transitions from each state from Q. That will be called as ε-closure(qi) where, qi ∈Q.
- **Step 2** − Then, $\delta 1$ transitions can be obtained. The $\delta 1$ transitions means an ε-closure on δ moves.
- **Step 3** − Step 2 is repeated for each input symbol and for each state of given NFA.
- **Step 4** − By using the resultant status, the transition table for equivalent NFA without ε can be built.

NFA with ε to without ε is as follows −

δ1(q,a) = ∈ - closure (δ (δ∈^(q,∈**ϵ**),a)) where, δ^(q,∈**ϵ**) = ∈ - closure(q)

Example

Convert the given NFA with epsilon to NFA without epsilon.



Solution

We will first obtain ε-closure of each state i.e., we will find ε-reachable states from the current state.

Hence,

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

ε-closure(q0) means with null input (no input symbol) we can reach q0, q1, q2.

In a similar manner for q1 and q2 ε-closure are obtained.

Now we will obtain $\delta 1$ transitions for each state on each input symbol as shown below −

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

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

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

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

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

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

δ'(q2, 0) = ε-closure(δ(δ^(q2, ε),0))
= ε-closure(δ(q2), 0))
= ε-closure(δ(q2, 0))
= ε-closure(Φ)
= Φ

δ'(q2, 1) = ε-closure(δ(δ^(q2, ε),1))
= ε-closure(δ(q2), 1)
= ε-closure(δ(q2, 1))
= ε-closure(Φ)
= Φ

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

Now, we will summarize all the computed δ' transitions as given below −

δ'(q0,0)={q0,q1,q2}
δ'(q0,1)={q1,q2}
δ'(q0,2)={q2}

δ'(q1,0)= { Φ }
δ'(q1,1)={q1,q2}
δ'(q1,2)={q2}

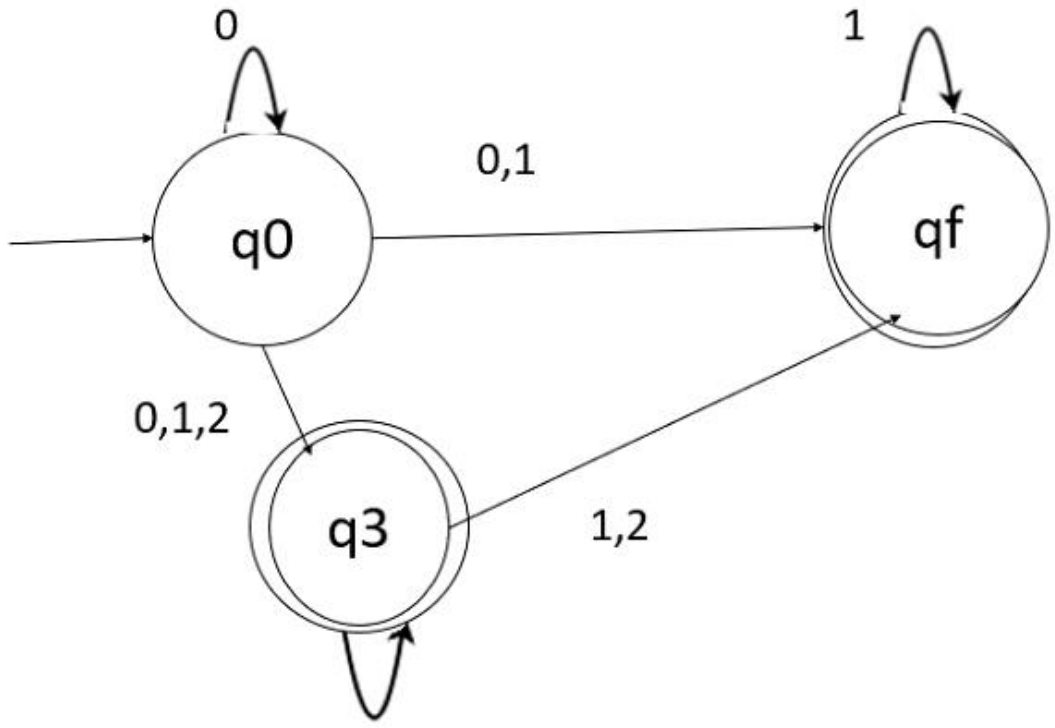
δ'(q2,0)={ Φ }
δ'(q2,1)={ Φ }
δ'(q2,2)={q2}

The **transition table** is given below −

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

NFA without epsilon

The NFA without epsilon is given below −



Here, q0, q1, q2 are final states because ε-closure(q0), ε-closure(q1) and ε-closure(q2) contain a final state q2.

Minimization of DFA:

The process of reducing a given DFA to its minimal form is called as minimization of DFA.

- It contains the minimum number of states.
- The DFA in its minimal form is called as a **Minimal DFA**.
- 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. δ (q, a) = p
2. δ (r, a) = p

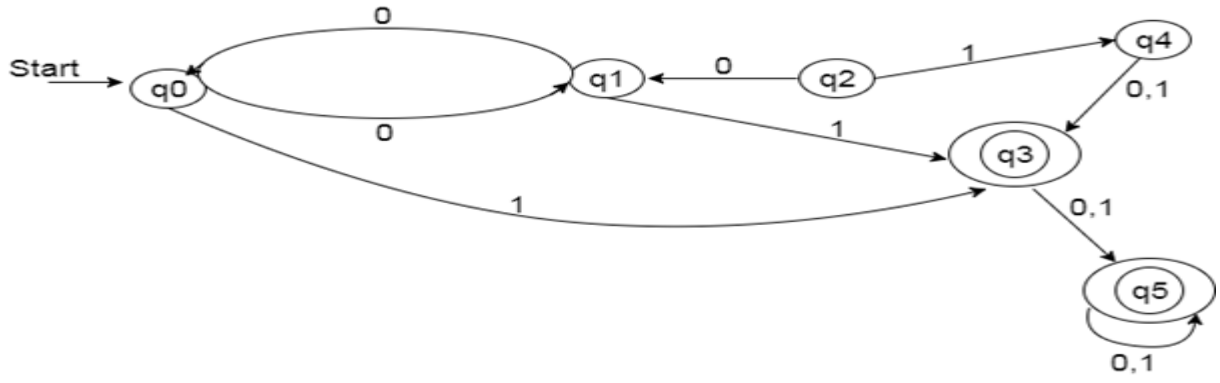
That means, find the two states which have the same value of a and b and remove one of them.

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.

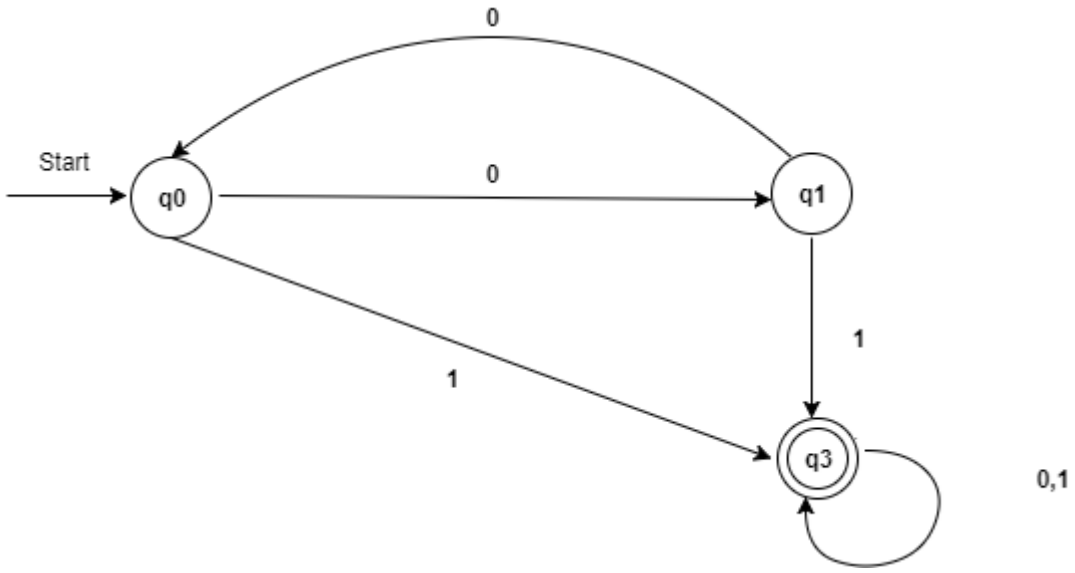
State	0	1
q3	q3	q3

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.



Equivalence of two DFA's:

Two Automaton are equivalent if they satisfy the following conditions :
1. The initial and final states of both the automatons must be same.
2. Every pair of states chosen is from a different automaton only.
3. While combining the states with the input alphabets, the pair results must be either both final states or intermediate states.(i.e both should lie either in the final state or in the non-final state).
4. If the resultant pair has different types of states, then it will be non-equivalent. (i.e. One lies in the final state and the other lies in the intermediate state).

Example 1 –
Consider Two Different Automaton shown below in Figure 1.

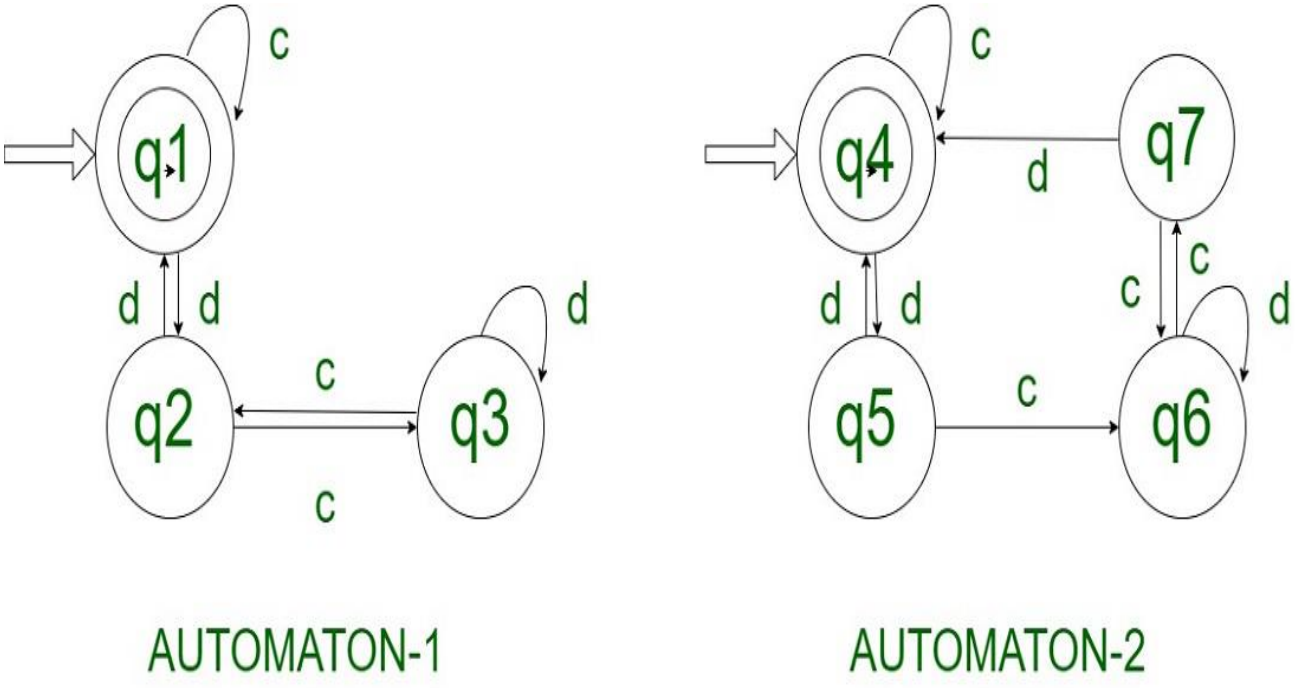


FIGURE -1

Solution –
Step 1 – Since the initial and final states of both the automaton are the same, so it verifies.
Step 2 – Check for each state by making a table of states with respect to input alphabets.

States	c	d
(q1,q4)	(q1,q4)=>(F.S,F.S)	(q2,q5)=>(I.S,I.S)
(q2,q5)	(q3,q6)=>(I.S,I.S)	(q1,q4)=>(F.S,F.S)
(q3,q6)	(q2,q7)=>(I.S,I.S)	(q3,q6)=>(I.S,I.S)
(q2,q7)	(q3,q6)=>(I.S,I.S)	(q1,q4)=>(F.S,F.S)

Here ,
F.S represents -> Final State.
I.S represents -> Intermediate State (Non-Final State) .

Step 3 – For every pair of states, the resultant states lie either in F.S or in I.S as a combination.
Conclusion – Both the Automaton are equivalent.

Note : If the resultant pair of states has a different combination of states(i.e. either (F.S, I.S) or (I.S, F.S), then both the automata are said to be non-equivalent.

Example 2 –
Consider Two Different Automaton shown below in Figure 2.

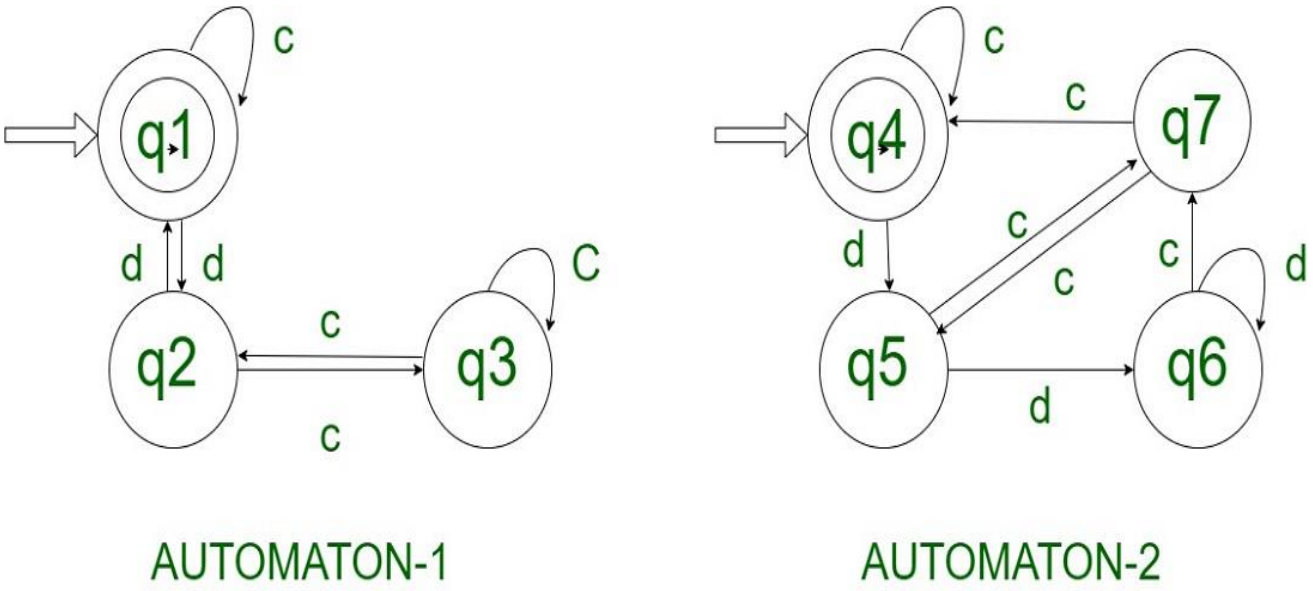


FIGURE -2

Solution –
Step 1 – Since the initial and final states of both the automaton are the same so it verifies.
Step 2 – Check for each state by making a table of states with respect to input alphabets.

States	c	d
(q1,q4)	(q1,q4)=>(F.S,F.S)	(q2,q5)=>(I.S,I.S)
(q2,q5)	(q3,q7)=>(I.S,I.S)	(q1,q6)=>(F.S,I.S) This is not a valid pair

Here ,
F.S represents -> Final State.
I.S represents -> Intermediate State (Non-Final State) .
Step 3 – For the first pair of states, the resultant states lie in F.S as a combination

Step 4 – But for the pair (q2,q5) when operated over input alphabets they lie in different states. (i.e. one in F.S and the other in I.S).
Conclusion – The Given Automata are not equivalent.

Example 3 –
Consider Two Different Automaton shown below in Figure 3.

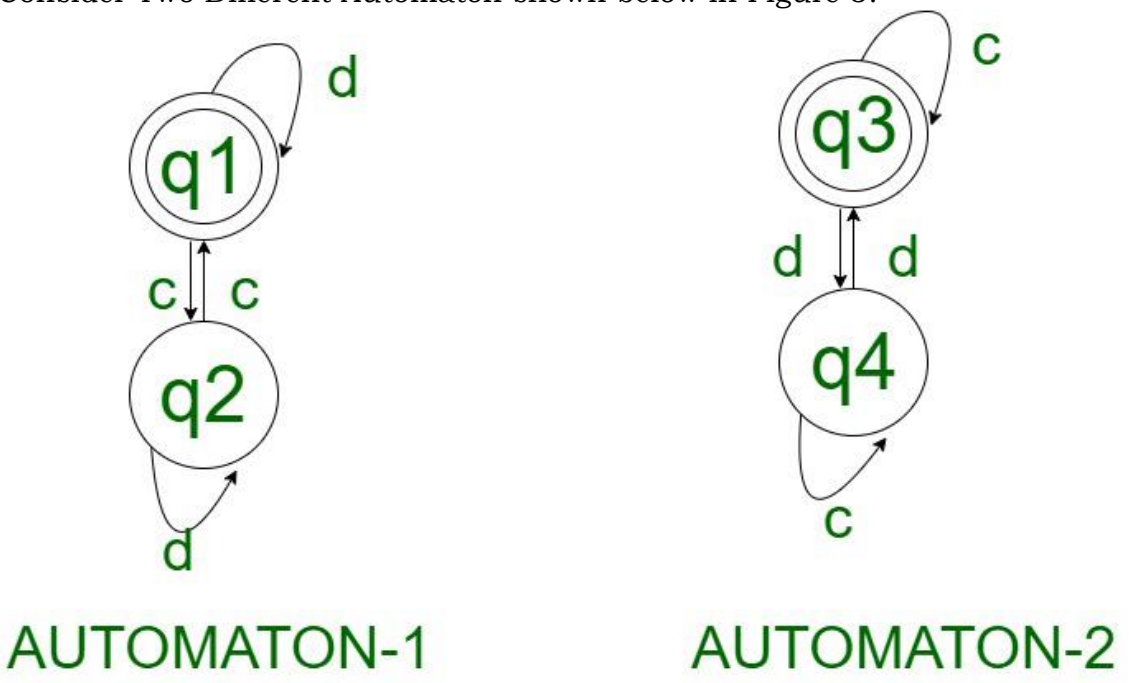


FIGURE -3

Solution –
Step 1 – Since the initial and final states of both the automaton are the same so it verifies.
Step 2 – Check for each state by making a table of states with respect to input alphabets.

States	C	d
(q1,q3)	(q2,q3) ⟶ (I.S,F.S) (This is not a valid pair)	(q1,q4) ⟶ (F.S,I.S) (This is not a valid pair)

Here,
F.S represents -> Final State.
I.S represents -> Intermediate State (Non-Final State) .

Step 3 – For the pair (q1,q3), when operated over input alphabets they lie in different states. (i.e. one in F.S and the other in I.S).
Conclusion – The Given Automata are not equivalent.

Finite automata with Output:

Two machines are FA with output

a) Moore Machine
b) Mealy Machine

Moore Machines

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 the transition function which maps Q×∑ → Q.*
- *λ is the output function which maps Q → O.*

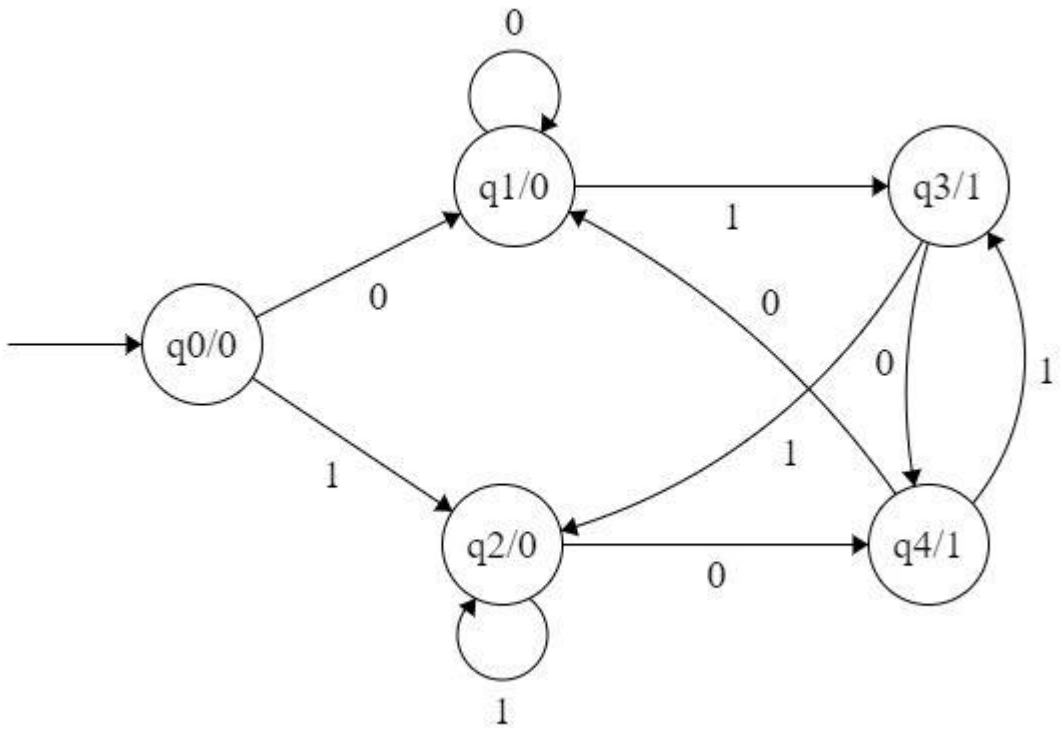


Figure 1: Moore Machines

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

- **Input:** 1,1
- **Transition:** δ (q0,1,1)=> 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 their 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.*

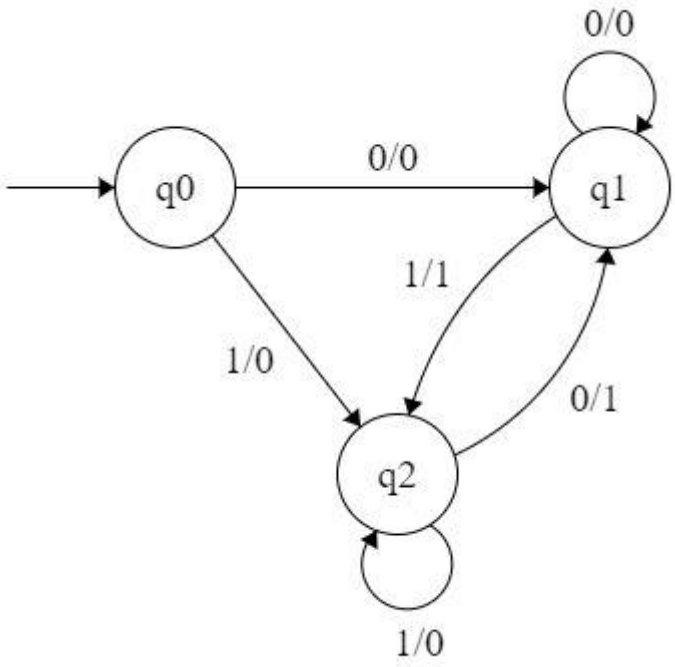


Figure 2:Mealy Machines

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

Moore Machines vs Mealy Machines

Aspect	**Moore Machines**	**Mealy Machines**
Output	Outputs depend only on the current state.	Outputs depend on the current state and input.

Aspect	Moore Machines	Mealy Machines
Number of States	Tends to require more states due to separate output behavior.	Might require fewer states as outputs are tied to transitions.
Response Time	Slower response to input changes as outputs update on state changes.	Faster response to input changes due to immediate output updates.
Complexity	Can be simpler due to separation of output behavior.	Can be more complex due to combined state-input cases.

For more differences, refer to Difference Between Moore and Mealy Machines.

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 wit h them. q1 and q2 are the states
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 (a state with output 0) and q11 (a state with output 1). Similarly, for q2, two states will be q20 and q21.

Step 3: Create an empty Moore machine with a
newly generated state. For more machines, Output will be associated with each state irrespective of inputs.

	Input=0	Input=1	

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

Table 2

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

	Input=0	Input=1	
q21	q11	q20	1

Table 3

This is the transition table of the 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 empty mealy machine using all states of the 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	

	Input=0		Input=1	
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 the Moore machine transition table. Use this to fill the Output entries.

Example: Output corresponding to q10, q11, q20 and q21 are 0, 1, 0 and 1 respe ctively.

	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 Output for
different Inputs). Similarly, q20 and q21 are also similar. So, q11 and q21 can be eliminated.

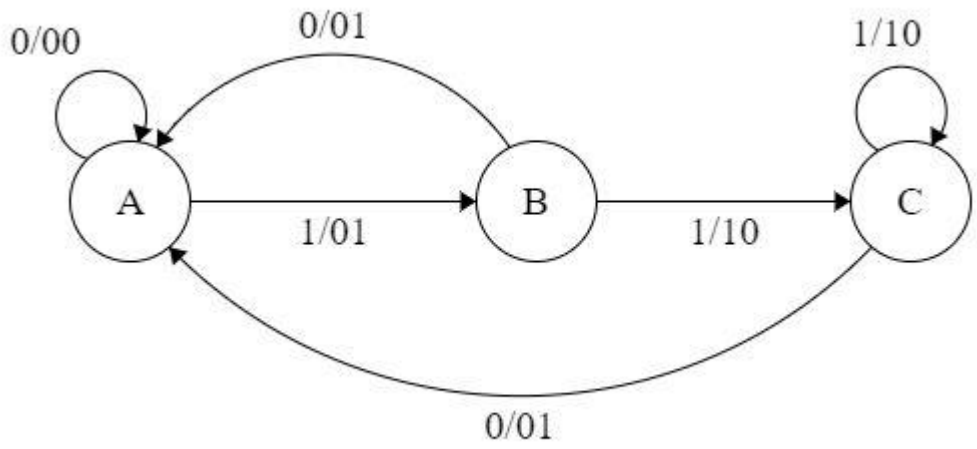
	Input=0		Input=1	
Present State	**Next State**	**Output**	**Next State**	**Output**
q0	q10	0	q20	0
q10	q10	0	q21	1
q20	q11	1	q20	0

Table 7

This is the same mealy machine shown in Table 1. So we have converted Mealy to Moore machine and converted 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.



Moore to Mealy Machines

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 wor ks:

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 s tate 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.