Chapter 2: **Finite Automata and Regular Language**

Finite Automata and Regular Language (10 hours) 13 marks

2.1 Introduction to finite automata, finite state machine
2.2 Deterministic finite automata (DFA), representation of DFA, language of DFA, design of DFA
2.3 Non deterministic finite automata (NFA), equivalence of DFA and NFA
2.4 Finite automata with epsilon transition (ε - NFA), equivalence of NFA and ε –NFA, equivalence of DFA and ε – NFA
2.5 Regular expressions and regular languages
2.6 Equivalence of regular expression and finite automata
2.7 Closure properties of regular languages
2.8 Pumping lemma for regular languages
2.9 Decision algorithm for regular language

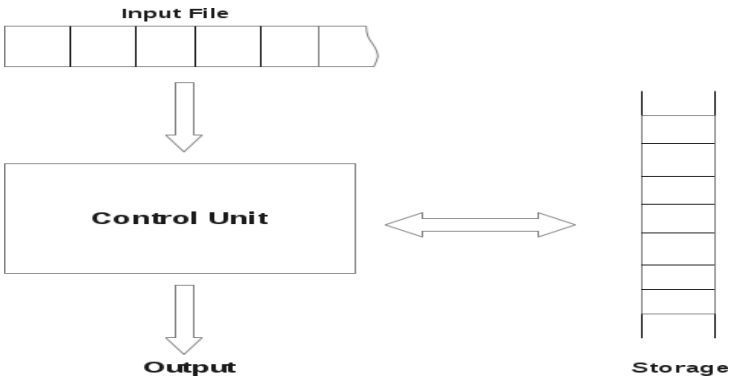
2.1 Introduction to finite automata, finite state machine

Automation:

- An automation is defined as a system that performs certain functions without human intervention.
- It accepts some input as row materials and converts it to a product under the guidance of control signals.
- An automation can be designed to perform a variety of tasks related to various domain of human life.

In terms of computer science, an automation is an abstract model of a digital computer. An automation has some features.

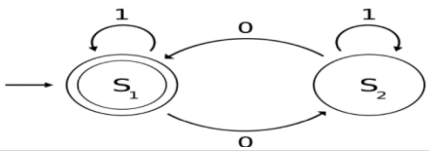
following figure shows the representation of an automation.



Input: An automation has a mechanism for reading input. It is assumed that input is in a file. The input file is divided into cells. Each cell can hold one symbol. Input mechanism can read input file from left to right.
Output: The automation can produce output of some form.
Storage: An automation can have a temporary storage device. the storage device can consist of unlimited number of cells. The automation can read and change the contents of the storage cells.
Control Unit: Automation has a control unit. At a given time, control unit is in some internal state.

Automata Theory

- The study of the mathematical properties of abstract machine or automata is automata theory.
- In theoretical computer science, automata theory is the study of abstract machines (or more appropriately, abstract 'mathematical' machines or systems, as they are described in mathematical terms) and the computational problems that can be solved using these machines. These abstract machines are called automata. Automata come from the Greek word, which means "self-acting".



- The figure above illustrates a finite state machine, which belongs to one well-known variety of automatons. This automaton consists of states (represented in the figure by circles), and transitions (represented by arrows). As the automaton sees a symbol of input, it makes a transition (or jump) to another state, according to its transition function (which takes the current state and the recent symbol as its inputs).
- Automata theory is also closely related to formal language theory.
- An automaton is a finite representation of a formal language that may be an infinite set.
- Automata are often classified by the class of formal languages they are able to recognize.
- Automata play a major role in theory of computation, compiler design, parsing and formal verification.

The study of automata is important because,

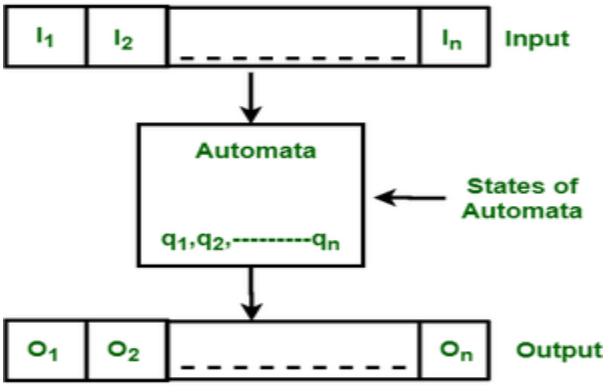
1. Automata theory plays an important role when we make software for designing and checking the behavior of a digital circuit.
2. The lexical analysis of a compiler breaks a program into logical units, such as variables, keywords and punctuation using this mechanism.
3. Automata theory works behind software for scanning large bodies of text, such as web pages to find occurrence of words, phrases etc.
4. Automata theory is a key to software for verifying systems of all types (software testing).
5. Automata theory is the most useful concept of software for natural language processing.

Finite-state machine:

- A finite-state machine (FSM) or finite-state automaton (plural: automata), or simply a state machine, is a mathematical model used to design computer programs and digital logic circuits.
- It is conceived as an abstract machine that can be in one of a finite number of states.
- The machine is in only one state at a time; the state it is in at any given time is called the current state.
- It can change from one state to another when initiated by a triggering event or condition, this is called a transition.
- A particular FSM is defined by a list of the possible transition states from each current state, and the triggering condition for each transition.
- Finite-state machines can model a large number of problems, among which are electronic design automation, communication protocol design, parsing and other engineering applications.
- In biology and artificial intelligence research, state machines or hierarchies of state machines are sometimes used to describe neurological systems and in linguistics—to describe the grammars of natural languages.

Introduction of Finite Automata:

- Finite automata are abstract machines used to recognize patterns in input sequences, forming the basis for understanding regular languages in computer science.
- They consist of states, transitions, and input symbols, processing each symbol step-by-step.
- If the machine ends in an accepting state after processing the input, it is accepted; otherwise, it is rejected.
- Finite automata come in deterministic (DFA) and non-deterministic (NFA), both of which can recognize the same set of regular languages.
- They are widely used in text processing, compilers, and network protocols



Features of Finite Automata

Input: Set of symbols or characters provided to the machine.
Output: Accept or reject based on the input pattern.
States of Automata: The conditions or configurations of the machine.
State Relation: The transitions between states.
Output Relation: Based on the final state, the output decision is made.

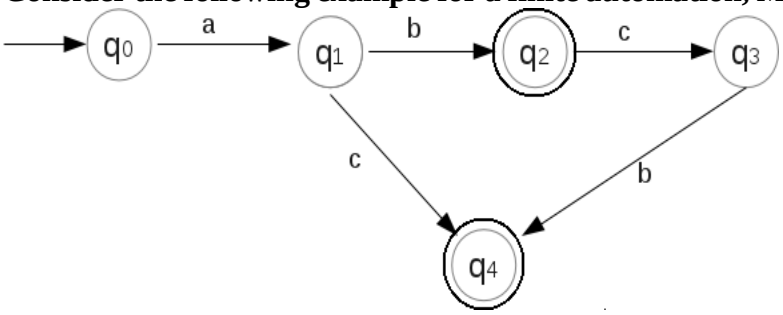
Formal Definition of Finite Automata:

A finite automaton can be represented formally by a 5-tuple:

{Q, Σ, q_0, F, δ}, where:

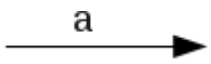
Q: is non empty Finite set of states (q_0, q_1, q_2,….)
Σ: is non empty finite Set of input symbols, called the alphabet (a, b, c, d ….)
q_0: Initial state/ is the start state i.e. the state of the automaton before any input has been processed, where $q_0 \in Q$.
F: accepting Set of states called accept states or final states of Q (i.e. F⊆Q)
δ: Transition function that is, δ: Q × Σ → Q

Consider the following example for a finite automation, M.

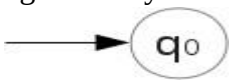


In the above finite automation, M,

1. Q = {q0, q1, q2, q3, q4}
   Q is the set of states in the finite automata. In the above finite automata, q0, q1, q2, q3, q4 are the states. The states are shown in circles.
2. Σ = a, b, c
   Σ is the set of input symbols in the finite automata. In the above, a, b, c are the input symbols. The arrows are labelled with input symbols.



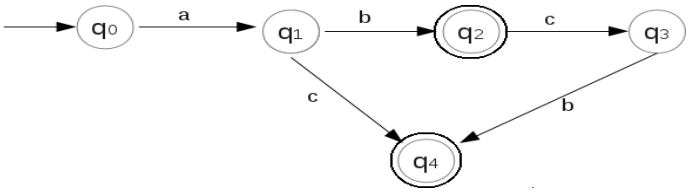
3. $q_0 = \{q_0\}$
   q0 is the start state. For our study, finite automata contain only one start state.A state with arrow from free space (not coming from any other state) is designated as start state.



4. δ describes the operation of finite automation. δ is the transition function. For example,
   $\delta(q_0, a) = q_1$
   This means, given the current state q_0 and the input symbol, a, the finite automation moves (transits) to state q_1.
   Similarly, $\delta(q1, b) = q2$
   $\delta(q2, c) = q3$
   $\delta(q1, c) = q4$
   $\delta(q3, b) = q4$
5. $F = \{q_2, q_4\}$
   F is the set of final states or accepting states. In the above finite automation, $q_2$and q_4 are the final states. They are shown enclosed in double circles.

Transition Diagrams and Transition Tables

Consider the following finite automation, M.



The above representation of finite automation is called a transition diagram.

A finite automata can also be represented as a transition table. This is shown below:

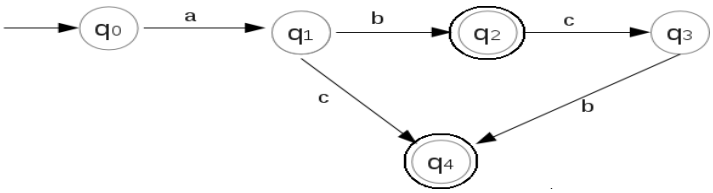
	Input Symbol		
Current State	a	b	c
$\longrightarrow q_0$	q_1	φ	φ
q_1	φ	q_2	q_4
$*q_2$	φ	φ	q_3
q_3	φ	q_4	φ
$*q_4$	φ	φ	φ

$\longrightarrow q_0$ denotes that q_0 is the start state.

$*$ before q_2 and q_4 indicates that q_2 and q_4 are the final states.

Language Acceptibility by Finite Automata String Processing by Finite Automation:

Example 1: Consider the following finite automation, M.



Check whether the string **abcb** is accepted by the above finite automation.
In the above, q_0 is the start state. the string abcb has the first symbol, a. So,

$\delta(q_0, \underline{a}bcb) = q_1$

$\delta(q_1, a\underline{b}cb) = q_2$

$\delta(q_2, ab\underline{c}b) = q_3$

$\delta(q_3, abc\underline{b}) = q_4$

Thus, $q_0 \overset{a}{\Rightarrow} q_1 \overset{b}{\Rightarrow} q_2 \overset{c}{\Rightarrow} q_3 \overset{b}{\Rightarrow} q_4$

Thus, after processing the string abcb, we reached the state, q_4. Here q_4 is a final state. So the string abcb is accepted by the finite automation.

Example2: check whether the string abc is accepted by above finite automata M.
On processing the string, beginning from the start state, q_0,

$\delta(q_0, \underline{a}bc) = q_1$
$\delta(q_1, a\underline{b}c) = q_2$
$\delta(q_2, ab\underline{c}) = q_3$

Here M halts at q_3. But q_3 is not a final state. Hence the string abc is rejected by the above finite automation, M.

Example 3: Check whether the string abca is accepted by the finite

automation, M. Beginning from the start state, q_0,

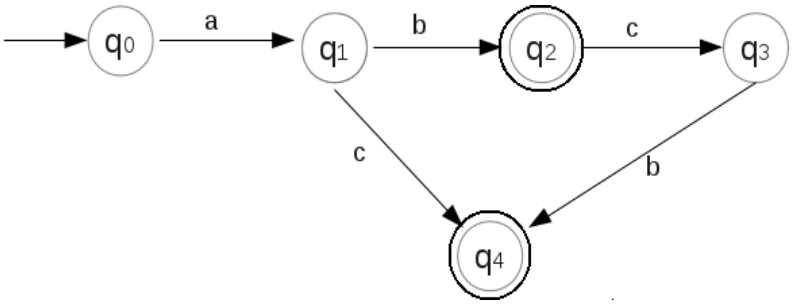
$\delta(q_0, \underline{a}bca) = q_1$
$\delta(q1, abca) = q2$
$\delta(q2, abca) = q3$
$\delta(q3, abca) =$

Here for current state, q_3, and for input symbol, a, there is no transition. This means M is unable to process the string abca completely. So the string abca is rejected by M.

Language of Finite Automation:

In the above, finite automation is used to check the validity of a string. A language is a finite set of strings. So we can use finite automation to check whether a string belongs to a language.

Consider the following finite automation,



Let L be the language corresponding to the above finite automation, M. Then a string belongs to the language L only if it is accepted by the finite automation, M.

We can say that,

- ✓ The string abcb belongs to the language, L.
- ✓ The string abc does not belong to the language, L.
- ✓ The string abca does not belong to the language,
- ✓ L. The string ab belongs to the language, L.
- ✓ The string ac belongs to the language, L.
- ✓ The string abcbc does not belong to the language, L

Types of Finite Automata

There are two types of finite automata:

1. Deterministic Finite Automata (DFA)
2. Non-Deterministic Finite Automata (NFA)

1. Deterministic Finite Automata (DFA)

A DFA is defined as an abstract mathematical concept, but due to the deterministic nature of a DFA, it is implementable in hardware and software for solving various specific problems.
DFAs can be built from nondeterministic finite automata through the power set construction.

A DFA is represented as {Q, Σ, q_0, F, δ}. In DFA, for each input symbol, the machine transitions to one and only one state. DFA does not allow any null transitions, meaning every state must have a transition defined for every input symbol.

Formal definition:

A deterministic finite automaton M is a 5-tuple, (Q, Σ, δ, q0, F), consisting of \

Q: set of all finite states.
Σ: set of input symbols. (Symbols which machine takes as input)
q_0 : Initial state. (Starting state of a machine) ($q_0 \in Q$)
δ : Transition Function, defined as δ : Q X Σ → Q.
F: set of final state / a set of accept states (F ⊆ Q)

Example:Construct a DFA that accepts all strings ending with 'a'.

Given:

Σ = {a, b},
Q = {q0, q1},
F = {q1}

State\Symbol	a	b
q0	q1	q0
q1	q1	q0

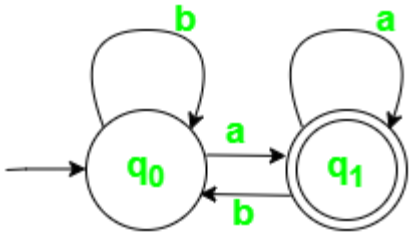
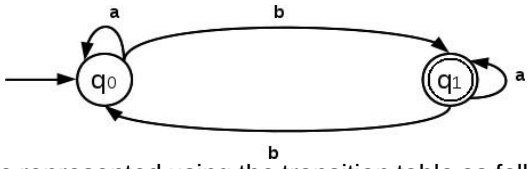


Fig 1. State Transition Diagram for DFA with Σ = {a, b}

In this example, if the string ends in 'a', the machine reaches state q1, which is an accepting state.

Example :The following shows a DFA,



Above transition diagram can be represented using the transition table as follows:

	Input Symbol	
Current State	a	b
$\longrightarrow q_0$	q_0	q_1
$*q_1$	q_1	q_0

Check whether the string aaba is accepted by the above DFA.
Beginning from the start state, q_0,

$\delta(q0, aaba) = q0$
$\delta(q0, aaba) = q0$
$\delta(q0, aaba) = q1$
$\delta(q1, aaba) = q1$

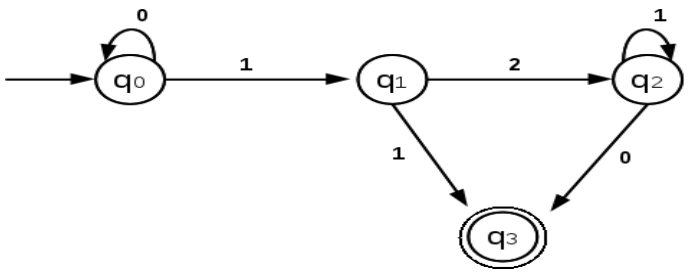
After processing all the characters in the input string, final state $q_1$is reached. So the string aaba is accepted by the above finite automation.

Also Check whether the string aabba is accepted by the above DFA.

$\delta(q_0, \underline{a}abba) = q_0$
$\delta(q_0, a\underline{a}bba) = q_0$
$\delta(q_0, aa\underline{b}ba) = q_1$
$\delta(q_1, aab\underline{b}a) = q_0$
$\delta(q_0, aabb\underline{a}) = q_0$

After processing all the symbols in the input string, aabba, the state q_0 is reached. But q_0 is not a final state. So the DFA rejects the string aabba.

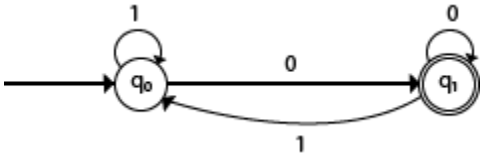
Consider the following finite automation,



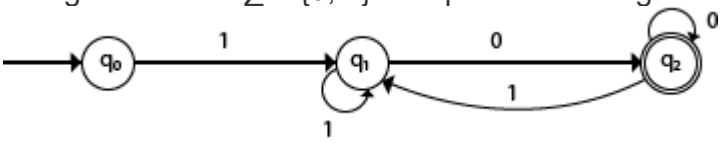
In the above, input symbols are 0, 1 and 2. From every state, there is only one transition for an input symbol.
From state, q_0, for symbol, 0, there is only one transition, that is to state q_0 itself.
From state, q_0, for symbol, 1, there is only one transition, that is to state q_1.
From state, q_1, for symbol, 1, there is only one transition, that is to state q_3, and so on.
Thus all the moves of the above finite automation can be determined uniquely by the current state and input symbol.
Such a finite automation is called Deterministic finite Automation (DFA). *Thus in a DFA, at every step, next move is uniquely determined. No ε transitions are present in a DFA.*

Workout Examples & Exercises of DFA:

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



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



Note that if the input ends with 0, it will be in the final state

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

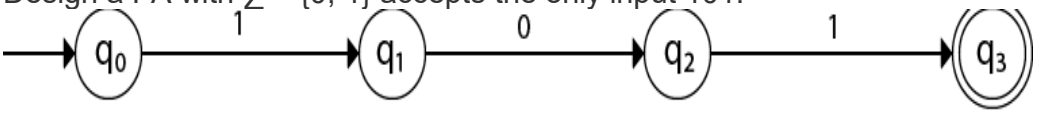
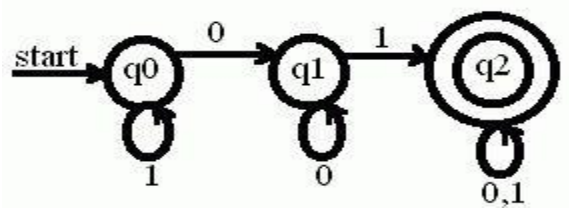


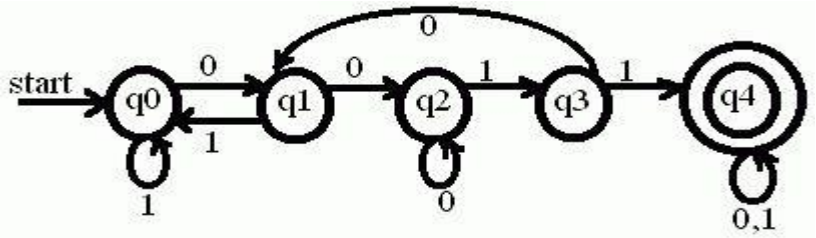
Fig: FA

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.

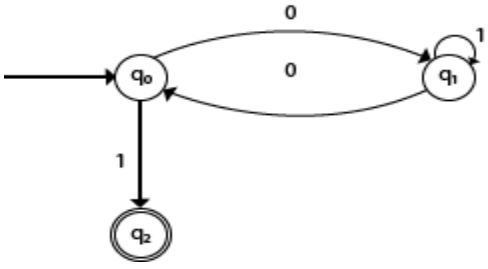
4. Construct a DFA to accept a string containing a zero followed by a one



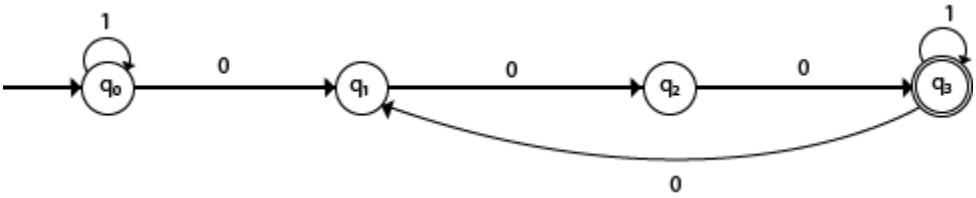
5. Construct a DFA to accept a string containing two consecutive zeroes followed by two consecutive ones.



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

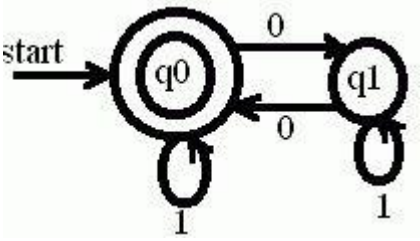


7. Design FA with ∑ = {0, 1} accepts the set of all strings with three consecutive 0's.

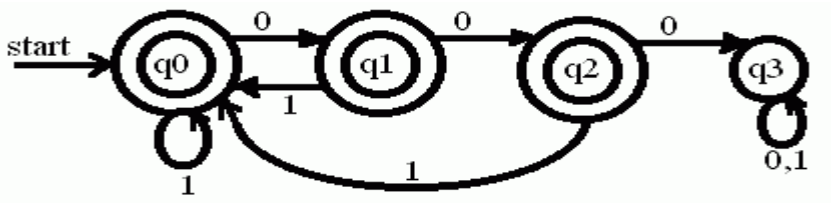


The strings that will be generated for this particular languages are 000, 0001, 1000, 10001, ... in which 0 always appears in a clump of 3.

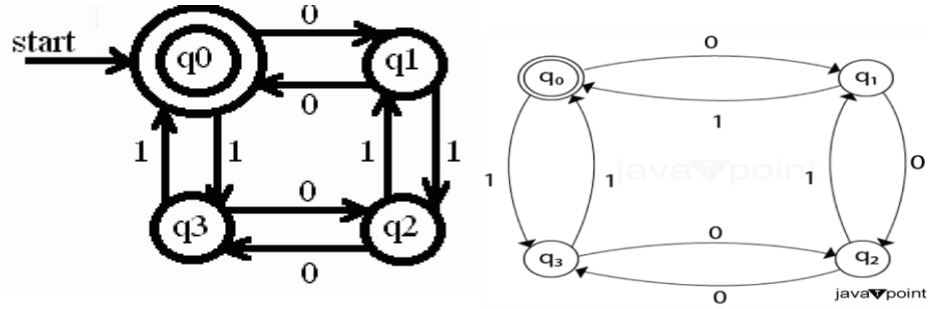
8. Construct a DFA to accept a string containing even number of zeroes and any number of ones.



Q.4. Construct a DFA to accept all strings which do not contain three consecutive zeroes.



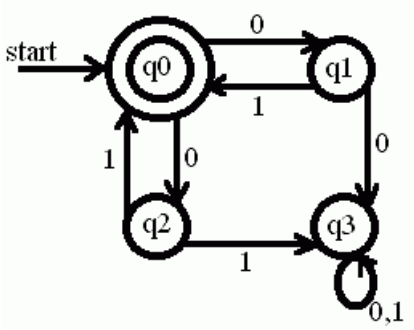
Q.5. Construct a DFA to accept all strings containing even number of zeroes and even number of ones.



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.

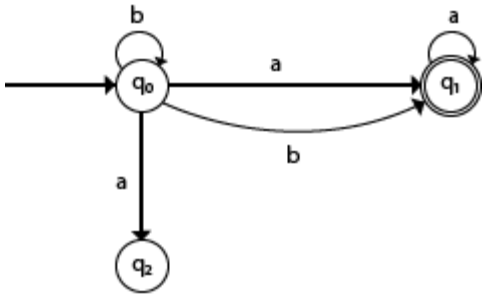
Q.6. Construct a DFA to accept all strings (0+1)* with an equal number of 0's & 1's such that each prefix has at most one more zero than ones and at most one more one than zeroes



2) Non-Deterministic Finite Automata (NFA)

- In the automata theory, a non-deterministic finite automata (NFA) is a finite state machine where from each state and a given input symbol the automaton may jump into several possible next states.
- 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.
- Although the DFA and NFA have distinct definitions, a NFA can be translated to equivalent DFA using power set construction, i.e., the constructed DFA and the NFA recognize the same formal language.
- Non-deterministic finite state machines are generalized by probabilistic automata, which assign a probability to each state transition.
- NFA is defined in the same way as DFA but with the following two exceptions:
  - It can transition to multiple next states for the same input.
  - It allows null (ϵ) moves, where the machine can change states without consuming any input.

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

An *NFA* is represented formally by a 5-tuple, $(Q, \Sigma, \Delta, q_0, F)$, consisting of

- a finite set of states Q
- a finite set of input symbols Σ
- a transition relation $\Delta : Q \times \Sigma \rightarrow 2^Q$. (Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
- an *initial* (or *start*) state $q_0 \in Q$
- a set of states F distinguished as *accepting* (or *final*) *states* $F \subseteq Q$.

Example 1:

Q = {q0, q1, q2}
∑ = {0, 1}
q0 = {q0}
F = {q2}

Transition diagram:

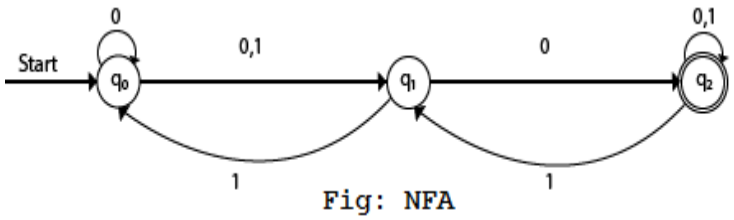


Fig: NFA

Solution:

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

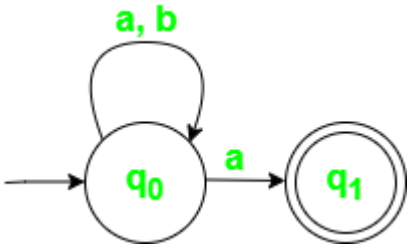
In the above diagram, we can see that when the current state is q0, on input 0, the next state will be q0 or q1, and on 1 input the next state will be q1. When the current state is q1, on input 0 the next state will be q2 and on 1 input, the next state will be q0. When the current state is q2, on 0 input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

Example2: Construct an NFA that accepts strings ending in 'a'.

Given:

Σ = {a, b},
Q = {q0, q1},
F = {q1}

Fig 2. State Transition Diagram for NFA with Σ = {a, b}



State Transition Table for above Automaton,

State\Symbol	a	b
q0	{q0,q1}	q0
q1	φ	φ

In an NFA, if any transition leads to an accepting state, the string is accepted.

Example 3: NFA with ∑ = {0, 1} accepts all strings with 01.

Solution:

Transition Table:

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

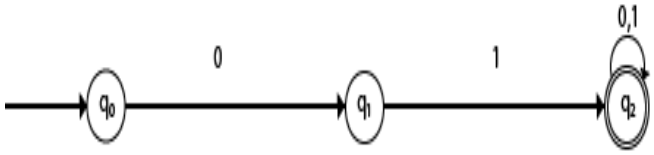


Fig: NFA

Example 4: NFA with ∑ = {0, 1} and accept all string of length at least 2.

Solution:

Transition Table:

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

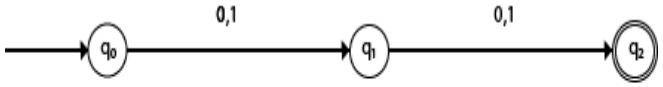
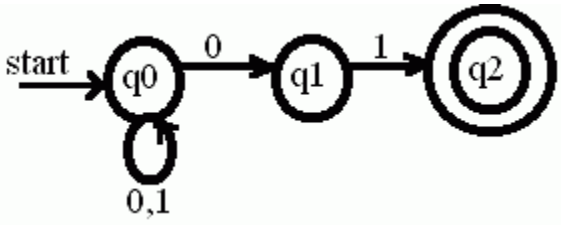


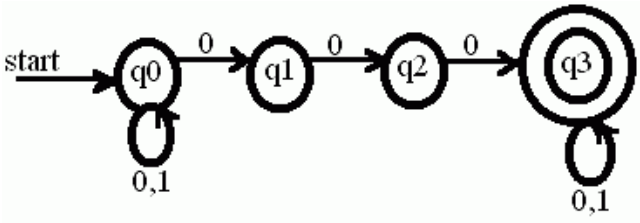
Fig: NFA

Examples and Exercises of NFA

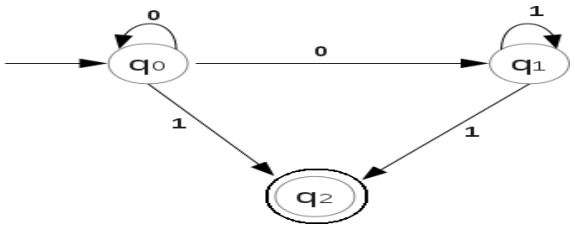
1. Construct an NFA to accept all strings terminating in 01



2. Construct an NFA to accept those strings containing three consecutive zeroes.

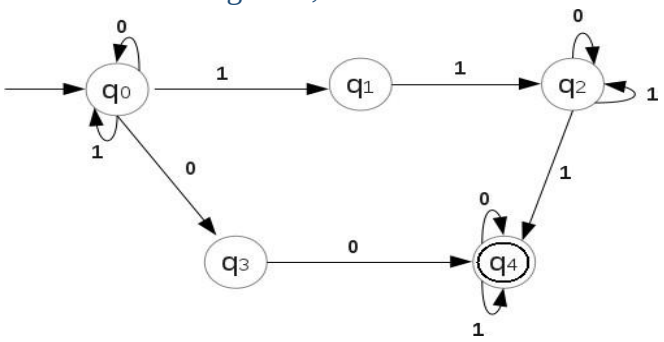


Consider the following finite automation, M,



- In the above, from state q_0 with input symbol 0, there are two possible next states. The next state can be either q_0 or q_1.
- Also from state q_1, on input symbol, 1, there are two possible next states. The next state can be either q_1 or q_2.
- Thus some moves of the finite automation cannot be detrmined uniquely by the input symbol and the current state.
- Such finite automation are called Non-deterministic Finite automata (NFA or NDFA).

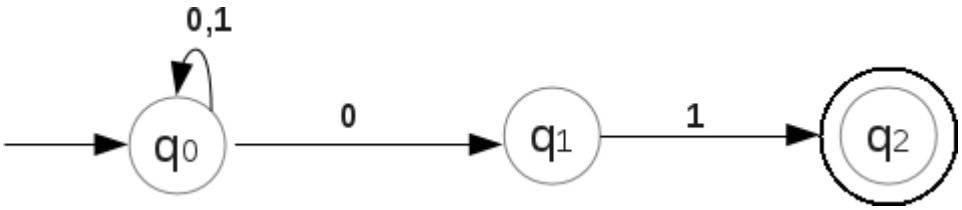
Example 1:Consider the following NFA,



Check whether the string 0100 is accepted by the above NFA. Beginning from the start symbol, q_0,

$\delta(q_0, \underline{0}100) = q_0$
$\delta(q_0, 0\underline{1}00) = q_0$
$\delta(q_0, 01\underline{0}0) = q_3$
$\delta(q_3, 010\underline{0}) = q_4$

q_4 is a final state. So the string 0100 is accepted by the above finite automation.

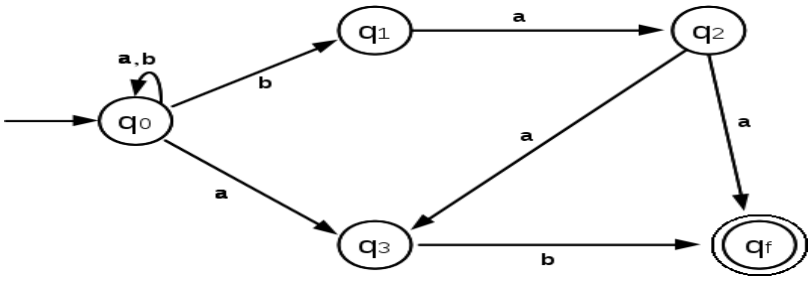


Example 2:Consider the following NFA,

Check whether the string 01101 is accepted by the above NFA.

$\delta(q_0, \underline{0}1101) = q_0$
$\delta(q_0, 0\underline{1}101) = q_0$
$\delta(q_0, 01\underline{1}01) = q_0$
$\delta(q_0, 011\underline{0}1) = q_1$
$\delta(q_1, 0110\underline{1}) = q_2$

q_2 is a final state. So the string 01101 is accepted by the above NFA.



Check whether the string abbaa is accepted by the above NFA.

$\delta(q_0, \underline{a}bbaa) = q_0$
$\delta(q_0, a\underline{b}baa) = q_0$
$\delta(q_0, ab\underline{b}aa) = q_1$
$\delta(q_1, abb\underline{a}a) = q_2$
$\delta(q_1, abba\underline{a}) = q_f$

q_f is a final state. So the string abbaa is accepted by the above NFA.

DFA vs NDFA

The following table lists the differences between DFA and NDFA.

DFA	NDFA
The transition from a state is to a single particular next state for each input symbol. Hence it is called *deterministic*.	The transition from a state can be to multiple next states for each input symbol. Hence it is called *non-deterministic*.
Empty string transitions are not seen in DFA.	NDFA permits empty string transitions.
Backtracking is allowed in DFA	In NDFA, backtracking is not always possible.
Requires more space.	Requires less space.
A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state.