INDEX

S.No	UNIT – I: Finite Automata	Page No
1	Why Study Automata Theory?	2
2	The Central Concepts of Automata Theory	3
3	Automation	6
4	Finite Automation	6
5	Transition Systems, Acceptance of a String by a Finite Automation,	9
6	DFA, Design of DFAs,	11
7	NFA, Design of NFA	12
8	Equivalence of DFA and NFA	13
9	Conversion of NFA into DFA	13
10	Finite Automata with E-Transition	14
11	Minimization of Finite Automata	15
12	Mealy and Moore Machines	16
13	Applications and Limitation of Finite Automata.	
14	Differences between DFA and NFA	19
15	Differences between Mealy and Moore Machines	19

Why Study Automata Theory?:

There are several reasons why the study of automata and complexity is an important part of the core of Computer Science.

Computer science can be divided into two main branches.

1. "Applied Computer Science" which is concerned with the design and implementation of computer systems, with a special focus on software.
2. "Theoretical Computer Science" or "Theory of computation" which is concerned with mathematical study of computation.

The central areas of theory of computation are

1. Automata theory
2. Computational theory
3. Complexity theory

Computational theory and Complexity theory:

The theories of computability and complexity are closely related. In complexity theory, the objective is to classify problems as easy ones and hard ones; whereas in computability theory, the classification of problems is by those that are solvable and those that are not.

Computational Problem:

"Computational Problem" means any problem that can be modeled to be solved by a computer.

Computation:

"Computation" can be defined as finding a solution to a problem from the given inputs by means of an algorithm.

Automata theory:

It deals with the formal definitions and properties of mathematical models of computation.

These models play a role in several applied areas of computer science. One model, called the *finite automaton*, is used in text processing, compilers, and hardware design. Another model, called the *context-free grammar*, is used in programming languages and artificial intelligence.

Finite automaton is a useful model for

1. Software for designing and checking the behavior of digital circuits (hardware design).
2. The "lexical analyzer" of a typical compiler, that is, the compiler component that breaks the input text into logical units, such as identifiers, keywords, and punctuation (compilers).
3. Software for scanning large bodies of text, such as collections of Web pages, to find occurrences of words, phrases, or other patterns (text processing).

The theories of computability and complexity require a precise definition of a *computer*. Automata theory allows practice with formal definitions of computation.

The Central Concepts of Automata Theory:

Symbol: Symbol is any character which is meaningful or meaningless.
Example: A,B,C,….,Z
a,b,c,…z
0,1,2,…9
∑, λ, δ, π…

Alphabet: An alphabet is a finite, and non empty set of symbols. It is denoted by ∑. Common alphabets include:
Example: 1. ∑= {0, 1}, is the binary alphabet.
2. ∑ = {a, b,…, z}, is lower-case English alphabet.
3. ∑ = {0,1,2,…9}, is decimal number alphabet.

String: A string is a finite sequence of symbols from some alphabet.
Example: ∑= {0, 1}, is the binary alphabet.
Strings are 00,01,10,11…

Empty String: The empty string is the string with zero occurrences of symbols. It is denoted by ε.

Length of a string: the length of a string is "the number of symbols" in the string. length of a string w is denoted by |w|. length of empty string is 0.
Example: w=sri, |w|=3

Powers of an alphabet (set of strings of length k): If ∑ is an alphabet, then set of all strings of a certain length k are denoted by $\sum^k$.
Example: if ∑= {0, 1}, then $\sum^0 = \{\varepsilon\}$ – strings of length 0,
$\sum^1=\{0,1\}$– strings of length 1,
$\sum^2=\{00,01,10,11\}$– strings of length 2,…

Prefix of a string: prefix of a string is formed by removing zero or more tailing symbols of the string. For a string of length n, number of prefixes= n+1.
Example: w=sri, prefix=sri,sr,s,ε.

Proper prefix: prefix of a string other than itself is called proper prefix.
Example: w=sri, proper prefix=sr,s, ε.

suffix of a string: prefix of a string is formed by removing zero or more leading symbols of the string. For a string of length n, number of suffixes= n+1.
Example: w=sri, suffix=sri,ri,i,ε.

Proper suffix: prefix of a string other than itself is called proper prefix.
Example: w=sri, proper suffix= ri,i,ε.

substring of a string: substring of a string is formed by removing either prefix or suffix or both from a string excluding ε. For a string of length n, number of substrings= n(n+1)/2.
Example: w=sri, substring=sri, sr,s,ri,i,r

Language: language is a collection of strings formed from a particular alphabet.
Example: if ∑= {0, 1}, then the language of all strings consisting of n 0' s followed by n l' s, for some n≥0 is L= {ε, 01,0011,000111,. .}.

Operations on strings:

1. Concatenation of a string
2. Kleene closure of a string

3. Positive closure of a string
4. Reverse of a string

Concatenation of a string: concatenation of a string is obtained by appending second string at the end of first string. If w1 and w2 are strings, then w1.w2 or w1w2 denotes the concatenation of w1 and w2.
Example: w1=sri   w2=lakshmi   then w1.w2=srilakshmi

length of concatenated string is |w1.w2|=|w1|+|w2|.
Example: w1=sri   w2=lakshmi   w1.w2=srilakshmi |w1.w2|=|w1|+|w2|=3+7=10

Kleene closure of a string: Set of all strings over an alphabet ∑ is known as kleene closure. It is denoted by $\sum^*$.

$$\sum^* = \sum^0 \cup \sum^1 \cup \sum^2 \cup \ldots$$

Example: if ∑= {0, 1}, then $\sum^0 = \{\varepsilon\}$ – strings of length 0,
$\sum^1=\{0,1\}$– strings of length 1,
$\sum^2=\{00,01,10,11\}$– strings of length 2,…

$\sum^* = \sum^0 \cup \sum^1 \cup \sum^2 \cup \ldots =\{ \varepsilon, 0, 1, 00 ,01, 10, 11, \ldots\}$

Positive closure of a string: Set of all strings over an alphabet ∑ excluding the empty string is known as positive closure. It is denoted by $\sum^+$.

$$\sum^+ = \sum^1 \cup \sum^2 \cup \sum^3 \cup \ldots \text{ or } \sum^+ = \sum^* - \{\varepsilon\}$$

Example: if ∑= {0, 1}, then $\sum^0 = \{\varepsilon\}$ – strings of length 0,
$\sum^1=\{0,1\}$– strings of length 1,
$\sum^2=\{00,01,10,11\}$– strings of length 2,…

$\sum^* = \sum^0 \cup \sum^1 \cup \sum^2 \cup \ldots =\{ \varepsilon, 0, 1, 00 ,01, 10, 11, \ldots\}$

$\sum^+ = \sum^* - \{\varepsilon\} = \{ 0, 1, 00 ,01, 10, 11, \ldots\}$

Reverse of a string: Reverse of a string is obtained by reading the string from back to front. If w is a string, then reverse of a string w is denoted by w^R.
Example: w=sri   then w^R=irs

<u>Operations on languages:</u>

1. Union of a language
2. Intersection of a language
3. Difference of a language
4. Complement of a language
5. Concatenation of a language
6. Kleene closure of a language
7. Positive closure of a language

Union of a language: Union of two languages is defined as collection of strings from L1 as well as from L2. It is denoted as $L_1 \cup L_2$.

$$\mathbf{L_1 \cup L_2=\{w \mid w \text{ in } L_1 \text{ or } w \text{ in } L_2\}}$$

Example: if L_1= {0,1} and L_2= {00,11} then $L_1 \cup L_2$={0,1,00,11}

Intersection of a language: Intersection of two languages is defined as collection of strings common in both L1 as well as L2. It is denoted as $L_1 \cap L_2$.

$$\mathbf{L_1 \cap L_2=\{w \mid w \text{ in } L_1 \text{ and } w \text{ in } L_2\}}$$

Example: if L_1= {00,1} and L_2= {00,11} then $L_1 \cap L_2$={00}

Difference of a language: Difference of two languages is defined as collection of strings from L1 that are not in L2. It is denoted as $L_1 - L_2$.

$$\mathbf{L_1 - L_2 = \{w \mid w \text{ in } L_1 \text{ and not in } L_2\}}$$

Example: if $L_1 = \{00,1\}$ and $L_2 = \{00,11\}$ then $L_1 \cap L_2 = \{1\}$

Complement of a language: Complement of a language is defined as language consisting of all the strings that are not in language L. It is denoted as L^C.

$$\mathbf{L^C = \sum^* - L}$$

Example: if L= {set of strings that starts with 010} over $\sum = \{0, 1\}$
then $L^C = \sum^* - L$ ={set of all strings} −{set of strings that starts with 010}
= { of strings that does not starts with 010}

Concatenation of a language: Concatenation of a two languages is obtained by appending every string in second language at the end of every string in first language. If L1 and L2 are two languages, then $L_1.L_2$ or L_1L_2 denotes the concatenation of L1 and L2.

$$\mathbf{L_1.L_2 = \{w \mid w1 \text{ in } L_1 \text{ and } w2 \text{ in } L_2\}}$$

Example: if $\sum = \{0, 1\}$, then the language of all strings consisting of n 0' s followed by n l' s, for some $n \geq 0$ is $L_1 = \{\varepsilon, 01, 0011, 000111, ..\}$.

if $\sum = \{0, 1\}$, then the language of all strings consisting of an equal number of 0's and l's is $L_2 = \{\varepsilon, 01, 10, 0011, 0101, 1001, ...\}$.

$L_1.L_2 = \{ \boldsymbol{\varepsilon}.\ \varepsilon,\ \boldsymbol{\varepsilon}.\ 01,\ \boldsymbol{\varepsilon}.\ 10,\ \boldsymbol{\varepsilon}.\ 0011,\ \boldsymbol{\varepsilon}.\ 0101,\ \boldsymbol{\varepsilon}.\ 1001, \ldots, \mathbf{01}10, \mathbf{01}0011, \mathbf{01}0101, \ldots\}$
$= \{\varepsilon, 01, 10, 0011, 0101, 1001, \ldots, 0101, 010011, 010101\ldots\}$

Kleene closure of a language: If L is a language, Set of all strings formed by concatenation of zero or more strings of the language is known as kleene closure. It is denoted by L^*.

$L^i = L^{i-1}.L,\ i \geq 1$
$\mathbf{L^* = L^0 \cup L^1 \cup L^2 \cup ...}$

Example: if $\sum = \{0, 1\}$, then language containing strings of length 1 is $L = \{0, 1\}$,
then $L^0 = \{\varepsilon\}$
$L^1 = L^0.L = \{\varepsilon\}\{0,1\} = \{0,1\}$
$L^2 = L^1.L = \{0,1\}\{0,1\} = \{00,01,10,11\}$

$L^* = L^0 \cup L^1 \cup L^2 \cup ... = \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$

Positive closure of a language: If L is a language, Set of all strings formed by concatenation of one or more strings of the language is known as positive closure. It is denoted by L^+.

$\mathbf{L^+ = L^1 \cup L^2 \cup L^3 \cup \ldots}$ or $\mathbf{L^+ = L^* - \{\varepsilon\}}$

Example: if $\sum = \{0, 1\}$, }, then language containing strings of length 1 is $L = \{0, 1\}$,
then $L^0 = \{\varepsilon\}$
$L^1 = L^0.L = \{\varepsilon\}\{0,1\} = \{0,1\}$
$L^2 = L^1.L = \{0,1\}\{0,1\} = \{00,01,10,11\}$

$L^* = L^0 \cup L^1 \cup L^2 \cup ... = \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$

$L^+ = L^* - \{\varepsilon\} = \{0, 1, 00, 01, 10, 11, \ldots\}$

Automata:

Automata is plural for the term "Automaton". An Automaton is defined as a system where information is transmitted and used for performing some internal operations without direct participation of human.

Finite State Machine/ Finite Automata:

Definition:

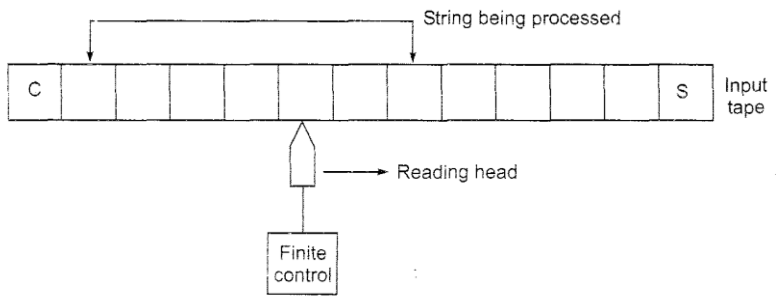
"Finite State Machine" is a model of computation which is used to simulate the behaviour of a machine or a system.

Model of Finite Automata:

It includes 1. Components of Finite State Machine
2. Elements of Finite State Machine
3. Working of Finite State Machine
4. Mathematical representation of Finite State Machine

1.Components of Finite State Machine:

Finite State Machine consists of 3 components. They are 1. Input tape
2. Read head and
3. Finite Control



Input tape:

It is used to store the input string that is to be processed by Finite State Machine. The input is finite and taken from a set of input alphabets ∑.

Read head:

It is used to read one symbol at a time from input tape and moves from left to right. Initially it points to the leftmost symbol on the tape and it is controlled by "Finite control".

Finite Control:

Finite control contains a set of states which are connected by transitions. In finite control, it is decided that "machine is in this state and it is getting this input, so it will go to this state".

2. Elements of Finite State Machine:

The main elements of Finite State Machine are 1. State
2. Rules

3. State transition
4. Input events

1. State:

This element defines the behaviour of the system and generate action to be performed. There are different types of states. They are 1. Start state
2. Next state
3. Accepting state
3. Dead state
4. Inaccessible state
5. Universal State
6. Existential State

Types of states:

1. **Start state:** State which is used to start processing of input string in finite state machine is called as "start state" or "initial state".
2. **Next state:** State which is defined by the transition function of finite state machine for current state q_i and input symbol x is called as "next state".
$$\delta(q_i, x) \rightarrow q_j$$
3. **Accepting state:** Once entire string is processed, state which leads to acceptance of string is called as "Accepting state" or "final state".
4. **Dead state:** A non final state of finite state machine whose transitions on every input symbol terminates on itself is called as "dead state".
$$q \notin F \; and \; \delta(q_i, \Sigma) \rightarrow q_i$$
5. **Inaccessible state:** State which can never be reached from initial state is called "inaccessible state" or "useless state".
6. **Equivalent state:** Two states are "equivalent" or "indistinguishable", if both produces final states or if both produces non final states on applying same input symbol.
$$\delta(q_i, x) \in F \rightarrow \delta(q_j, x) \in F$$
$$\delta(q_i, x) \in NF \rightarrow \delta(q_j, x) \in NF$$
7. **Distinguishable state:** Two states are "not equivalent" or "distinguishable", if both produces final states or if both produces non final states on applying same input symbol.
$$\delta(q_i, x) \in F \rightarrow \delta(q_j, x) \in NF$$
$$\delta(q_i, x) \in NF \rightarrow \delta(q_j, x) \in F$$

2. Rules:

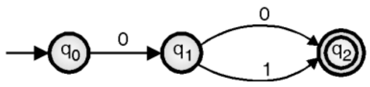
This element defines the conditions that are to be satisfied for enabling state transition.

3. State transition:

This element defines the change in state. i,e., moving from one state to another state is known as transition. Transitions are represented in 3 ways. They are 1. Transition diagram
2. Transition table
3. Transition function.

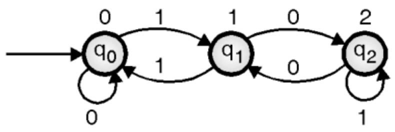
Types of transition:

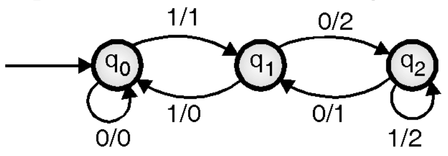
1. **Transition diagram/ Transition graph/ Transition system:**
   A diagrammatical representation of states and transitions is called as transition diagram. In transition diagram, circles are used to represent states and directed arrows are used to represent transitions between states.



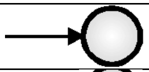
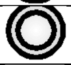
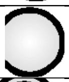
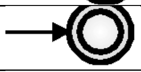
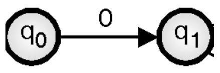
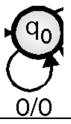
For a finite state machine, transition diagram or transition graph or a transition system is formally defined as a directed labeled graph where each vertex is a state, and directed edge is a transition between two states and edges are labeled with input/output.

Note: For DFA, NFA, and Moore machine, only input is labeled on each edge, where as in mealy machine, both input and output are labeled on each edge.





Representations:

Start state	
Final state	
Other states	
initial and final states are same	
transition between two states	
transition between two states with input and output	

2. **Transition table:** Tabular representation of states and transitions is called as transition table. In transition table, rows are used to represent states and columns are used to represent input symbols and entry in table represents the next state based on current state and input symbol.
   In this representation start state is marked by an arrow (→) and final state is marked by an asterisk (*) or enclosed within a single circle (①).
3. **Transition function:** Mathematical representation of states and transitions is called as transition function. It is denoted by δ. There are two types of transition function depending on number of arguments. They are 1. Direct transition function
   2. Indirect transition function.

   Direct transition function: When the input is a symbol, it is known as direct transition function. It is denoted by $\boldsymbol{\delta(q_i, x) = q_j}$ where
   q_i is used to represent current state
   x is used to represent input symbol and
   q_j represents the next state based on current state and input symbol.

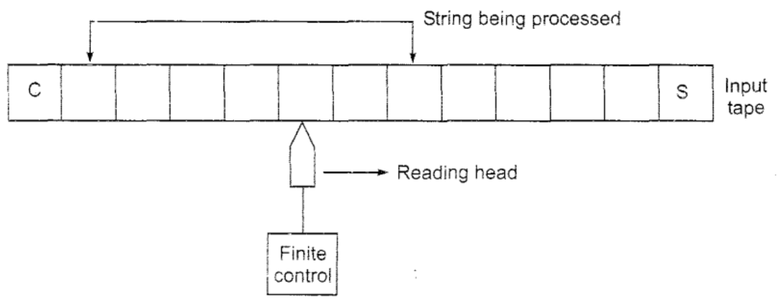
Note: $\boldsymbol{\delta(q_i,\ \varepsilon) = q_i}$, i.e state changes only on applying an input symbol.

Indirect transition function:
When the input is a string xw , it is known as indirect transition function. It is denoted by $\boldsymbol{\delta(q_i, xw) \vdash \delta(\delta(q_i, x), w)}$ where

q_i is used to represent current state
x is used to represent current input symbol and
$\delta(q_i, x)$ represents the next state based on q_i and x and
w represents the remaining input string.

4. Input events:
This element is used to evaluate rules and lead to transition. These may be generated internally or externally.

3. Working of finite state machine:



Computation begins in start state with input string in input tape. The read head scans the input from input tape and sends it to finite control.

Based on current state and input symbol of finite state machine, next state will be determined.

After processing the entire string, if the finite control enters into one of the final states, FSM declares that string is accepted and recognizes that string belongs to the given language., otherwise it declares that string doesn't belongs to the given language.

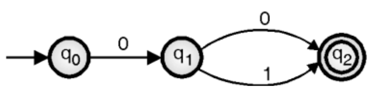
4. Mathematical representation of Finite State Machine/Finite Automata:

Finite Automata is formally defined as 5-tuple $\boldsymbol{M = (Q, \Sigma, \delta, q_0, F)}$ where

$M = Finite\ State\ Machine$
$Q - finite\ number\ of\ states$
$\Sigma - input\ alphabet\ or\ set\ of\ input\ symbols$
$\delta - transition\ function\ defined\ by\ \delta: QX\Sigma \rightarrow \mathrm{Q}$
$q_0 -\ start\ state\ q_0 \in Q$
$F - set\ of\ final\ states\ F \subseteq Q$

Transition System:

A transition system is formally defined as a directed labeled graph where each vertex is a state, and directed edge is a transition between two states and edges are labeled with input/output.



A transition system is a 5-tuple $(\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{F})$ where

$$\begin{aligned}&Q - finite\ number\ of\ states\\&\Sigma - input\ alphabet\ or\ set\ of\ input\ symbols\\&\delta - transition\ function\ defined\ by\ \delta: QX\Sigma^* XQ\\&q_0 - \ start\ state\ q_0 \in Q\\&F - set\ of\ final\ states\ F \subseteq Q\end{aligned}$$

i.e if $(q_0, w. q_2)$ is in $\boldsymbol{\delta}$. it means that the graph starts at the vertex q_0, goes along a set of edges, and reaches the vertex q_2 . The concatenation of the label of all the edges thus encountered is *w*.

Note: Every finite automaton $(Q, \Sigma, \delta, q_0, F)$ can be viewed as a transition system $(Q, \Sigma, \delta'\ Q_0, F)$ if we take $Q_0 = \{q_0\}$ and $\delta' = \{(q, w, \delta(q, w)) | q \in Q, w \in \Sigma^*\}$. But a transition system need not be a finite automaton. For example, a transition system may contain more than one initial state.

Acceptability of a string by finite automata:

Definition 3.3 A transition system accepts a string w in Σ^* if

(i) there exists a path which originates from some initial state, goes along the arrows, and terminates at some final state; and

(ii) the path value obtained by concatenation of all edge-labels of the path is equal to w.

i.e., A string w is accepted by finite automata $M = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{F})$ if $\delta(q_0, w) = q_f$ for some $q_f \in F$. This is basically acceptability of a string by the final state.

Consider the finite state machine whose transition function δ is given by Table 3.1 in the form of a transition table. Here, $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $F = \{q_0\}$. Give the entire sequence of states for the input string 110001.

TABLE 3.1 Transition Function Table for Example 3.5

State	Input 0	Input 1
$\rightarrow \boxed{q_0}$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Solution

$$\begin{aligned}\delta(q_0, \overset{\downarrow}{1}10101) &= \delta(q_1, \overset{\downarrow}{1}0101)\\&= \delta(q_0, \overset{\downarrow}{0}101)\\&= \delta(q_2, \overset{\downarrow}{1}01)\\&= \delta(q_3, \overset{\downarrow}{0}1)\\&= \delta(q_1, \overset{\downarrow}{1})\\&= \delta(q_0, \Lambda)\\&= q_0\end{aligned}$$

Hence,

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_0$$

The symbol $\downarrow$ indicates that the current input symbol is being processed by the machine.

Types of Finite Automata:

Finite Automata can be classified into 2 types. They are

1. Finite Automata without output (recognizers)
2. Finite Automata with output (tranducers)

Finite Automata without output are classified into 3 types. They are

1. Deterministic Finite Automata (DFA)
2. Non Deterministic Finite Automata (NFA)
3. Non Deterministic Finite Automata with Epsilon transitions (NFA-ε)
4. Minimized DFA

Finite Automata with output are classified into 2 types. They are

1. Moore Machine
2. Mealy Machine

Finite Automata without output (RECOGNIZERS):

1. DETERMINISTIC FINITE AUTOMATA (DFA)

Definition:

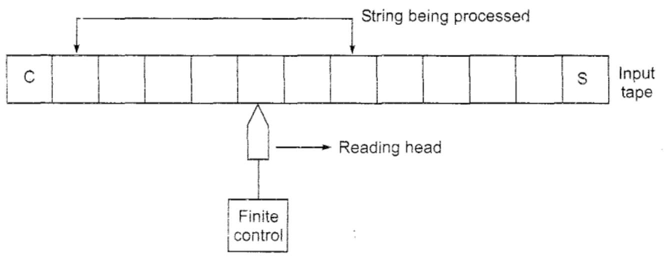
Deterministic Finite Automata can be defined as "it is a finite automata in which all the moves of a machine can be uniquely determined by using current state and input symbol. i.e., For the given input symbol, there will be only one transition from the current state.

Mathematical representation of DFA:

DFA is formally defined as 5-tuple $\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{F})$ where

$$M = Finite\ State\ Machine$$
$$Q - finite\ number\ of\ states$$
$$\Sigma - input\ alphabet\ or\ set\ of\ input\ symbols$$
$$\delta - transition\ function\ defined\ by\ \delta: QX\Sigma \to \mathrm{Q}$$
$$q_0 - \ start\ state\ q_0 \in Q$$
$$F - set\ of\ final\ states\ F \subseteq Q$$

Working of DFA:



Initially DFA is assumed to be in its start state q_0 with its read head on leftmost symbol of input string in input tape. The read head scans the input from input tape and sends it to finite control.

Based on current state and input symbol of finite state machine, next state will be determined by finite control. During each move, read head moves one position to the right.

After processing the entire string, if the DFA enters into one of the final states, the string is accepted, otherwise the string is rejected.

Design of DFAs:

1. Write the strings in the given language using the given input alphabet ∑.
2. Design DFA for the minimum string in the language.
3. At each state, for the input symbol which has no transition, put a loop over that particular state with that input symbol.
4. If the language is satisfied for that loop on that state, then check for next state. Otherwise, put the transition for the previous state with that input symbol.
5. If then also, it is not satisfied, then put the transition for the previous previous state with that input symbol.
6. If it is a start state, include a new state called as dead state.
7. Repeat the steps 2-5 until each and every state in DFA is having a transition with each and every input symbol.

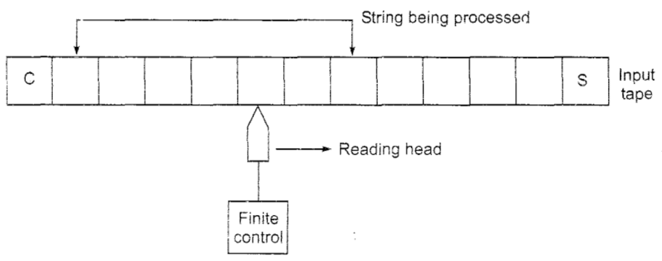
2. NON-DETERMINISTIC FINITE AUTOMATA (NDFA / NFA)

Definition:
Non Deterministic Finite Automata can be defined as "it is a finite automata in which some of the moves of a machine cannot be uniquely determined by using current state and input symbol. i.e., For the given input symbol, there will be more than one transition from the current state.
Mathematical representation of NFA:

NFA is formally defined as 5-tuple $\boldsymbol{M = (Q, \Sigma, \delta, q_0, F)}$ where

$M = Finite\ State\ Machine$
$Q - finite\ number\ of\ states$
$\Sigma - input\ alphabet\ or\ set\ of\ input\ symbols$
$\delta - transition\ function\ defined\ by\ \delta : QX\Sigma \rightarrow 2^{Q}, 2^{Q} \subseteq Q$
$q_0 -\ start\ state\ q_0 \in Q$
$F - set\ of\ final\ states\ F \subseteq Q$

Working of NFA:



Initially NFA is assumed to be in its start state q_0 with its read head on leftmost symbol of input string in input tape. The read head scans the input from input tape and sends it to finite control.

Based on current state and input symbol of finite state machine, several choices may exist for the next state. Then the machine chooses one of them and continues.

If the next input symbol matches, it continues. Otherwise, it chooses another way. During each move, read head moves one position to the right.

After processing the entire string, if the NFA enters into one of the final states, the string is accepted, otherwise the string is rejected.

Design of NFAs:

1. Write the strings in the given language using the given input alphabet ∑.
2. Design NFA for the minimum string in the language.
3. Put the transition with the input symbol wherever necessary, such that all strings of given language are accepted by NFA.

Equivalence of DFA and DFA:

Two DFAs M1 and M2 are said to be equivalent, if the language accepted by 2 DFAs is same irrespective of number of states. i.e., if L(M1)=L(M2), then M1 and M2 are equivalent.

Procedure:

1. Draw transition table format with input symbols and pair of states. Each pair of states contains state from M1 and state from M2.
2. Initially take start states of M1 and M2 as a pair and generate new pair of states with the input symbols of DFA.
3. During the process of generating new pair of states, if a pair of type (F,NF) or (NF,F) is generated, then immediately stop the process and conclude that given DFAs are not equivalent.
4. During the process of generating pairs of states, if a pair of type (F,F) or (NF,NF) is generated, then repeat the process until no new pair of states are found and conclude that given DFAs are equivalent.

Equivalence of DFA and NFA/ Conversion from NFA to DFA:

1. Draw the transition table for the given NFA.
2. Draw the transition table format for the required DFA with the input symbols in the given DFA having only initial state.
3. The start state of DFA is the start state of given NFA.
4. If over an input symbol in DFA, a new state is obtained, then make that state as the stat in DFA.
5. Repeat step4 until no new state is found.
6. The final states of DFA are those states that contain the final states of NFA.

3. NFA WITH EPSILON TRANSITIONS (NFA-ε / ε-NFA)

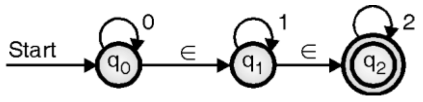
Definition:
Non Deterministic Finite Automata with epsilon transitions can be defined as "it is a non deterministic finite automata in which some of the transitions of a machine can be made without reading any input symbol. i.e., without any input symbol (ε), there will be a transition from the current state.

ε - transitions:

ε - transitions are the transitions by using which NFA can change its state without reading any input symbol.

ε - closure:

ε - closure of a state is set of all the states reachable from that state by using only ε - transitions including itself.

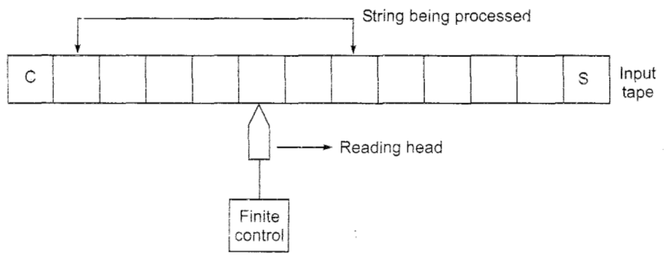
ε-closure(q0)={q0,q1,q2} for 

Mathematical representation of NFA-ε:

NFA-ε is formally defined as 5-tuple $\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{F})$ where

$$\begin{aligned}
&M = Finite\ State\ Machine\\
&Q - finite\ number\ of\ states\\
&\Sigma - input\ alphabet\ or\ set\ of\ input\ symbols\\
&\delta - transition\ function\ defined\ by\ \delta: QX\Sigma \cup \{\varepsilon\} \rightarrow 2^{Q}, 2^{Q} \subseteq Q\\
&q_0 - \ start\ state\ q_0 \in Q\\
&F - set\ of\ final\ states\ F \subseteq Q
\end{aligned}$$

Working of NFA-ε:



Initially NFA-ε is assumed to be in its start state q_0 with its read head on leftmost symbol of input string in input tape. The read head scans the input from input tape and sends it to finite control.

Based on current state and input symbol of finite state machine, several choices may exist for the next state. Then the machine chooses one of them and continues.

If there is no input symbol, then using epsilon transition, it moves to next state and continues. If the next input symbol matches, it continues. Otherwise, it chooses another way. During each move, read head moves one position to the right.

After processing the entire string, if the NFA-ε enters into one of the final states, the string is accepted, otherwise the string is rejected.

Design of NFA-ε:

1. Write the strings in the given language using the given input alphabet ∑.
2. Design NFA- ε for the minimum string in the language.
3. Put the transition with the ε symbol wherever necessary, such that all strings of given language are accepted by NFA-ε.

Equivalence of NFA-ε and NFA/ Conversion from NFA-ε to NFA:

1. Find ε - closures of all the states in given NFA- ε.

2. Find transitions for NFA $\boldsymbol{\delta'}$ with every state and every input symbol of given NFA- ε using $\boldsymbol{\delta'(q_p, i) = \varepsilon - closure\ (\delta(\varepsilon - closure(\ q_p), i)}$
3. Draw the transition diagram based on transitions obtained.
4. The start state of NFA is the start state of given NFA- ε.
5. The final states of NFA are the final states of NFA- ε.

4. Minimized Deterministic Finite Automata:

Definition:

DFA with number of states as minimum as possible is called as "Minimized Finite Automata".

Procedure to minimize DFA:

There are 2 ways to minimize DFA. They are 1.Equivalence method
2. Myhill-Nerode theorem

1. Equivalence method:

Definitions:

k-equivalent:

Two states are said to be "k-equivalent" (k≥0) , if both produces final states or if both produces non final states on applying same input of length k or less.

K=0, 0-equivalent:

The only string with length "0" is ε. If ε is applied on any state, it cannot reach a final state, but if ε is applied on any state, it reaches a final state, since the ε- closure of any state includes the state itself.

Therefore at level "0", behaviour of final states differ from non final states. so, partition the states into 2 groups, final and non final. It is denoted by π_0.

K=1, 1-equivalent:

It is denoted by π_1. Equivalence at level 1 can exist if and only if there is equivalence at level "0". Therefore here equivalence is to be checked among the states that were equivalent in level 0 (subsets of π_0). States having the same behavior will be grouped.

Procedure:

1. Write the set of states Q in the given DFA.
2. Find 0-equivalence (π_0): Partition Q into 2 groups, I - Final states 'F' and II - Non-Final states 'NF'.
$$\pi_0 = (I, II) = (F, Q - F)$$
3. Find 1-equivalence (π_1): check equivalence on subsets (groups) of π_0 and perform partition on subsets of π_0 F, NF to find new groups.
$$\pi_1 = (I, III, IV..)$$
    - If the group of π_0 contains a single state, then further it cannot be partitioned.

- For each group, write the transition table with input symbols on row side and states on column side.
- Rewrite the transition table by replacing the transition with its group number that it falls into.
- For any input symbol, if transition is made to same subset of π_0, then it cannot be partitioned.
- For any input symbol, if transition is made to different subset of π_0, then subset is again partitioned.

4. Similarly find 2-equivalence (π_2) and so on until $\pi_{n-1} = \pi_n$.
5. For any input symbol, if no partitions are made on subsets of π_n, then that states are indistinguishable and these groups are the states of modified DFA.
6. Draw transition table for the minimized DFA with modified states on row side and input symbols on column side and generate the transitions.
7. Draw transition diagram based on transition table obtained in step 6. This is the required minimized DFA.
8. Remove any useless states to obtain optimized DFA.

2. Myhill-Nerode theorem:

1. Draw transition table for given DFA.
2. Write the set of states Q in the given DFA and partition Q into 2 groups, Final states ‘F’ and Non-Final states ‘NF’.
3. Make two dimensional matrix labeled by states of DFA at left side and bottom.
4. The major diagonal and upper triangular matrix is marked with dashes.
5. Write the combinations by attaching final states to non final states and put X in that combinations in the matrix.
6. Write the combinations by attaching non final states to non final states.
7. For each pair of states, draw the transition table with input symbols on row side and states on column side and generate new pair of states.
8. If any of new pair of states generated is either X (or) *x*, then the pair taken is marked with *x*. Otherwise pair is marked with zero ‘0’.
9. Write the modified matrix.
10. The combinations of entries 0 are the states of minimized DFA.
11. Draw transition table for the minimized DFA with modified states on row side and input symbols on column side and generate the transitions.
12. Draw transition diagram based on transition table obtained in step 6. This is the required minimized DFA.
13. Remove any useless states to obtain optimized DFA.

Finite Automata with output (TRANSDUCERS):

1. MOORE MACHINE:

Definition:

It was proposed by Edward F. Moore. Moore machine is a finite automata with output where output depends on present state only. In this machine, states are labeled with state name and output.

Mathematical representation of Moore machine:

Moore machine is formally defined as 6-tuple $\boldsymbol{M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)}$ where

$$
\begin{aligned}
&M = Moore\ Machine \\
&Q - finite\ number\ of\ states \\
&\Sigma - input\ alphabet\ or\ set\ of\ input\ symbols \\
&\Delta - output\ alphabet\ or\ set\ of\ output\ symbols \\
&\delta - input\ transition\ function\ defined\ by\ \delta: QX\Sigma \rightarrow \mathrm{Q} \\
&\lambda - output\ transition\ function\ defined\ by\ \delta: Q \rightarrow \Delta \\
&q_0 - \ start\ state\ q_0 \in Q
\end{aligned}
$$

2. MEALY MACHINE:

Definition:

It was proposed by George H. Mealy. Mealy machine is a finite automata with output where output depends on present state and present input. In this machine, transition is labeled with input and output.

Mathematical representation of Mealy machine:

Mealy machine is formally defined as 6-tuple $\boldsymbol{M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)}$ where

$$
\begin{aligned}
&M = Moore\ Machine \\
&Q - finite\ number\ of\ states \\
&\Sigma - input\ alphabet\ or\ set\ of\ input\ symbols \\
&\Delta - output\ alphabet\ or\ set\ of\ output\ symbols \\
&\delta - input\ transition\ function\ defined\ by\ \delta: QX\Sigma \rightarrow \mathrm{Q} \\
&\lambda - output\ transition\ function\ defined\ by\ \delta: QX\Sigma \rightarrow \Delta \\
&q_0 - \ start\ state\ q_0 \in Q
\end{aligned}
$$

Conversion of Moore Machine to Mealy machine:

1. Draw transition table of moore machine.
2. Draw transition table format for mealy machine.
3. Put present states of moore machine in present state of mealy machine.
4. Put next states for corresponding present states and input of moore machine in next states for those present states and input of mealy machine.
5. For output, consider present state and output of moore machine.
6. Output for next state at input in mealy machine will be Output for that state as present state in moore machine.

Conversion of Mealy Machine to Moore machine:

1. Draw transition table of mealy machine.
2. Observe next state and output columns .
3. If output differs for same next state, then break that state into number of states.
4. Change next states according to new set of states.
5. Put output in modified mealy machine.

6. Draw transition table format for moore machine.
7. Put present states and next states of modified mealy machine in present state and next states of moore machine.
8. For output, consider next state and output of mealy machine.
9. Output for next state in moore machine will be Output for that state as present state in mealy machine.

Differences between DFA and NFA:

DFA	NFA
“it is a finite automata in which all the moves of a machine can be uniquely determined by using current state and input symbol	“it is a finite automata in which some of the moves of a machine cannot be uniquely determined by using current state and input symbol.
DFA is formally defined as 5-tuple $\boldsymbol{M = (Q, \Sigma, \delta, q_0, F)}$ where $M =$ $Finite\ State\ Machin$ $Q - finite\ number\ of\ states$ $\Sigma - input\ alphabet$ $\delta - transition\ function\ defined\ by$ $\delta: QX\Sigma \rightarrow \mathrm{Q}$ $q_0 -\ start\ state\ q_0 \in Q$ $F -$ $set\ of\ final\ states\ F \subseteq Q$	NFA is formally defined as 5-tuple $\boldsymbol{M = (Q, \Sigma, \delta, q_0, F)}$ where $M =$ $Finite\ State\ Machin$ $Q - finite\ number\ of\ states$ $\Sigma - input\ alphabet$ $\delta - transition\ function\ defined\ by$ $\delta: QX\Sigma \cup \{\varepsilon\} \rightarrow 2^{\mathrm{Q}}, 2^{\mathrm{Q}} \subseteq Q$ $q_0 -\ start\ state\ q_0 \in Q$ $F -$ $set\ of\ final\ states\ F \subseteq Q$
It does not allow any epsilon transitions	It allows epsilon transitions
It takes less time to recognize input string.	It takes more time to recognize input string.
It occupies more space.	It occupies less space.
It accepts a string, if it is in one of the final states, after processing entire string.	It accepts a string, if some sequence of possible moves of a machine reaches final state, after processing entire string.
It rejects a string, if it is in non-final states, after processing entire string.	It rejects a string, if no sequence of possible moves of a machine reaches final state, after processing entire string.
It is bigger than NFA.	It is smaller than DFA.

Differences between Moore Machine and mealy Machine:

Moore Machine	**Mealy Machine**
It was proposed by Edward F. Moore.	It was proposed by George H. Mealy.
Moore machine is a finite automata with output where output depends on present state only.	Mealy machine is a finite automata with output where output depends on present state and present input.
In this machine, states are labeled with state name and output.	In this machine, transition is labeled with input and output.
Moore machine is formally defined as 6-tuple $\boldsymbol{M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)}$ where $M = Moore\ Machine$ $Q -$ $finite\ number\ of\ states$ $\Sigma - input\ alphabet$ $\Delta - output\ alphabet$ $\delta - input\ transition\ function\ defined$ $by\ \delta: QX\Sigma \rightarrow \mathrm{Q}$ $\lambda - output\ transition\ function\ defined$ $by\ \delta: Q \rightarrow \Delta$ $q_0 -\ start\ state\ q_0 \in Q$	Mealy machine is formally defined as 6-tuple $\boldsymbol{M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)}$ where $M = Moore\ Machine$ $Q -$ $finite\ number\ of\ states$ $\Sigma - input\ alphabet$ $\Delta - output\ alphabet$ $\delta - input\ transition\ function\ defined$ $by\ \delta: QX\Sigma \rightarrow \mathrm{Q}$ $\lambda - output\ transition\ function\ defined$ $by\ \delta: QX\Sigma \rightarrow \Delta$ $q_0 -\ start\ state\ q_0 \in Q$
For input ε, output is $\lambda(\square_0)$	For input ε, output is ε.
For input string of length ‘n’, output string consists of length ‘n+1’.	For input string of length ‘n’, output string consists of length ‘n’.