Why study automata theory :- the study of automata and complexity is an important part of the core of computer science.

1) Introduction to finite automata :-

Finite automata are a useful model for many important kinds of hardware and software.

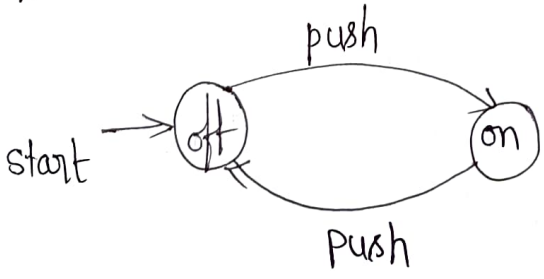
→ software for designing and checking the behavior of digital circuits.

→ 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.

→ Software for scanning large bodies of text, such as collection of web pages, to find occurency of words.

→ Software for verifying systems of all types that have a finite number of distinct states.

Perhaps the simplest nontrivial finite automaton is an on/off switch.



If the switch is in the off state, then pressing the button changes it to the on state, and if the switch is in on state then pressing the same button turns it to the off state.

ii) Structural representations :-

There are two important notations that play an important role in the study of automata & their applications.

→ Grammars are useful models when designing software that processes data with a recursive structure. The best-known example is a parser.

→ Regular expressions also denote the structure of data, especially text strings. The unix style regular expression '[A-Z][a-z]*[ ][A-Z][A-Z]' represents capitalized words followed by a space and two capital letters.

iii) Automata and Complexity :- Automata are essential for the study of the limits of computation. There are two important issues,

→ what can a computer do at all? This study is called "decidability" and the problems that can be solved by computer are called "decidable".

→ what can a computer do efficiently? This study is called "intractability" and the problems that can be solved by a computer using no more time than some slowly growing function of the size of the input are called "tractable".

The central Concepts of Automata theory :-

Alphabets :- An alphabet is finite, nonempty set of symbols. Conventionally, we use the symbol Σ for an alphabet. Common alphabets include:

→ $\Sigma = \{0,1\}$, the binary alphabet

→ $\Sigma = \{a, b, \ldots z\}$, the set of all lower-case letters.

→ The set of all ASCII characters, or the set of all printable ASCII characters.

Strings :- A string is a finite sequence of symbols choosen from some alphabet.

Ex :- 01101 is a string from the binary alphabet $\Sigma = \{0,1\}$.

The string 111 is another string choosen from this alphabet.

1) **Empty string :-** The empty string is the string with zero occurences of symbols. This string denoted "ε" is a string that may be choosen from any alphabet whatsoever.

2) **String length :-** The length of the string is the no. of positions for symbols in the string. For instance, 01101 has length 5.

The standard notation for the length of a string w is $|w|$. Example, $|011| = 3$ and $|\varepsilon| = 0$.

3) Powers of an alphabet :- If Σ is an alphabet, we define Σ^k to be the set of strings of length k, each of whose symbols is in Σ.

Ex :- $\Sigma = \{a, b, c\}$ then $\Sigma^1 = \{a, b, c\}$

$\Sigma^2 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$.

4) Concatenation of strings :- Let x and y be strings. Then xy denotes the concatenation of x and y, that is the string making a copy of x and following it by a copy of y.

Ex :- $x = 1101 \quad y = 0011$

$xy = 11010011$.

Languages :- A set of all strings, which are choosen from some Σ^*, where Σ is a particular alphabet, is called language. if Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a language over Σ.

Ex :- The set of binary numbers whose value is prime;

$\{10, 11, 101, 111, 1011\}$..

→ The set of strings of 0's and 1's with an equal number of each : $\{\varepsilon, 01, 10, 0011, 0101, 1001 \ldots\}$

→ Σ^* is a language for any alphabet Σ

Symbols, Alphabets and strings :-

1) **Symbols** :- A symbol is a single object and is an abstract entity that has no meaning by itself. It is often called uninterpreted. It can be character. Normally characters from a typical keyboard are used as symbols.

Ex :-

A	α	Alpha
B	β	beta
Γ	γ	gamma
Δ	δ	delta
E	ε	Epsilon

Alphabets :- An alphabet is any finite, non empty set of symbols / characters. It is denoted by Σ. while distinguishing two alphabets, the second one is usually denoted by Γ.

Ex :- $\Sigma = \{0,1\}$ is binary alphabet, consisting of 0's and 1's

$\Sigma = \{a,b \ldots z\}$ is the lowercase english alphabets.

$\Sigma = \{a,b,c\}$ is an alphabet of three symbols.

$\Sigma = \{-, \cup, \cap\}$ is an alphabet consisting of some set of operations.

Strings :- A string is a finite sequence of symbols chosen from some alphabet. In otherwords, a string is a finite sequence of symbols over an alphabet. It is usually denoted by 'w' (or) 's'. Quite often, the letters u, v, w, x, y, z are

used to denote strings.

→ the length of the string is defined as the number of symbols in the string and is denoted by $|x|$ for the string x.

→ A null string is a string with no symbols. In otherwards it is a string of length '0' and is denoted by λ (or) $\wedge$ (or) ϵ.

Ex :- if $\Sigma = \{0,1\}$ then 01010, 1111, 11, 00, 01, 10 etc, are some of the strings choosen from this alphabet.

→ if $\Sigma = \{a,b\}$, then ab, abab, abba etc are the words choosen from this alphabet.

→ Applejuice, is a string over an alphabet, is $\Sigma = \{a,b \cdots z\}$.

operations on strings :- Basically no. of operations on strings, Some of them are,

1) concatenation of strings :- It is simply the 'gluing' of Strings togather and placing them adjacent to each other inorder to form a new string.

Mathematically, if s_1 and s_2 are two strings, then the concatenation 's' of s_1 and s_2 is given by

$$S' = S_1 \circ S_2$$
$$= \{xy \mid x \in S_1, y \in S_2\}$$

Ex :- if S_1 = para and S_2 = graph then $S_1 \circ S_2$ = paragraph.

let $x = 0110$ and $y = 00110$ then $x \circ y = 0110\,00110$

let z be string then $z \circ \varepsilon = \varepsilon \circ z$.

Inductive (or) recursive definition of Concatenation of strings :-

This kind of inductive definition is very common in computer science and is given by,

let x be any string over an alphabet Σ. Then

$$x^0 = \epsilon \; ; \quad x^{i+1} = x^i \circ x \, , \; i \in N.$$

where $N = \{0, 1, 2 \ldots\}$ is the set of non negative integers.

2) Kleene closure :- The Kleene closure (or) Kleene star was introduced by stephen Kleene. It is unary operation, either on sets of strings (or) on sets of symbols (or) characters. the application of the Kleene closure to a set S is written as S^*. It is widely used for regular expressions.

→ if S is a set of strings, then S^* is defined as the smallest super set of S that contains ϵ and is closed under the string concatenation operation.

→ If S is a set of symbols (or) characters, then S^* is the set of all strings over the symbols in S, including ε.

$S^* = \cup S^i$ where $i \in N$

$i \geq 0$

$i = \{\varepsilon\} \cup S \cup S^2 \cup S^3 \cup \ldots\ldots$

where s^i refers to the i^{th} strings of S.

Ex:- if $S = \{cc, d\}$ then

$S^* = \{\epsilon$ (or) any word of composed factors of cc and d$\}$

$= \{\epsilon$ of all strings of c's and d's in which 'c' occurs in pairs.

$= \{\cup s^i,\ i \in N\}$

where $S^0 = \{\epsilon\}$

$S^1 = \{cc, d\}$

$S^2 = \{cc, dd\}$

$S^3 = \{ccd, ddd\}$

$S^4 = \{cccc, ccdd, ddcc, dddd\}$

Thus

$S^* = \{\epsilon, d, cc, dd, ccd, ddd, cccc, ccdd, ddcc, dddd\}$

3) positive closure :- The closure property, extended to set of all words of any length, except the null string i.e ϵ, is the operation called positive closure. Thus, if S is a set of strings/words, then s^+ is called positive closure of 's' with

$$S^+ = \bigcup_{i \geq 1} s^i,\ i \in 1, \ldots N$$

Ex :- if $S = \{a, ba\}$ then $S^+ = \{a, ba\}^+$ i.e

$$S^+ = \cup S^i, \quad i = 1 \text{----} n$$

where $S^1 = \{a\}$

$S^2 = \{aa, ba\}$

$S^3 = \{aaa, aba, baa\}$

$S^4 = \{aaaa, abaa, aaba, baba\}$

thus $S^+ = \{a, aa, ba, aaa, aba, baa, aaa, baba, baaa, aaba..\}$

4) String Reversal :- Intuitively, when a string x is reversed, i.e spelt backwards, then x^R is the resulting string which satisfies :

→ $x^R(i) = x(n+1-i)$, for $1 \le i \le n$

→ $|x^R| = |x|$.

Ex :- if $x = mus$ then $x^R = sum$

let $x = bottle$. Then $x^R = elttob$ and $(x^R)^R = bottle = x$

5) Part of strings :- It is important to talk about the various parts of a string, for example, a leading part, a middle segment and a trailing segment. These concepts are used at several occasions and are referred to as prefix, substring and suffix respectively.

Ex :- if x is a string $x = yz$ for some z, then y is prefix of x. such that $x = yzu$, then z is a substring of x. if $x = yz$ for some y, then z is a suffix of x.

Formal languages :- A formal language is an abstraction for general characteristics of programming languages, that can be defined as a set of strings, all of which are chosen from some particular alphabet Σ.

→ A formal language is a set of finite length words drawn from some finite alphabet.

The languages are denoted by letter L, with (or) without a subscript.

→ L(M) is a language defined by a machine M, that accepts a certain set of strings.

→ L(G) is is a language defined by a grammar G, that recognises a certain set of strings.

→ L(r) is a language defined by a regular expression r.

Ex :- Let $\Sigma = \{z\}$

Then the language of all possible strings is given by

$$L_1 = \{z, zz, zzz, \ldots\}$$

which can be written as

$$L_1 = \{z^n\}, \text{ for } n = 1, 2, 3 \ldots$$

Here L_1 does not contain the null string.

it is to be noted that the concatenation of z^m and z^n is z^{m+n}

Formal language specification :- Formal language can be specified in a number of ways, ~~like~~

K. Lavanya

Elements of finite state system :- Finite state machines consist of 4 main elements:

→ states : which define the behaviour and may produce actions.
→ state transitions : which are movement from one state to other.
→ rules (or) conditions : which must be allow a state transition.
→ input events : which are either externally (or) internally generated.

State :- A state is a complete set of properties, transmitted by an object to an observer via one (or) more channels. Any change in the nature, a state is detected by an observer.

The following are the some of types of states.

1) **start state :-** An initial state (or) condition of finite state machine.
2) **accepting state :-** If finite state machine finishes an input string and is in an accepting state, the string is accepted (or) considered to be valid.
3) **Next state :-** The state immediatly following the current state, defined by the transition function of a FSM and the input, is the next state.
4) **universal state :-** A state is in an alternating Turing machine, from which the machine only accepts, all the possible moves leading to acceptance, is called universal state.
5) **Existential state :-** A state in a non deterministic Turing machine, from which the machine accepts any move that

leads to acceptance, is the existential state.

6) Dead/Trap state :- A non final state of the FSM, whose transitions on every input symbol terminates on itself.

Transition :- the following are the two equivalent definitions for the transition of a finite state system.

1) Transition is the act of passing from one state to the next.
2) A change from one place (or) state (or) object (or) stage to another

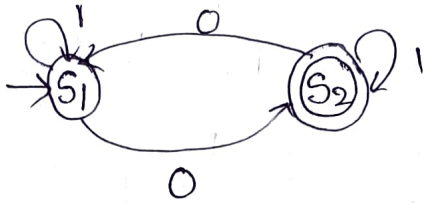
Transitions are represented in the following ways,

1) State diagram / Transition diagram.
2) state transition table.
3) Transition functions.

State diagram :- A diagram consisting of circles to represent states and directed line segments to represent transitions b/w the states is called a state diagram. The symbols used in state diagrams are,

Notation	meaning
→○	start of process
◎	End of process / accepting state
→	Transition
Ready →	input (Guarded transition)
open →	output.

Ex:-



For finite state machine, a state diagram is a directed graph where s_1 and s_2 are states and s_2 is the accepting state and s_1 is the initial state. Each edge is labelled with the input.

State transition table :- A state transition table can be described, in general as;

→ the tabular representation of transitions that take two arguments and a return value.

→ Rows correspond to states and columns correspond to inputs.

→ Entries correspond to next states.

→ The start state is marked with arrow (→).

→ The accepting states are marked with a star (*).

Inputs / States	Present inputs			
	a_1	a_2	...	a_n
S_0				
S_1				
⋮				
S_n				

For a finite state machine, a state transition table is a table describing the transition function, δ of a finite automata.

K. Lavanya

Ex :-

Inputs / States	present states	
	0	1
→ S1	S2	S1
* S2	S2,1	S2

Transition function :- The transition function of a automaton, which is also called it's transition matrix (or) transition table specifies the state into the which machine will move, on the basis of it's current state. Thus, the transition function δ can be represented as a set of ordered triples of the form

$$\langle q_i, w, q_j \rangle$$

$q_i \in Q$ is a label, identifying the current state
$w \in \delta$ is a the word
$q_j \in Q$ is the label of the new state.

Mathematical Representation of finite State machine :-

Ordered quintuple specification of finite automata :- A finite state automata, A can be specified by the entries in the following ordered quintuple :

$$A = \langle Q, \Sigma, \delta, q_0, F \rangle$$

where

→ 'Q' is a finite non empty set of labels identifying the states of the machine.

→ Σ is a finite non empty set of input symbols.

→ δ is the transition function of the machine which maps from $\delta : Q \times \Sigma \to Q$ ($Q \times \Sigma$ into Q)

→ $q_0 \in Q$ identifies the starting state of the machine

→ $F \subseteq Q$ is a set of accepting states (or) final states.

Properties of transition functions :-

$P_1 \to$ $\delta(q, \Lambda) = q$ is a finite automaton. this means that the state of the system can be changed only by an input symbol.

$P_2 \to$ For all strings w and input symbols a,

$$\delta(q, aw) = \delta(\delta(q, a), w)$$

$$\delta(q, wa) = \delta(\delta(q, w), a)$$

this property gives the state after the automaton consumes or reads the first symbol of a string aw and the state after the automaton consumes a prefix of the string wa.

Acceptability of a string by a finite automaton :-

A string x is accepted by a finite automaton

$$M = (Q, \Sigma, \delta, q_0, F)$$

if $\delta(q_0, x) = q$ for some $q \in F$

this is basically the acceptability of a string by the final state.

→ A final state is also called ...

Ex :-

state	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Give the entire sequence of the ~~string~~ states for the input string 110101

Sol :- $\delta(q_0, 110101) = \delta(q_1, 10101)$

$= \delta(q_0, 0101)$

$= \delta(q_2, 101)$

$= \delta(q_3, 01)$

$= \delta(q_1, 1)$

$= \delta(q_0, \Lambda)$

$= q_0$

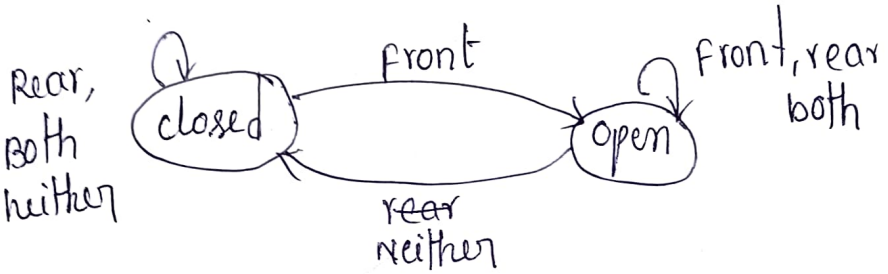
hence

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

The symbol ↓ indicates that the current input sy... is being processed by the machine.

Finite automata :- It is mathematical model of a system with discrete inputs and outputs, such a system can be in any one of the finite number of internal states and each state of the system provides sufficient information concerning the past inputs.

Ex :- The state diagram and the transition table for automatic door controller,



1. State diagram.

input signal / state	Neither	Front	rear	Both
closed	closed	open	closed	closed
open	closed	open	open	open

2. state transition table.

According to the formal definition, a finite automata is a list of five objects : set of states, input alph rules for moving, start and accept states. Thus a mathem definition of finite automata is a five tuple consisting of

→ **Deterministic finite automata (DFA) :-**

A deterministic finite automaton is a finite state machine where for each pair of states and input symbol there is a unique next state.

Elements of DFA :- The deterministic finite automata exhibits the following five characteristics.

→ a finite set of states Q

→ an alphabet Σ of possible input symbols,

→ a transition function δ such that

$$\delta(x, 1) = y \text{ where } x, y \in Q \text{ and } 1 \in \Sigma$$

→ The initial state $q_0 \in Q$,

→ The set of final states (F), where $F \subseteq Q$.

Operation of DFA :-

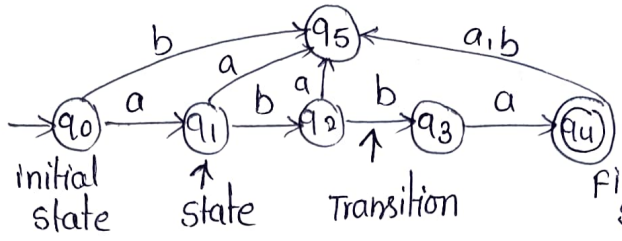
Initially, the DFA is assumed to be in the initial state q_0 with it's read head on the left most symbol of the input string. During each of the movie of DFA, the read head moves one position the right.

Thus each move consumes one input symbol, when the end of the string is reached, the string is accepted, if DFA is in one of it's initial states, else rejected.

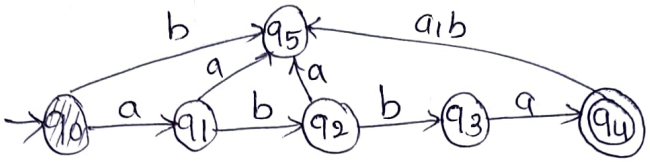
Ex :- The working of DFA is

→ DFA that accepts the input string abba

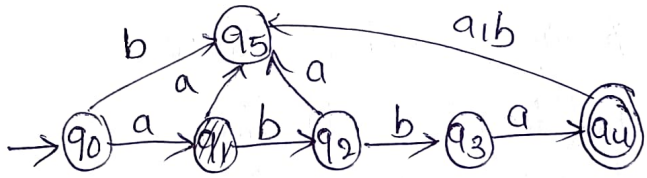
Step1:-



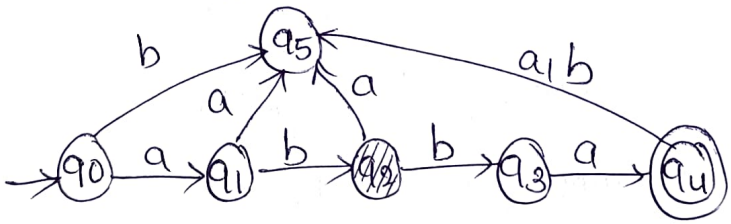
Step2:- Now the read the input string abba



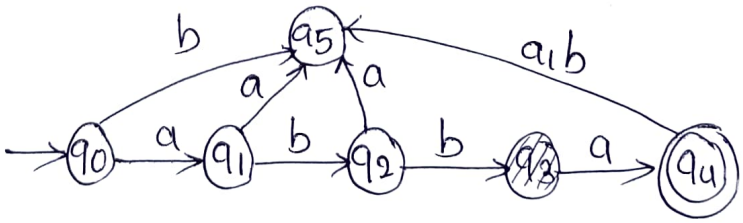
Step3:- Now read the input symbol a



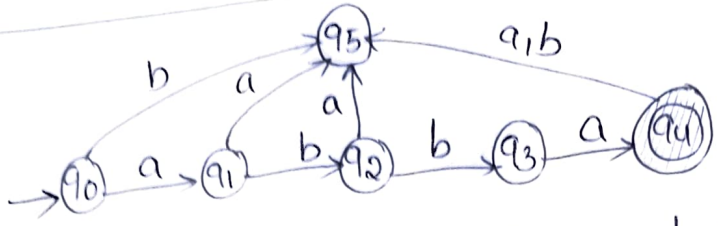
Step4:- Read the symbol b from state q_1



Step5:- Read the next symbol b, the transition from q_2



Step6:- Read the last symbol is a



The transition reaches to final state, hence, the string is accepted by DFA.

Ordered quintuple specification of DFA :- Finally a DFA, M is a five tuples $M = (Q, \Sigma, \delta, q_0, F)$

where,

- → Q is a finite set of states of finite automata.
- → Σ is a finite set of input symbols called the alphabet.
- → $\delta : Q \times \Sigma \to Q$ is the transition function.
- → $q_0 \in Q$ is the start state.
- → $F \subseteq Q$ is the set of accept states.

if from a state p, there exists a transition going to state q on an input symbol a, then this is written as

$$\delta(p, a) = q$$

→ δ - is a function whose domain is a set of ordered pairs (p, a)

→ p - a state

→ a - input symbol.

Thus δ defines a mapping whose range will be a set of states

$$\delta : Q \times \Sigma \longrightarrow Q.$$

Description of a DFA :- The transitions of DFA can be represented using transition diagram (or) transition table,

Transition diagram :- if $\delta(p,a) = q$, then the arrow goes from the vertex which corresponds to state p, to the vertex that corresponds to state q labelled by a.

Transition table :- Rows corresponds to states, and columns corresponds to inputs. Entries correspond to next state to indicate the transition of DFA.

Extended transition function for DFA :-

For DFA, $M = (Q, \Sigma, \delta, q_0, F)$ the function δ is extended as

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

→ For any state q of Q

$$\delta(q, \varepsilon) = q$$

This means that DFA stays in same state q, when it reads any empty string at q.

→ For any state q of Q, any string $x \in \Sigma^*$ with a as the last symbol of x and $a \in \Sigma$.

$$\delta(q, xa) = \delta(\delta(q, x), a)$$

languages accepted by DFA:-

The language accepted by DFA, $M = (Q, \Sigma, \delta, q_0, F)$ is the set of all strings in Σ accepted by M i.e

$$L(M) = \{ w \in \Sigma^* \mid \delta(q_0, w) \in F \}$$

A language is said to be rejected by DFA, such that

$$L(M) = \{ w \in \Sigma^* \mid \delta(q_0, w) \notin F \}.$$

Design of DFA's :- The basic design strategy for DFA is as follows:

1) understand the language properties for which the DFA has to be designed.
2) determine the state set required.
3) Identify the initial, accepting and dead state of DFA.
4) For each state, decide on transition to be made for ~~which~~ each character of the input string.
5) obtain the transition table and diagram for DFA.
6) Test the DFA obtained on short strings.

Example :- To design a DFA that accepts set of all strings that contain 0's (or) 1's and end with 00.

Sol:- we are designed required to design a DFA for the regular expression $r = (0+1)^*00$

In otherwords the DFA should be designed to accept the language of r,

$L(M) = \{00, 100, 1100, 0000, 111000 \cdots\}$ where M is a DFA. Consider,

$\Sigma = \{0, 1\}$

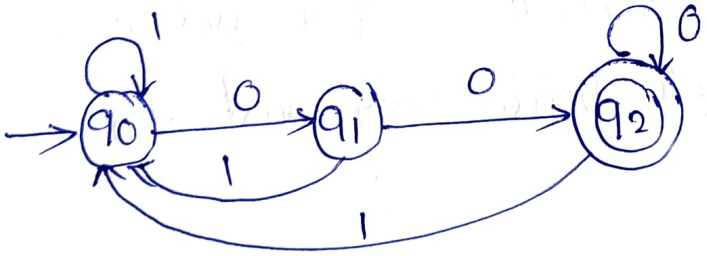
$Q = \{q_0, q_1, q_2\}$

q_0 = initial state

q_2 = final state

$\delta : Q \times \Sigma \to Q$ is given by

Transition diagram:-



$L(M) = \{w \in \Sigma^* \mid w \text{ ends with } 00\}$

Transition table:-

Q \ Σ	0	1
→q0	q1	q0
q1	q2	q0
*q2	q2	q0

DFA action for the input string :-

→ To show that the string 100 is accepted by DFA. using vdash function (⊢).

Consider

$$(q_0, 100) = \vdash M(q_0, 00)$$
$$= \vdash M(q_1, 0)$$
$$\vdash M(q_2, e) \Rightarrow q_2$$

Since we reached the end of the input (e) and we are in the final state q_2. hence 100 is accepted by M. Thus 100 is in L(M).

Non deterministic finite automata (NFA) :-

when a machine reads the next input symbol, then the next state is uniquely determined. This is called as deterministic computation. However in case of non-deterministic machine, multiple choices may exist for the next state at any point.

Thus, every deterministic finite automaton (DFA) is automatically a non deterministic finite automaton

Elements of NFA :- A non deterministic finite automaton exhibits the following 5 characteristics :

→ Finite set of states (Q)

→ An alphabet Σ of possible input symbols.

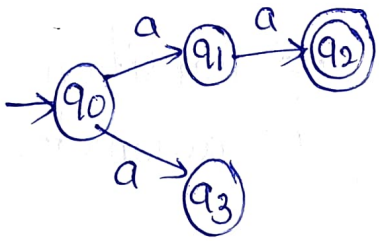
→ the transition function (δ), which specifies the nature of transition at a given state due to the input symbols.

→ the initial state q_0.

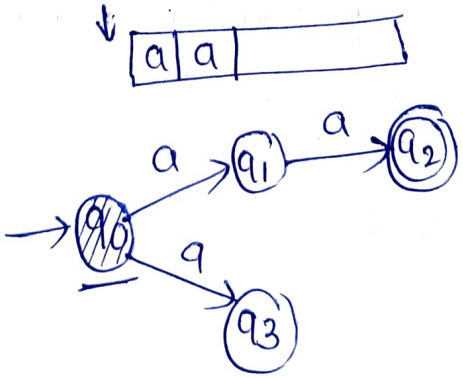
→ the set of final states (F).

Operation of NFA :- The nondeterminism may be viewed as a type of parallel computation in which several processes are running concurrently. If atleast one of these processes accepts, then the entire computation automatically accepts. The demonstration of operation of NFA is,

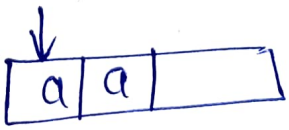
→ string aa , Alphabet = {a}, Two choices

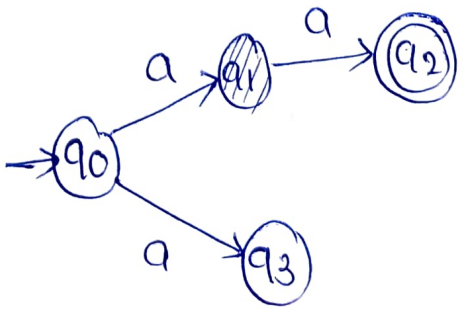


→ First choice,

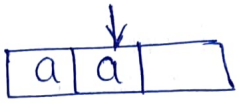


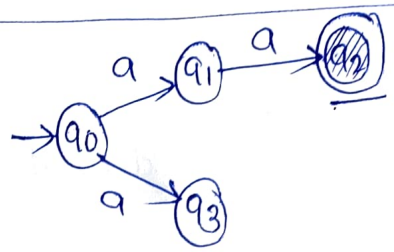
→ First choice,



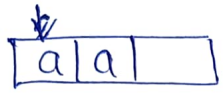


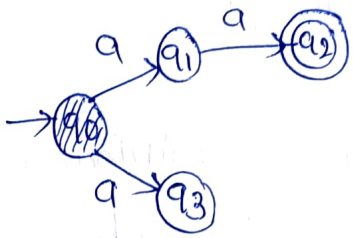
→ First choice



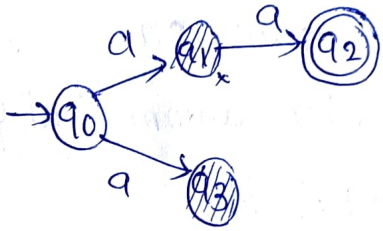


→ Second choice

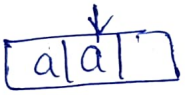


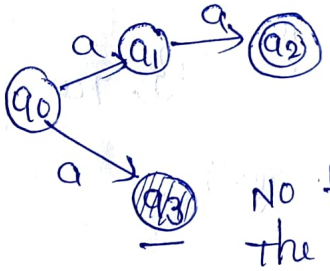


→ second choice



→ Second choice





No transition: the automaton hands "rejects"

ordered quintuple specification :- formally a, nondeterministic finite automaton M is a five tuple :

$$M = (Q, \Sigma, \delta, q_0, F)$$

→ Q is a finite set of states.
→ Σ is a set of input symbols
→ δ is a transition function i.e $\delta : Q \times \Sigma \to P(Q)$
→ q_0 is a initial state
→ $F \subseteq Q$ is a set of final states.

Description of an NFA :- Transition of NFA can be represented using a transition table or diagram.

Transition diagram :- NFA says that if $\delta(q,a) = \{q_1, \ldots q_k\}$ then there will be an arc labelled 'a' from 'q' to each of $q_1, q_2 \ldots q_k$.

Transition Table :- NFA has rows correspond to states, which are represented with in the braces, and columns correspond to the inputs.

language accepted by NFA :- the language accepted by NFA, M for a string x accepted as;

$$L(M) = \{x \mid \delta(q_0, x) = p, \text{ where } p \text{ contains atleast one number of } F\}$$

(or)

$$L(M) = \{x \mid \delta(q_0, x) \cap F \neq \{\emptyset\}\}.$$

This means NFA accepts the string x, iff it can reach an accepting state by reading x starting at the initial state.

Design of NFA'S :- The basic design strategy for an NFA is as follows :

→ understand the language properties for which the NFA has to be designed.

→ Determine the alphabet and state set required.

→ Identify the initial, accepting and dead states of NFA.
→ obtain the transitions to be made for each state on each character of the input string.
→ Draw the transition table and diagram for NFA.
→ Test the NFA obtained on short strings.

Example :- Design an NFA the accepts set of all strings over {0,1} that have at least two consequtive 0's ~~and~~ (or) 1's.

Sol :- we are required to design a NFA for the regular Expression $r = (0+1)^*(00+11)(0+1)^*$

'M' should be designed to accept the language of r.

i.e $L(M) = \{00, 11, 0110, 1001, 10110101 ----\}$ where 'M' is NFA. Consider

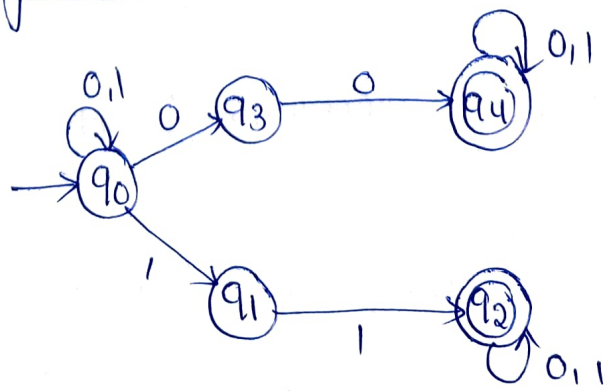
$\Sigma = \{0,1\}$

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

q_0 = initial state

$\{q_2, q_4\}$ = final states and δ is given by

Transition diagram :-



$L(M) = \{w \in \Sigma^* / w$ contains atleast two consecutive 0's, 1's$\}$

Transition Table :-

Q \ Σ	Present i/p	
	0	1
→q_0	$\{q_0, q_3\}$	$\{q_0 q_1\}$
q_1	ϕ	$\{q_2\}$
q_2	q_2	q_2
q_3	q_4	ϕ
*q_4	q_4	q_4

- NFA action for the input string :- To show that the string is accepted by NFA : 010011

i) using extended transition function :-

$\delta(q_0, 0) = \{q_0, q_3\}$ $\delta(q_0, 1) = \{\phi q_0, q_1\}$

$\delta(q_0, 01) = \delta(\delta(q_0, 0), 1)$

$= \delta(\{q_0, q_3\}, 1)$

$= \delta((q_0, 1) \cup \delta(q_3, 1)$

$= \{q_0, q_1\} \cup \{\phi\} = \{q_0, q_1\}$

$\delta(q_0, 010) = \delta(\delta(q_0, 01), 0)$

$= \delta(\{q_0, q_1\}, 0)$

$= \delta(q_0, 0) \cup \delta(q_1, 0)$

$= \{q_0, q_3\} \cup \{q_0, q_3\} = \{q_0, q_3\}$

$\delta(q_0, 0100) = \delta(q\delta(q_0, 010), 0)$

$= \delta((q_0, q_3), 0) \Rightarrow \{q_0, q_3\} \cup \{\phi\} = \{q_0, q_3, q_4\}$

K. Lavanya

$\delta(q_0, 01001) = \delta(\delta(q_0, 0100), 1)$

$= \delta(\{q_0, q_3, q_4\}, 1)$

$= \delta(q_0, 1) \cup \delta(q_3, 1) = \{q_0, q_1\} \cup \{\phi\} = \{q_0, q_1\}$

$= \delta(q_0, 1) \cup \delta(q_3, 1) \cup \delta(q_4, 1)$

$= \{q_0, q_1\} \cup \{\phi\} \cup \{q_4\} \Rightarrow \{q_0, q_1, q_4\}$

$\delta(q_0, 010011) = \delta(\delta(q_0, 01001), 1)$

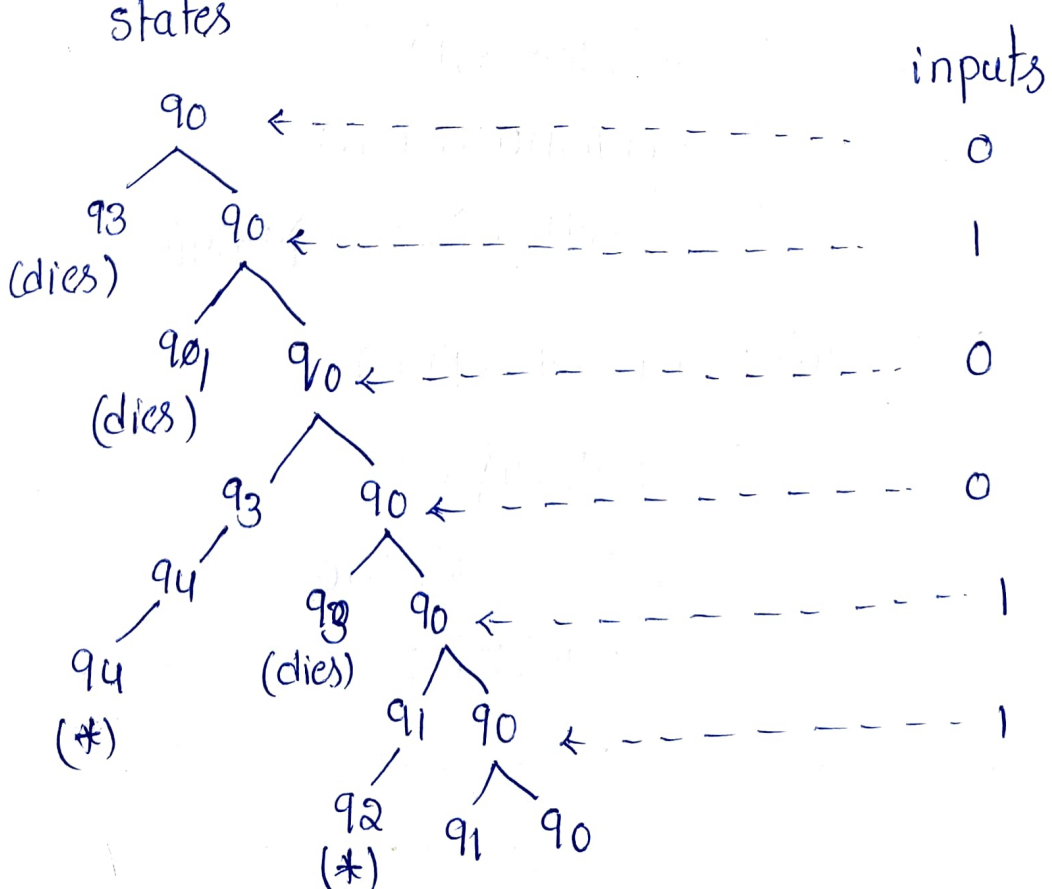
$= \delta(\{q_0, q_1, q_4\}, 1)$

$= \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_4, 1)$

$= \{q_0, q_1\} \cup \{q_2\} \cup \{q_4\} \Rightarrow \{q_0, q_1, q_2, q_4\}$.

Since p contains q_4 and q_2, clearly 010011 is accepted by L(M).

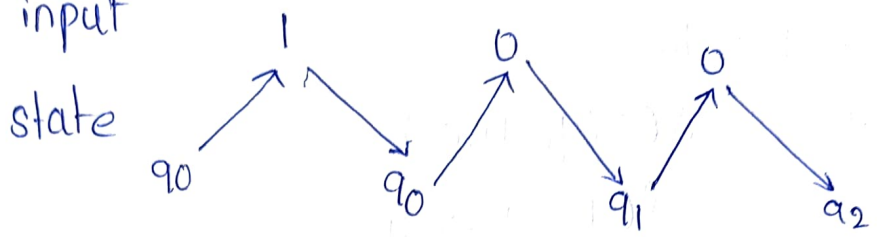
ii) using tree state diagram for input 010011.



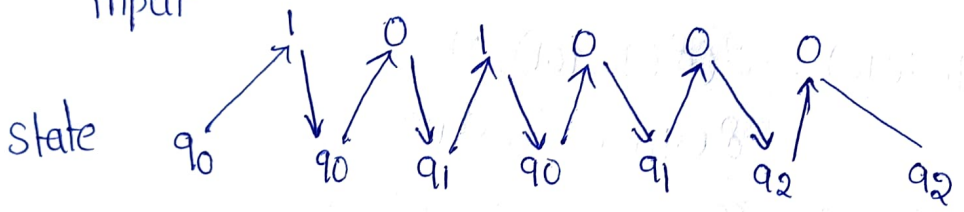
DFA action for the input string:-

→ To show that the string is 100 is accepted by DFA:

i) using sequence state diagram:-



another string is 101000



Since we encounter the end of the input and we are in the final state, we say that string is accepted by machine M. thus 100, 101000 are in L(M)

ii) using extended transition function:-

Consider the strings 100, 101000

→ 100

$$\delta(q_0,1) = q_0$$

$$\delta(q_0,10) = \delta(\delta(q_0,1),0)$$
$$= \delta(q_0,0)$$
$$= q_1$$

$\delta(q_0,100) = \delta(\delta(\delta(q_0,10),0)$

$= \delta(q_1,0)$

$= q_2$

→ 101000

$\delta(q_0,1) = q_0$

$\delta(q_0,10) = \delta(\delta(q_0,1),0)$

$= \delta(q_0,0) \Rightarrow q_1$

$\delta(q_0,101) = \delta(\delta(q_0,10),1)$

$= \delta(q_1,1) \Rightarrow q_0$

$\delta(q_0,1010) = \delta(\delta(q_0,101),0)$

$= \delta(q_0,0) = q_1$

$\delta(q_0,10100) = \delta(\delta(q_0,1010),0)$

$= \delta(q_1,0) = q_2$

$\delta(q_0,101000) = \delta(\delta(q_0,10100),0) = q_2$

Since after scanning the entire string, we reach the final state q_2, the given strings are accepted by DFA M.

iii) using Yadyh function (⊢) : Consider strings 101000, 100.

→ 101000 $\delta(q_0,101000) \vdash \delta(q_0,01000)$

$\vdash \delta(q_1,1000)$

$\vdash \delta(q_0,000)$

$\vdash \delta(q_1,00)$

$\vdash \delta(q_2,0)$

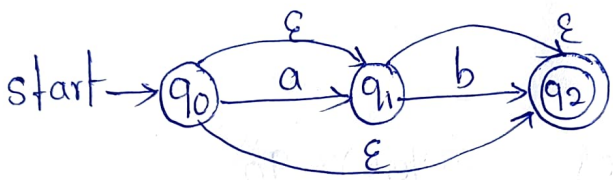
$\vdash q_2$ //

Non deterministic automata with ϵ-moves :-

The NFA that responds to an empty string 'ϵ' and moves to the next state is called NFA with ϵ-moves.

Alternatively, a finite automata that is modified to permit transitions without input symbols, and with '0', one, or more transitions on input symbols, is an NFA with ϵ-moves.

Ex :- NFA with ϵ transitions b/w q_0 to q_1 and q_1 to q_2



Transitions of NFA-ϵ :- with respect to an NFA-ϵ, there exist two transitions.

ϵ-transitions :- Transitions that take place without reading any input symbols are called ϵ-Transitions.

Non ϵ-transitions :- Transitions that take place with the reading of input symbols are called non ϵ-transitions.

Acceptance of strings by NFA-ϵ :- In order to identify all states in the actual path, that are formed with ϵ-transitions, the Concept of ϵ-closure is used.

ϵ-closure for a state is the set of states reachable from the state without reading any symbol.

K. Lavanya

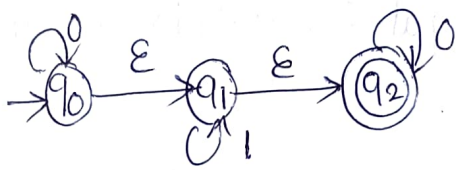
In general, if P is the power set of Q then,

→ $P \subseteq \epsilon\text{-closure}(P)$

→ For q of Q, if $q \in \epsilon\text{-closure}(P)$ then

$\delta(q, \epsilon) \subseteq \epsilon\text{-closure}$

Ex :-



To compute ϵ-closure for all states

ϵ-closure (q_0) :-

$\epsilon\text{-closure}(\{q_0\}) \subseteq \{q_0\}$ $\quad \delta(q_0, \epsilon) = q_0$

$\epsilon\text{-closure}(\{q_0\}) \subseteq \{q_0, q_1\}$ $\quad \delta(q_0, \epsilon) = q_1$

$\epsilon\text{-closure}(\{q_0\}) \subseteq \{q_0, q_1, q_2\}$ $\quad \delta(q_1, \epsilon) = q_2$

$\epsilon\text{-closure}(\{q_0\}) \subseteq \{q_0, q_1, q_2\}$ $\quad \delta(q_2, \epsilon) = \phi$

Hence, the process of generating ϵ-closure transitions terminates and $\epsilon\text{-closure}(\{q_0\}) = \{q_0, q_1, q_2\}$

ϵ-closure (q_1) :-

$\epsilon\text{-closure}(\{q_1\}) \subseteq \{q_1\}$ $\quad \delta(q_1, \epsilon) = \{q_1\}$

$\epsilon\text{-closure}(\{q_1\}) \subseteq \{q_1, q_2\}$ $\quad \delta(q_1, \epsilon) = \{q_2\}$

$\epsilon\text{-closure}(\{q_1\}) \subseteq \{q_1, q_2\}$ $\quad \delta(q_2, \epsilon) = \{\phi\}$

Hence $\epsilon\text{-closure}(\{q_1\}) = q_0\{q_1, q_2\}$

Extended transition function for NFA-ε :- For an NFA-ε the function δ is extended as

$$\delta : Q \times \Sigma^* \to P(Q)$$

and is recursively defined as follows :

a) → For any state q of Q

$$\delta(q, \epsilon) = \epsilon\text{-closure}(\{q\})$$

b) → For any state q of Q, any string $x \in \Sigma^*$ with 'a' as the last symbol of x and $a \in \Sigma$,

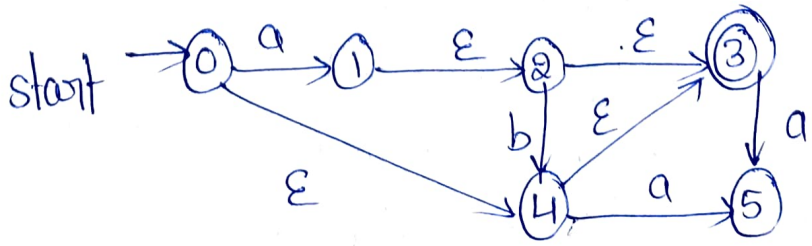
$$\delta(q, xa) = \epsilon\text{-closure}\left(\bigcup_{\text{for every } p \text{ in } q} \delta(p, a)\right)$$

(or)

$$\delta(q, xa) = \epsilon\text{-closure}\left(\bigcup_{p \in \delta(q,x)} \delta(p, a)\right)$$

This means that $\delta(q, xa)$ is obtained by first finding the states that can be reached from q by reading x ($\delta(q, x)$), then from each of those states which can be reached from p by reading a are found. and finally ϵ-closure (i.e by taking the ϵ-closure of $\delta(p, a)$ are read.

Ex :- To compute $\delta(0, ab)$ for the NFA-ε



Sol:- To Compute ϵ-closure :-

ϵ-closure({0}) :-

ϵ-closure ({0}) $\sqsubseteq$ {0} $\quad \delta(0,\epsilon) = \{0\}$

ϵ-closure ({0}) $\sqsubseteq$ {0,4} $\quad \delta(0,\epsilon) = \{4\}$

ϵ-closure ({0}) $\sqsubseteq$ {0,4,3} $\quad \delta(04,\epsilon) = \{3\}$

ϵ-closure ({0}) $\sqsubseteq$ {0,4,3} $\quad \delta(3,\epsilon) = \{\phi\}$

i.e ϵ-closure ({0}) = {0,4,3}

Ily

ϵ-closure (1) = {1,2,3}

ϵ-closure (2) = {2,3}

ϵ- closure (3) = {ϕ}

ϵ-closure (4) = {4,3}

ϵ- closure (5) = {5}

→ To Compute δ :-

δ (ϵ-closure(0), a) = δ({0,3,4}, a)

= $\delta(0,a) \cup \delta(3,a) \cup \delta(4,a)$

= {1} ∪ {5} ∪ {5} => {1,5}

δ (ϵ-closure(0), ab) = δ (ϵ-closure (δ(ϵ-closure(0), a), b)

= δ(ϵ-closure ({1,5}, b))

= δ(ϵ-closure(1) ∪ ϵ-closure(5)), b)

= δ({1,2,3,5}, b)

= $\delta(1,b) \cup \delta(2,b) \cup \delta(3,b) \cup \delta(5,b)$

= {ϕ} ∪ {4} ∪ {ϕ} ∪ {ϕ} => {4}

Finally ϵ-closure(δ(ϵ-closure(0), ab))

$= \epsilon$-closure(4)

$= \{3, 4\}$ //

Design of NFA-ϵ moves :- The basic design strategy for NFA-ϵ is as follows,

1) understand the language properties for which the NFA-ϵ has to be designed.
2) Determine the alphabet and state set required.
3) identify the initial, accepting and dead states of NFA-ϵ.
4) Identify all ϵ-transitions.
5) obtain the non ϵ-transitions for each state, on each character of the input string.
6) Draw the transition table and diagram for NFA-ϵ.
7) Compute ϵ-closure for each state.
8) Test the NFA-ϵ obtained on short strings.

Example :- Design an NFA-ϵ that accepts the string abac and all its suffixes.

Sol:- we are required to design an NFA-ϵ for the regular expression $r = c + ac + bac + abac$.

M should be designed to accept the language for r.

i.e $L(M) = \{\epsilon, c, ac, bac, abac\}$

where M is the NFA-ϵ

Consider, $\Sigma = \{a, b, c\}$

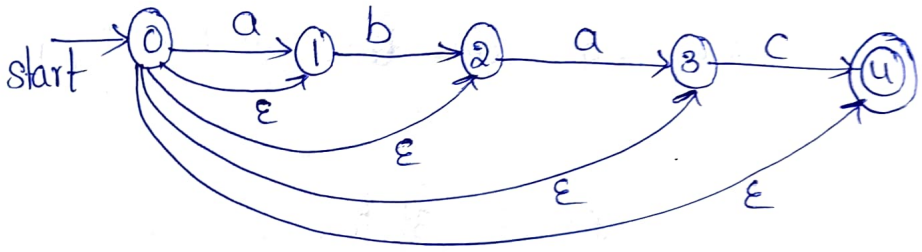
$Q = \{0, 1, 2, 3, 4\}$

q_0 = initial state

F = Final state.

and δ is given by,

The transion diagram is,



$L(M) = \{\omega \in \Sigma^* \mid \omega$ contain abac and all its suffixes.

Transition table :-

Q \ Σ	present inputs			
	a	b	c	ε
→0	{1}	ϕ	ϕ	{1,2,3,4}
1	ϕ	{2}	ϕ	ϕ
2	{3}	ϕ	ϕ	ϕ
3	ϕ	ϕ	{4}	ϕ
*4	ϕ	ϕ	ϕ	ϕ

NFA-ϵ action for the input string :-

To show that string ac is accepted by NFA - ϵ :

a) To compute ϵ-closure :-

ϵ-closure $\{0\} = \{0,1,2,3,4\}$

ϵ-closure $(1) = \{1\}$

ϵ-closure $(2) = \{2\}$

ϵ-closure $(3) = \{3\}$

ϵ-closure $(4) = \{4\}$

b) To compute δ :-

$\delta(\epsilon\text{-closure}(0), a) = \delta(0,1,2,3,4), a)$

$= \delta(0,a) \cup \delta(1,a) \cup \delta(2,a) \cup \delta(3,a) \cup \delta(4,a)$

$= \{1\} \cup \phi \cup \phi \cup \{3\} \cup \phi$

$= \{1,3\}$

$\delta(\epsilon\text{-closure}(0), abc) = \delta(\epsilon\text{-closure}(\delta(\epsilon\text{-closure}(0), a), c))$

$\Rightarrow \delta(\epsilon\text{-closure}(\{1,3\}, c))$

$= \delta((\epsilon\text{-closure}(1) \cup \epsilon\text{-closure}(3)), c)$

$= \delta((1,3), c)$

$= \delta(1,c) \cup \delta(3,c)$

$= \{\phi\} \cup \{4\}$

$\Rightarrow \{4\}$

Now here '4' is the final state, thus ac is accepted by NFA-ϵ.

Advantages of non deterministic finite automata :-

1) NFA can be smaller, easier to construct and understand than a DFA that accepts the same language.
2) It is useful for proving some theorms.
3) It gives a good introduction to non determinism in more powerful computational models, where non determinism plays an important role.
4) Non determinism is a kind of parallal computation where several processes can be running concurrently.

When NFA splits to follow several choices, that correspond to a process 'forking' into several children, each proceding seperatly.

NFA versus DFA :-

1) a DFA is a special case of NFA.

→ a) In a DFA, at every state q and for every symbol 'a', has a unique a transition i.e there is unique q' such that $q \xrightarrow{a} q'$. So this is not necessary in an NFA. At any state, an NFA may have multiple a-transitions.

→ b) In a DFA, transition arrows are labelled by symbols from Σ while in an NFA, they are labelled by symbols from $\Sigma \cup \{\epsilon\}$, i.e an NFA may have ϵ-transitions

2) NFA's are very convenient for demonstrating the languages to be regular. often it is much easier than the to provide an NFA for a language than a DFA.

3) space and time taken to recognize regular expressions :-

NFAS are not as well suited to language recognition because non determinism gives the machine the possibility of making incorrect choices.

Thus NFAS are more compact (in space), but take time to backtrack all choices. on the other hand, a DFA takes more space, but saves time (speeds up computation).

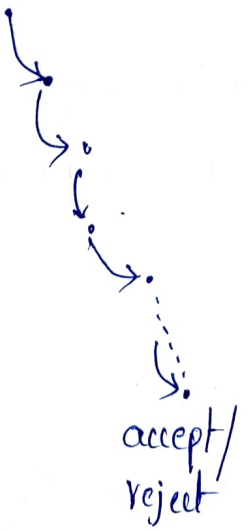
4) Transition function:-

For DFA, 'δ' is defined as $\delta : Q \times \Sigma \to Q$

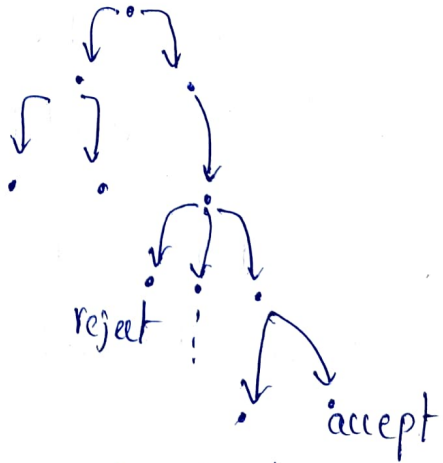
For NFA, δ is defined as $\delta : Q \times (\Sigma \cup \epsilon) \to P(Q)$.

5) NFA computation can be represented as a tree structure.

Ex:-



unidirectional DFA computation



multi directional.

Equivalent automata

Equivalent finite state automata :- The set of words by a FSA, M is the language accepted by M and is denoted by L(M).

Two finite automata M_1 and M_2 are equivalent if and only if $L(M_1) = L(M_2)$

i.e M_1 and M_2 accept (or) recognise the same language.

The equivalences of finite automata are :

→ equivalence of NFA and DFA (or) equivalence of NFA-ϵ & DFA

→ equivalence of NFA-ϵ and NFA.

Equivalence of NFA / NFA-ϵ and DFA :-

Two finite automata N and D are said to be equivalent, if $L(N) = L(D)$

where D represents deterministic finite automata and N represents non deterministic finite automata.

That is, N and D accept the same language. This means,

→ any language accepted by D can also be accepted by N, and

→ any language accepted by N can also be accepted by some D.

For an NFA with n-states, the DFA can be have 2^n states. But it is not necessary to construct 'δ' for all these 2^n states. It can constructed only for those states which are reachable from the initial state.

Theorm :- A language L is accepted by some NFA, if and only if, it is accepted by some DFA. Alternatively, for every NFA, there exists a DFA, that accepts the same language.

Sol:- This theorm has two parts to prove

→ if L is accepted by DFA, D then L is accepted by NFA, N.

→ if L is accepted by NFA, N then L is accepted by DFA, D.

Proof :-

Part a :- if L is accepted by D, then L is accepted by N.

Definition of 'D' :-

A DFA is defined by 5 tuple

$D = (Q', \Sigma', \delta', q_0', F')$

Q' = finite no. of states

δ' - Transition function $Q' \times \Sigma \to Q'$.

q_0' - Initial state

F' - final states $F' \subseteq Q'$.

Σ - Finite set of symbols.

Definition of 'N' :-

A NFA is defined by 5 tuples

$N = (Q, \Sigma, \delta, q_0, F)$

Q - Finite set of states

Σ - finite set of symbols

δ - Transition fun $Q \times \Sigma \to 2^Q$

q_0 - initial state

F - final state.

From the above definitions, it follows that every DFA is also an NFA, which implies that

if $w \in L(D)$, then $w \in L(N)$

Part b :- if L is accepted by N, then L is accepted by D.

The initial state of D is the initial state of N.

$q_0' = \{q_0\}$

$F' = \{F$, any final state$\}$

here δ' is defined as follows:

$$\delta'(\{q_1, q_2 \ldots q_i\}, a) = \{P_1, P_2 \ldots P_j\}.$$

if and only if

$$\delta(\{q_1, q_2, \ldots q_j\}, a) = \{P_1, P_2, \ldots P_j\}$$

Now we prove that for some input string 'x',

$\delta'(q_0', x) = \{q_1, q_2 \ldots q_i\}$ iff, $\delta(q_0, x) = \{q_0, q_2, \ldots q_i\}$.

we prove this by induction.

Base case :- the result is true for $|x| = 0$, if $x \in x = \epsilon$, because

$\delta'(q_0, x) = \{q_0\}$ and $\delta(q_0, x) = \{q_0\}$

hence $\delta'(q_0', x) = \{q_0\}$ iff $\delta(q_0, x) = \{q_0\}$

Now, we will show that this result is true for any string of length $(n+1)$.

let $w = xa$ with $|w| = (n+1)$ and $|x| = n$ and $a \in \Sigma$. Thus by induction,

$$\delta'(q_0', x) = \{P_1, P_2 \ldots P_j\}$$

iff $\delta(q_0, x) = \{P_1, P_2 \ldots P_j\}$ — (1)

where $\{P_1 \ldots P_j\}$ are states of N.

By definition δ'

$\delta'(\{P_1 \cdots P_j\}, a) = \{r_1 \cdots r_k\}$

iff $\delta(\{P_1 \cdots P_j\}, a) = \{r_1 \cdots r_k\}$ — (2)

Thus $\delta'(q_0', xa) = \delta'(\delta'(q_0', x), a)$

$= \delta'(\{P_1 \cdots P_j\}, a)$ using (1)

$= \{r_1, r_2 \cdots r_k\}$ using (2)

hence the result is true for $|w| = n+1$, when the result is true for a string of length n. Now $\delta'(q_0', x) \in F'$. Exactly when $\delta(q_0, x) \in F$.

$\Rightarrow L(N) = L(D)$.

Thus for every NFA there exists an equivalent DFA, which accepts the same language.

Conversion algorithm for NFA/NFA-ϵ to DFA :-

Subset Construction algorithm :-

→ Construct the start state q_0' (or) $\{q_0\}$, consisting of q_0 and all the states of NFA that can be reached from q_0 by one or several ϵ-transitions. Mark q_0' as 'unfinished'.

→ while there are 'unfinished' states'

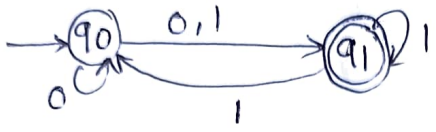
a) Take an unfinished' state S

b) For each $a \in \Sigma$, let $\delta(S,a) = \cup_{q \in S} \{t \mid q \xrightarrow{a} t\}$. if $\delta(S,a)$ is neither 'finished' nor 'unfinished' yet, then mark $\delta(S,a)$ as 'unfinished'

c) mark S as 'finished'.

→ Mark all states that contain a final state from N as the final state of D.

Ex:- Construct a DFA, equivalent to the NFA given in figure.



Sol:- The transition table of the above NFA is

Q \ Σ	present i/p's	
	0	1
q0	{q0,q1}	{q1}
q1	ϕ	{q0,q1}

no. of states in NFA = 2

possible states in DFA is $2^2 = 4$

consider

$q_0 = \{q_0\} \Rightarrow$

δ	0	1
{q0}		

← unfinished

→ process {q0}

$\delta(q_0,0) = \{q_0,q_1\}$

$\delta(q_0,1) = \{q_1\}$

δ	0	1	
{q0}	{q0,q1}	{q1}	← finished
{q0,q1}			← unfinished
{q1}			← unfinished

→ process {q0,q1}

$\delta(\{q_0,q_1\},0) = \delta(q_0,0) \cup \delta(q_1,0)$

$= \{q_0,q_1\} \cup \{q_1\} = \{q_0,q_1\}$

$\delta(\{q_0,q_1\},1) = \delta(q_0,1) \cup \delta(q_1,1)$

$= \{q_1\} \cup \{q_1,q_0\} = \{q_0,q_1\}$

δ	0	1
$\{q_0\}$	$\{q_0,q_1\}$	$\{q_1\}$
$\{q_0,q_1\}$	$\{q_0,q_1\}$	$\{q_0,q_1\}$
$\{q_1\}$		

↑ unfinished.

→ process $\{q_1\}$:-

$\delta(\{q_1\},0) = \phi$

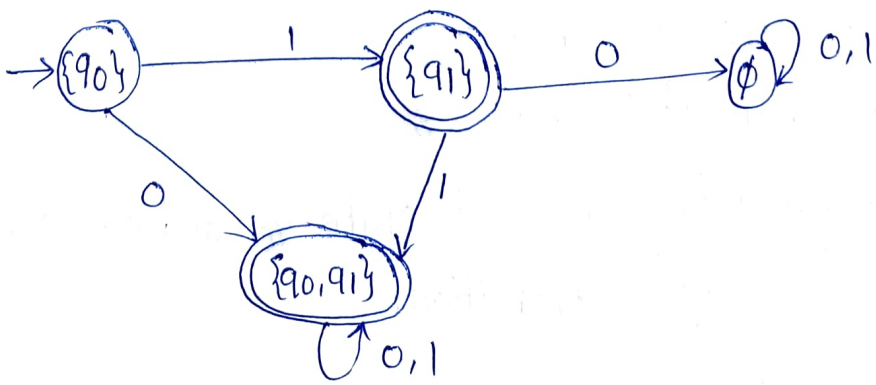
$\delta(\{q_1\},1) = \{q_0,q_1\}$

→ process ϕ :- $\delta(\phi,0) = \delta(\phi,1) = \phi$

δ	0	1
$\{q_0\}$	$\{q_0,q_1\}$	$\{q_1\}$
$\{q_0,q_1\}$	$\{q_0,q_1\}$	$\{q_0,q_1\}$
$\{q_1\}$	ϕ	$\{q_0,q_1\}$
ϕ	ϕ	ϕ

Thus, the equivalent DFA is $D = (Q', \Sigma, \delta', q_0', F')$

where $Q' = \{\{q_0\},\{q_0,q_1\},\{q_1\},\phi\}$, $F' = (\{q_0,q_1\},\{q_1\})$. ~~and~~



Kilavanya

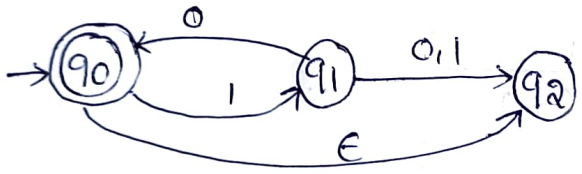
Construction of a DFA equivalent to NFA-ϵ :-

To obtain the DFA from the given NFA-ϵ, the subset construction algorithm is used. The following points have to be noted for ϵ-transitions.

→ Start state q_0 consists of start state q_0 and all states, reachable from q_0 by ϵ-transitions.

→ For any state processing q', add all transitions from q' reachable with ϵ.

Ex :- Construct DFA, equivalent to the NFA given in figure



Sol :- The transition table for NFA-ϵ is given below

δ	0	1	ϵ
q_0	ϕ	$\{q_1\}$	$\{q_2\}$
q_1	$\{q_0, q_2\}$	$\{q_2\}$	ϕ
q_2	ϕ	ϕ	ϕ

No. of states NFA-ϵ = 3

Possible DFA states = $2^3 = 8$

Consider, q_0 consisting of start state q_0 and all states reachable from q_0 by ϵ-transition.

$\Rightarrow q_0 = \{q_0, q_2\} \Rightarrow$

δ	0	1
$\{q_0, q_2\}$		

← unfinished

→ process $\{q_0, q_2\}$:- add all transitions from $\{q_0, q_2\}$ reachab with ε

$\delta(\{q_0, q_2\}, 0) = \delta(q_0, 0) \cup \delta(q_2, 0) = \phi$

$\delta(\{q_0, q_2\}, 1) = \delta(q_0, 1) \cup \delta(q_2, 1) = \{q_1\}$

δ	0	1
$\{q_0, q_2\}$	ϕ	$\{q_1\}$
ϕ	ϕ	ϕ
$\{q_1\}$		

← unfinished state

→ process $\{q_1\}$:- add all transitions from q_1, reachable with ϵ.

$\delta(q_1, 0) = \{q_0, q_2\}$

$\delta(q_1, 1) = \{q_2\}$

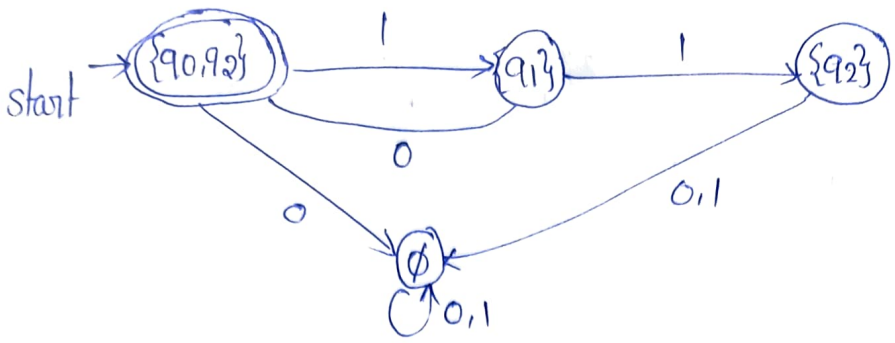
δ	0	1
$\{q_0, q_2\}$	ϕ	$\{q_1\}$
ϕ	ϕ	ϕ
$\{q_1\}$	$\{q_0, q_2\}$	$\{q_2\}$
$\{q_2\}$	ϕ	ϕ

→ process $\{q_2\}$:- add all transitions from q_2, reachable from ϵ

$\delta(q_2, 0) = \delta(q_2, 1) = \phi$

Thus, the equivalent DFA is $D = (Q', \Sigma, \delta', q_0', F')$

where $Q' = (\{q_0, q_2\}, \phi, \{q_1\}, \{q_2\})$, $F' = (\{q_0, q_2\})$ and δ' is shown below.



Equivalence of NFA, with ε moves to NFA without ε-moves :-

Two finite automata N_ϵ and N are said to be equivalent if

$$L(N_\epsilon) = L(N)$$

where N_ϵ represents an NFA with ε-moves and N represents NFA without ε-moves.

This means that for any language described by some N_ϵ, there is an N that accepts the same language.

Theorm :- For every NFA, with-ε-moves, there exists an NFA, without ε-moves, that accepts the same language. Alternatively if L is accepted by an NFA with ε-moves, then L is accepted by an NFA without ε-moves.

Proof :-

NFA with ε-moves defined by the 5 tuple

$$N_\epsilon = (Q, \Sigma, \delta, q_0, F)$$

Q – Finite set of states q_0 – initial state δ – $Q \times \Sigma \cup \{\epsilon\} \to 2^Q$

Σ – set of i/p symbols F – Final state

Minimization / optimization of DFA

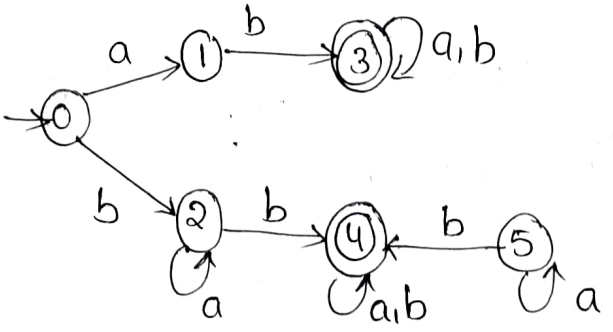
optimum DFA :- Minimization / optimization of a deterministic finite automata refers to the detection of those states of DFA, whose presence (or) absence in a DFA does not affect the language accepted by the automata.

The states that can be eliminated from automata without affecting the language accepted by automata are:

- → unreachable (or) inaccessible states
- → Dead states
- → Non distinguishable / indistinguishable / equivalent states.

1) **unreachable states :-** these are the states that cannot possibly be reached from the initial state. unreachable states of a DFA are not reachable from the initial state of DFA, by any possible input sequences.

Ex :-



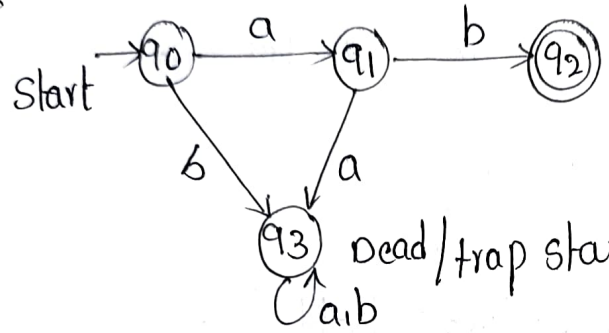
Here state 5 is unreachable, from the initial state with any input string (either b (or) a).

K. Lavanya

2. Dead state (or) Trap state :- A state is dead, if it is not an accepting state and has no out going transitions, expect to itself. Alternatively, a dead state is a nonfinal state of a DFA, whose transitions on every input symbol terminates on itself.

Formally, 'q' is a dead state, if q is in Q and $\delta(q,a)=q$ for every 'a' in Σ.

Ex :-



3. Indistinguishable states :- states are said to be indistinguishable, if their merger does not change the language accepted by a DFA; otherwise states are distinguishable.

Formally they are defined as,

→ Indistinguishable : Two states p and q of a DFA are called indistinguishable,

if $\delta(p,w) \in F \Rightarrow \delta(q,w) \in F$

and $\delta(p,w) \notin F \Rightarrow \delta(q,w) \notin F, \forall w \in \Sigma^*$

→ distinguishable : Two states p and q of a DFA are called distinguishable.

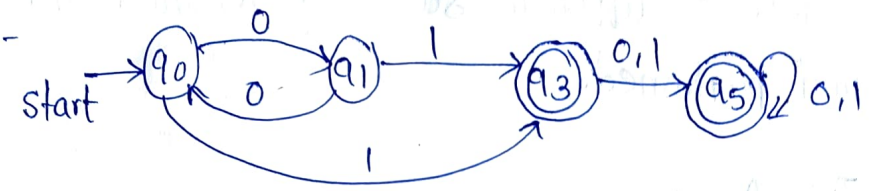
if $\delta(p,w) \in F$ and $\delta(q,w) \notin F$

and $\delta(p,w) \notin F$ and $\delta(q,w) \in F$, $\forall w \in \Sigma^*$

→ DFA's are equivalent (or) indistinguishable, if every string w leads from p to a final state, if and only if, it also leads from q to a final state.

→ DFA'S are non-equivalent (or) distinguishable, we need to find only one string that leads from one of them to a final states and leads from other to a non final states.

Ex :-



Consider two states of the DFA, q_0 and q_1, with $Q = \{q_0, q_1, q_3, q_5\}$ and $F = \{q_3, q_5\}$.

→ input 1 $\quad \delta(q_0,1) = q_3 \in F$

$\delta(q_1,1) = q_3 \in F$

→ input 0 $\quad \delta(q_0,0) = q_1 \notin F$

$\delta(q_1,0) = q_0 \notin F$

Here for every string 1, leads from q_0 to the final state q_3 and also 1 leads from q_1 to the final state q_3, the two states (q_0, q_1) are called indistinguishable (or) equivalent states.

Proadure to detect indistinguishable state :-

Step1 :- partition $\pi = (F, Q-F)$, where F is the set of final states and Q is the set of states.

Step2 :- perform partition π_{new} into subsets of π such that,

→ for any input symbols a and b, if a transition is made to states of the same subset of π (or) final states, then no partitions are made further.

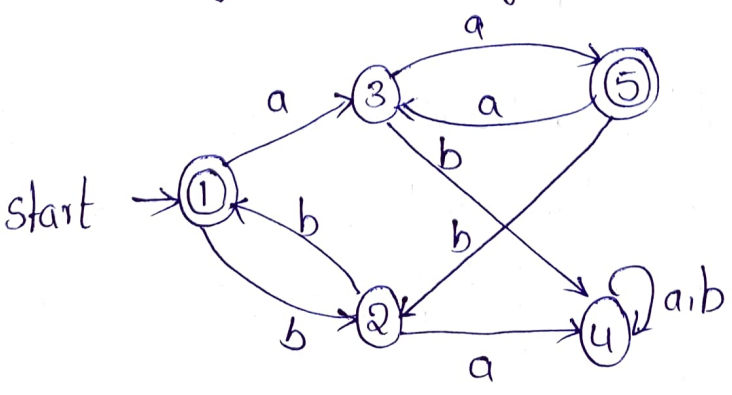
→ For any input symbol a and b, if a transition is made to a state of different subset of π, then the subset is further partioned.

Step3 :- $\pi = \pi_{new}$

Step4 :- Repeat step-1 through step-3, untill $\pi_{new} \neq \pi$.

Step5 :- if numbers of the subset π on input string 'a' (or) 'b' go for the same transition states, then the states are indistinguishable.

Ex:- identify the indistinguishable states from the DFA,



Sol:→ $\pi = (\{2,3,4\}, \{1,5\})$

Partitions : [2 3 4] [1 5]

form π_{new}, with subset $\{2,3,4\}$:

input = a, $\delta(2,a) = 4$
$\delta(3,a) = 5$
$\delta(4,a) = 4$

since for the input a, state 3 gives a transition to a state of other number in the subset $\{1,5\}$, 3 is partioned.

$\pi_{new} = (\{2,4\}, \{3\}, \{1,5\})$

$\pi = \pi_{new}$.

→ $\pi = (\{2,4\}, \{3\}, \{1,5\})$

Partitions : [2 4] [3] [1 5]

form π_{new}, with subset $\{2,4\}$:

Input = a, $\delta(2,a) = 4$
$\delta(4,a) = 4$

Input = b, $\delta(2,b) = 1$
$\delta(4,b) = 4$

since for the input b, state 2 gives a transition to a state of the other number $\{1,5\}$, 2 is partioned.

$\pi_{new} = (\{2\}, \{4\}, \{3\}, \{1,5\})$

$\pi = \pi_{new}$

→ $\pi = (\{2\}, \{4\}, \{3\}, \{1,5\})$

partitions : [2] [4] [3] [1 5]

form π_{new}, with subset {1,5} :

Input = a, $\delta(1,a) = 3$
$\delta(5,a) = 3$

Input = b, $\delta(1,b) = 2$
$\delta(5,b) = 2$

since for the inputs a,b, states 1 and 5 give the transition to the same state, they are not partioned.

hence, $\pi_{new} = (\{2\}, \{4\}, \{3\}, \{1,5\})$

$\Rightarrow \pi_{new} = \pi$ is true. i.e $\pi_{new} \neq \pi$ is false

Thus, states 1, and 5, on the inputs a and b, give transitions to same non final states. Hence 1 and 5 are called Indistinguishable states.

Minimal DFA :- A DFA is minimal if and only if,
→ all it's states are reachable from the start state
→ all it's states are distinguishable.

Minimization Algorithm for DFA :- Let $M = <Q, \Sigma, \delta, q_0, F>$ be a DFA that accepts a language L. Then the following algorithm produces the DFA, that has the smallest no. of states among all the DFA's, that accept L.

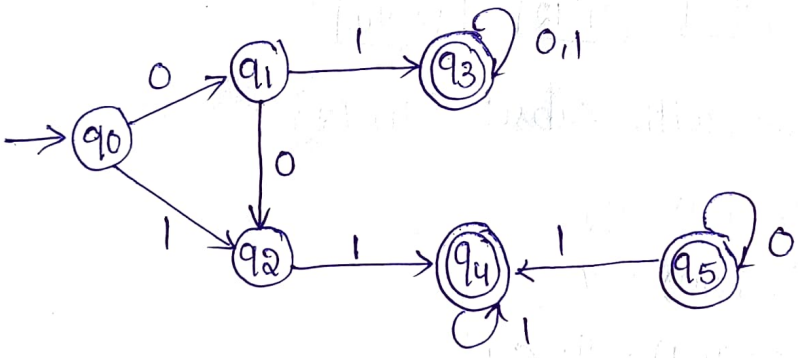
Step1 :- Identify all unreachable (or) inaccessible states and eliminate them from the DFA, M.

Step2 :- Identify all indistinguishable states from the DFA and merge them all to form the DFA with smallest no. of states.

Step3 :- Construct a DFA from π final

Step4 :- END.

Example :- Minimize the DFA.



Sol:-

→ unreachable state :- if we enumerate all simple paths starting from the initial state, then we find that the state q_5 is said to be unreachable, so, it is removed.

→ Indistinguishable state :-

→ $\pi = (\{q_0, q_1, q_2\}, \{q_3, q_4\})$

partitions : [q0 q1 q2] [q3 q4]

Form π_{new}, with subset $\{q_0, q_1, q_2\}$.

→ input = 0, $\delta(q_0, 0) = q_1$

$\delta(q_1, 0) = q_2$

$\delta(q_2, 0) = \phi$

→ input = 1, $\delta(q_0, 1) = q_2$

$\delta(q_1, 1) = q_3$

$\delta(q_2, 1) = q_4$

since for the input '1' states q_1 and q_2 give transitions to the states of other number $\{q_3, q_4\}$, $\{q_1, q_2\}$ are partitioned.

→ $\pi_{new} = (\{q_0\}, \{q_1, q_2\}, \{q_3, q_4\})$

Partition: | q_0 | q_1 q_2 | q_3 q_4 |

Form π_{new}, with subset $\{q_1, q_2\}$:

Input=0, $\delta(q_1, 0) = q_2$

$\delta(q_2, 0) = \phi$

Input = 1 $\delta(q_1, 1) = q_3 \in F$

$\delta(q_2, 1) = q_4 \in F$

on input 1, q_1 and q_2 gives transitions to states of other numbers $\{q_3, q_4\}$. Also both q_3 and q_4 belong to final states, thus q_1 and q_2 are not partitioned. Hence they are indistinguishable.

Forming π_{new} with subset $\{q_3, q_4\}$:

Input = 0 $\delta(q_3, 0) = q_3$

$\delta(q_4, 0) = \phi$

Input = 1, $\delta(q_3, 1) = q_3 \in F$

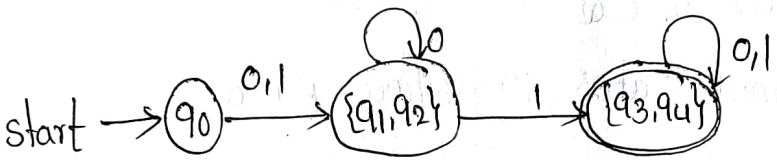
$\delta(q_4, 1) = q_4 \in F$

since for the input '1', q_3 and q_4 belonging to the same subset members, they are not partitioned and are indistinguishable.

$\Rightarrow \quad \pi \neq \pi_{new}$ is false

merging all indistinguishable states leads to a minimum DFA, Now the DFA is,

:- Minimized DFA.



Two-way DFA :- one way is a mathematical model of a machine with a finite amount of memory, where the input is processed once from left to right. After the input has been read, the DFA decides whether the input is to be accepted (or) rejected.

A two way DFA consists of a finite state control and a read only input tape that allows an i/p to be read back and forth. As in case of DFA, the 2DFA decides whether a given input is to be accepted (or) rejected.

A mathematical model of machine, with the ability of a read-head to move left as well as right is a two way DFA.

Elements of 2DFA :- the 2DFA exhibits the following five characteristics.

→ a finite set of states Q,
→ An alphabet Σ of possible input symbols.
→ a transition function δ
→ the initial state $q_0 \in Q$
→ the set of final states (F), where $F \subseteq Q$.

Ordered quintuple specification of 2DFA :-

Formally, a 2DFA M is a five tuple $M = (Q, \Sigma, \delta, q_0, F)$, where

→ Q is finite set of states
→ Σ is a finite set of symbols
→ $\delta : Q \times \Sigma \to Q \times \{L, R\}$
→ $q_0 \in Q$ is the start state
→ $F \subseteq Q$ is the set of accept state.

the transition function δ is a map from

$Q \times \Sigma \to Q \times \{L, R\}$

Ω - Transducers

Moore machine :- A moore machine is a finite state automata where the outputs are determined by the current state alone.

A moore machine associates an output symbol with each state, and each time a state is entered, an output is obtained simultaneously. So the output always occurs as soon as the machine starts.

Elements of moore machine :-

Formally, the moore machine M_0 is represented by 6-tuples

$$M_0 = (Q, \Sigma, \delta, q_0, \Gamma, \lambda)$$

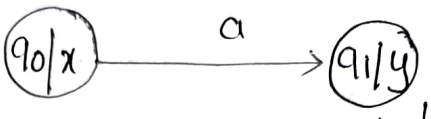
where,

- → Q is a finite set of states.
- → Σ is a finite set of input symbols.
- → $\delta : Q \times \Sigma \to Q$ is the transition function.
- → $q_0 \in Q$ is the initial state.
- → Γ is a finite set of output symbols.
- → $\lambda : Q \to \Gamma$ is the output function.

Description of moore machine :- The transitions of moore machine can be represented using the transition diagram and transition table.

1) Transition diagram :- the transition diagram for a moore machine will include the output for each state. Each circle of a transition diagram is labelled with a compound symbol q_i/x, to indicate that $\delta(q_i, a) = q_j$ and $\lambda(q_i, a) = x$.

Ex :-



Here the output is associated with a state q_0 is x, then it is written as q_0/x inside the circle.

Transition table :- In moore machine, every state is associated with output, the transition table called transition and output table.

The rows of the table corresponds to states and columns correspond to inputs and output.

Ex :- for the above Example, the transition table is

Σ / Q	inputs a	output (Γ) (x)
q_0	q_1	x
q_1	-	y

Design of moore machine :-

Basic design strategy for moore machine is

→ understand the problem definition for which moore machine tobe designed.

→ Determine the required alphabet and state set.
→ Determine the required output set.
→ For each state, decide on the transition to be made for each character of the input string.
→ For each state, decide the required output.
→ obtain the transition table and diagram.
→ Test the moore machine obtained on short strings.

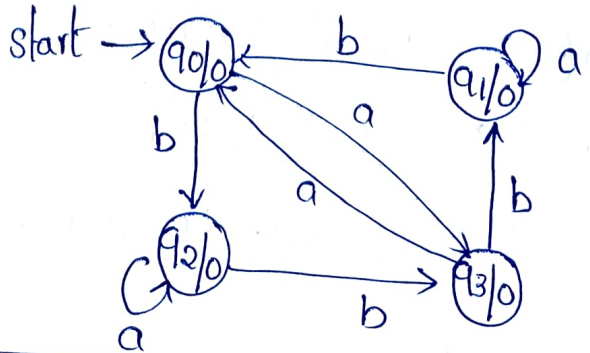
Ex :- Construct a moore machine for the following,

input alphabet $\Sigma = \{a, b\}$
output alphabet $\Gamma = \{0, 1\}$
states $Q = \{q_0, q_1, q_2, q_3\}$

Q \ Σ	a	b	Γ
q_0	q_3	q_2	0
q_1	q_1	q_0	0
q_2	q_2	q_3	1
q_3	q_0	q_1	0

Sol :- Given $\Sigma = \{a, b\}$ $\Gamma = \{0, 1\}$ $Q = \{q_0, q_1, q_2, q_3\}$

the moore machine is,



No action for the input string :-

For the input string $s = ba\ babbb$ where $n = 7$ what is the output string ?

$$\lambda(\delta(q_0, \epsilon)) = \lambda(q_0) = 0$$

$$\lambda(\delta(q_0, b)) = \lambda(q_2) = 1$$

$$\lambda(\delta(q_0, ba)) = \lambda(\delta(\delta(q_0, b), a)$$
$$= \lambda(\delta(q_2, a)$$
$$= \lambda(q_2) = 1$$

$$\lambda(\delta(q_0, bab)) = \lambda(\delta(\delta(q_0, ba), b)$$
$$= \lambda(\delta(q_2, b)$$
$$= \lambda(q_3) = 0$$

$$\lambda(\delta(q_0, baba)) = \lambda(\delta(\delta(q_0, bab), a)$$
$$= \lambda(\delta(q_3, a)$$
$$= \lambda(q_0) = 0$$

$$\lambda(\delta(q_0, babab)) = \lambda(\delta(\delta(q_0, baba), b)$$
$$= \lambda(\delta(q_0, b)$$
$$= \lambda(q_2) = 1$$

$$\lambda(\delta(q_0, bababb)) = \lambda(\delta(\delta(q_0, babab), b)$$
$$= \lambda(\delta(q_2, b)$$
$$= \lambda(q_3) = 0$$

$$\lambda(\delta(q_0, bababbb)) = \lambda(\delta(\delta(q_0, bababb), b)$$
$$= \lambda(\delta(q_3, b) = \lambda(q_1) = 0.$$

Thus for the input string s, the output is 0110010 0

$(n = n+1)$.

Mealy machine :- A mealy machine is a finite state machine, where the outputs are determined by the current state and the input.

A mealy machine associates an output symbol with each transition and the output depends on the current input.

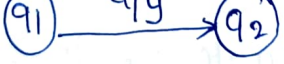
Elements of mealy machine :- Formally, the mealy machine M_e is represented by 6-tuples.

$$M_e = (Q, \Sigma, \delta, q_0, \Gamma, \lambda).$$

Description of mealy machine :-

1) **Transition diagram :-** The transition diagram for mealy machine will include the output for each transition edge. Each edge is labelled with a compound symbol a/b from state p to state q to indicate the $\delta(p,a) = q$ and $\lambda(p,a) = b$

Ex :-



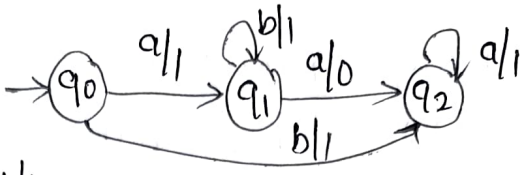
if the o/p associated with edge labelled with symbol 'a' is 'y', then it is written as a/y on that edge.

2) **Transition Table :-** The transition table for mealy machine consists of two sub-tables.

a) **Transition table :-** Rows correspond to states and columns correspond to inputs. Entries correspond to next states.

b) output table :- Rows correspond to states and columns correspond to inputs. Entries correspond to output.

Ex :-



The transition table for the above transition diagram is,

Q \ Σ	a	b
q_0	q_1	q_2
q_1	q_2	q_1
q_2	q_2	–

Transition Table

Q \ Γ	a	b
q_0	1	1
q_1	0	1
q_2	1	–

output table

Design of mealy machine :-

1) understand the problem definition for which Me is required.
2) Determine the required alphabet, state and o/p set.
3) For each state, decide the transition to be made for i/p string.
4) Decide the output to be associated with each edge label.
5) obtain the transition table, output table and diagram.
6) Test the mealy machine obtained on short strings.

Example :- Construct a mealy machine for the following.

$\Sigma = \{0,1\}$ $\Gamma = \{y,n\}$ states $Q = \{q_0, P_0, P_1\}$

Now the transition and output tables are

Σ / Q	0	1
q_0	P_0	P_1
P_0	P_0	P_1
P_1	P_0	P_1

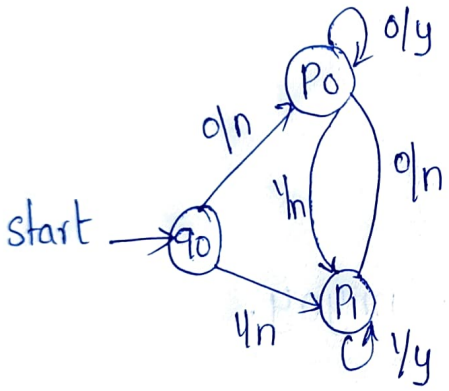
Γ / Q	0	1
q_0	n	n
P_0	y	n
P_1	n	y.

Sol:- Let $M_e = (Q, \Sigma, \delta, q_0, \Gamma, \lambda)$ be the mealy machine

Given $\Sigma = \{0, 1\}$

$Q = \{q_0, P_0, P_1\}$

$\Gamma = \{y, n\}$ at $q_0 = q_0$ the functions δ and λ are



Me action for the string:- For the i/p string 01100, what is the o/p string and the sequence of states entered.

$\delta(q_0, 0) = P_0$

$\delta(q_0, 01) = \delta(\delta(q_0, 0), 1)$

$= \delta(P_0, 1) = P_1$

$\delta(q_0, 011) = \delta(\delta(q_0, 01), 1)$

$= \delta(P_1, 1) = P_1$

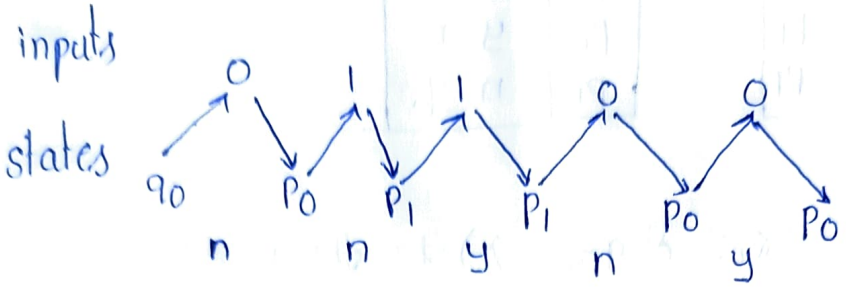
$\delta(q_0, 0110) = \delta(\delta(q_0, 011), 0)$

$= \delta(P_1, 0) = P_0$

K. Lavanya

$\delta(q_0, t001100) = \delta(\delta(q_0, 0110), 0)$

$= \delta(P_0, 0) = P_0$



sequence of states entered for input 01100 is $q_0 P_0 P_1 P_1 P_0 P_0$. From the sequence state diagram:

$\lambda(q_0, 0) = n$

$\lambda(P_0, 1) = n$

$\lambda(P_1, 1) = y$

$\lambda(P_1, 0) = n$

$\lambda(P_0, 0) = y$

Thus output sequence = nnyny.

Difference b/w moore and mealy machines :-

Moore machine	mealy machine
1) It gives output $\lambda(q_0)$ in response to the input ϵ, so the o/p sequence is n+1, not n.	1) It does not give output $\lambda(q_0)$ in response to the input ϵ, so the o/p sequence is n, not n+1.
2) moore machine has actions associated with states (or) it prints characters when in state.	2) It has actions associated with transitions (or) it prints characters, when traversing an arc.

3) The output of this machine depends only on the current state and does not depend on current input.

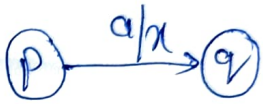
4) In moore machine, if output associated with state q is x, then q/x written inside the circle of transition diagram.

5) the transition diagram of this machine is.

3) The output of this machine depends only on the current input.

4) In mealy machine, if o/p associated with the edge labelled with the letter a is x, it is written as a/x on that edge.

5) the transition diagram of this machine is,



→ Advantages of Finite automata :-

→ their simplicity makes it easy for inexperienced developers to impliment them with no extra knowledge.

→ predictability : given set of inputs and a known current state, the state transition can be predicted.

→ Due to their simplicity, FSM's are quick to design, implement and execute.

→ FSM is an old knowledge representation and system modeling technique.

→ FSM's are relatively flexible.

→ It enables easy transfer from a meaningful abstract representation to coded implimentation.

K. Lavanya

→ Low processor overhead makes it well suited to domains where execution time is shared between modules (or) systems.
→ It enables easy determination of the 'reachability' of a state, when represented in an abstract form.

Disadvantages of FSM :-

→ The predictable nature of deterministic finite state machine can be unwanted in some domains such as Computer games.
→ Larger systems, implimented using a FSM, can be difficult to manage and maintain without a well thought out design.
→ unsuitable for all problem domains.
→ The conditions for state transitions are rigid.

Applications of FSM :-

→ FSM's are extensively used in the video game industry.
→ Besides controling bots, dialogue and environmental conditions within video games
→ FSMs also have a large role outside the video game industry.
→ The applications of FSM's are also found in language Processing : parsing, morphological representations etc.
→ The cellular automata are used for making pretty pictures and animations.
→ FSM'S applied to biological and bio medical problem solving, is the most significant improvement in the recent years.