Formal language and Automata theory

Computation :- The process of solving a problem to obtain a result is called computation.

→ The computation process can be represented by using Mathematical models.

Types of mathematical models :-

The Mathematical models are four types are there in given below.

* finite automata
* push down automata
* linear Bounded automata
* Turing Machine.

finite automata :- finite automata is understanding only one language that language is called Regular language (RL) followed by with the help of regular grammer (RG)

Push down automata :- push down automata is understanding by particular language this language is known as context free language (CFL) is followed by with the understand by the also regular language.

linear bounded automata :- linear bounded automata is understand for language is context sensitive language (CSL) and formed by the with the help of context sensitive grammer (CSG) and long with the understand by two more languages are gRL and CFL Language.

Turing Machine :- Turing Machine is understand for language is recursive enumerable language (REL) and is followed by with the help of grammer is recursive enumerable grammer (REGL) and also along with the understand by three more language RL & CFL & CSL language.

Relation btw automata :-

The relation btw above four language and grammers

$$RL \subseteq CFL \subseteq CSL \subseteq REL$$

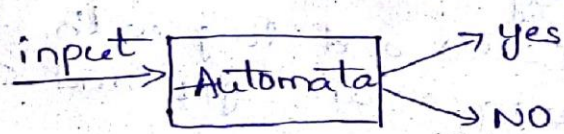
$$RG \subseteq CFG \subseteq CSG \subseteq REG$$

Why study automata theory :-

* This theory is a fundamental course of compu... science.
* Automata theory is the study of abstract mach... and automata as well as the competational prob... that can be solved using them.
* Automata theory will help you understand how people have thought about computer science as a science.
* Automata theory is mainly about
  1. what kind of things can you really compute mechanically.
  2. how fast it take to do it (time complexity)
  3. how much space does it take to do it (space com)

Eg :- 1. Binary strings ends with 0?
  1. 10101101$\underline{0}$ → accepted
  2. 10100010$\underline{1}$ → Rejected.

Eg : 2 Declaration statement in c language like
int a,b,c;



The central concepts of Automata theory

Basic concepts :- 1. symbol
2. Alphabet
3. strings
4. languages.

1. Any formal language can be constructed by the basic concepts of automata theory.
2. Basic concepts of building blocks of automata theory.
   1. symbol :- Symbol is an obj (or) a thing
   Eg : a,b,c,d ----
   0,1,2,3 ----

#, *, +, -, @ - - -

symbols are used to form a string.

2. Alphabet :- It is a non empty and finite set of symbols

* It is denoted by Σ

Eg :- $\Sigma = \{a, b, \dots z\}$

$\Sigma = \{A, B, \dots z\}$

$\Sigma = \{0, 1, 2, \dots 9\}$

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

$\Sigma = \{a, b, c, \dots z, 0, 1, 2, \dots 9\}$

3. string :- It is a sequence of symbols from Sigma

Eg :- $S_1 = abc$

$S_2 = 010$

$S_3 = a, b, c$

$S_4 = 01234$

Length of a string :- The no. of symbols appears in a given string S is called of length of S.

* It is denoted by $|s|$

Eg :- $S_1 = abc$ then $|S_1| = 3$

$S_2 = 010$ then $|S_2| = 3$

$S_3 = a, b, c$ then $|S_3| = 1$

$S_4 = 01234$ then $|S_4| = 5$

Language :- It is a collection of strings over sigma (Σ)

Eg :- let $\Sigma = \{0, 1\}$

L_1 = set of strings have length of 2

$L_1 = \{00, 01, 10, 11\}$ → finite set.

Eg :- L_2 = set of strings have length of 3

$L_2 = \{000, 001, 010, 01, 100, 101, 110, 111\}$ → finite set.

Eg :- L_3 = set of strings ends with 0

$L_3 = \{000, 00, 0, 10, 1010, 11110 \dots\}$ → Infinite set

power of Σ :- $\Sigma\{0, 1\}$

NULL string :- A string without symbol is a null string.

It is denoted by ϵ

∴ length of null string $|\epsilon| = 0$

Σ^0 set of all strings length 0 = {ϵ}

Σ^1 set of all strings length 1 = {0,1}

Σ^2 set of all strings length 2 = {00, 01, 10, 11}

Σ^3 set of all strings length 3 = {000, 001, 010, 011, 100, 110, 111}

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \text{-----} \rightarrow$ star closure :- set of all strings including null string.

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \text{-----} \rightarrow$ positive closure :- excluding NULL string.

1. $\Sigma^* = \Sigma^0 \cup \Sigma^+$

$= \epsilon \cup \Sigma^+$

$\boxed{\Sigma^* = \epsilon \cup \Sigma^+}$

$\Sigma^+ = \Sigma^* - \{\epsilon\}$

$\boxed{\Sigma^+ = \Sigma^* - \epsilon}$

String operations :-

1. length of a string :- It means no. of symbols in the given string 's'. It is denoted by |s|.

Eg :- let s = "google" be a string over $\Sigma = \{a, b, c, \text{----} z\}$

length s ⇒ |s| = 7

2. Position of a symbol in string :- consider an input symbol "a" in the input string 's' the position "a" in s is "q" then it is denoted by $s_a(i)$

$\boxed{\begin{matrix} s_a(i) = 1 \\ = 0 \end{matrix}}$ if a is in i^{th} position of s otherwise.

Eg :- Consider an input symbol g in s then

$s_g(1) = 1$ $s_g(4) = 1$ $s_g(7) = 0$

$s_g(2) = 0$ $s_g(5) = 1$

$s_g(3) = 0$ $s_g(6) = 0$

3. Concatenation :- It means combine two (or) more strings into a single string.

→ mathematically if s_1 & s_2 are two strings then the concatenation s' of s_1 & s_2 is given by

$s' = s_1 \circ s_2$

$= \{xy \mid x \in s_1, y \in s_2\}$

$= \{yx \mid y \in s_1, x \in s_2\}$

Eg:- If s_1 = para and s_2 = graph then $s_1 o s_2$

$\therefore s_1 o s_2$ = paragraph

Eg:- If $\Sigma = \{a, b\}$, $s_1 = ab$

$s_2 = baa$

then $s' = s_1 o s_2 = abbaa$

Eg:- let $x = 0100101$ and $y = 1111$

$x o y = 0100101111$

Eg:- $abba\, o\epsilon = abba|\epsilon| = 0$

Note:- Eg:- let z be any string then $z o \epsilon = \epsilon o z = z$.

Identity rule.

Note:- Associativity rule:- $a(bc) = ab(c)$

$a o (boc) = (aob) oc$

4. Reverse of a string:- Reverse of a string means, Reverse the symbols in a string.

→ It is denoted by s^R where s is given string

Eg:- 1. if s = mus then s^R = Sum.

2. let s = bottle then s^R = elttob and $(s^R)^R$ = bottle

3. let $|s| = |s^R|$.

let x = K lim and y = nettub then $(x o y)^R$

$x o y$ = K lim nettub

$(x o y)^R$ = buttermilk.

$y^R o x^R = y^R$ = butter x^R = milK.

$y^R o x^R$ = buttermilk $[\because (x o y)^R = y^R o x^R$

Kleene closure (or) star closure:-

Kleene closure is introduced by Kleene mathematician

It is a set which contains all strings including empty string (or) null string.

It is denoted by s^*

It is used for regular expression

$s^* = \{s^0, s^1, s^2, s^3, s^4 \cdots\}$

$$\boxed{s^* = s^0 \cup s^1 \cup s^2 \cup s^3 \cup \cdots}$$

Eg:- if $s = \{a\}$ then find s^*

$s^* = s^0 \cup s^1 \cup s^2 \cup s^3 \cup \cdots$

$s = \{a\}$

$S^0 = \{\epsilon\}$ $S^3 = \{aaa\}$

$S^1 = \{a\}$ $S^4 = \{aaaa\}$

$S^2 = \{aa\}$

$S^* = S^0 \cup S^1 \cup S^2 \cup S^3 \cup S^4 \cup \cdots$

$= \{\epsilon\} \cup \{a\} \cup \{aa\} \cup \{aaa\} \cup \{aaaa\} \cup \cdots$

$= \{\epsilon, a, aa, aaa, aaaa\}$

Eg:- if $S = a, b$ then find S^*

$S = \{a, b\}$

$S^* = S^0 \cup S^1 \cup S^2 \cup S^3 \cup \cdots$

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

$S^1 = \{a\}\ \{b\} = \{a, b\}$

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

$S^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

$S^* = S^0 \cup S^1 \cup S^2 \cup S^3 \cup \cdots$

$= \{\epsilon, a, b, aa, bb, ab, ba, aaa, aab, aba, abb, baa, bbb\}$

Eg:- if $S = \{cc, d\}$ then find S^*

$S^* = S^0 \cup S^1 \cup S^2 \cup S^3 \cdots$

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

$S^1 = \{d\}$

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

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

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

$\therefore S^* = S^0 \cup S^1 \cup S^2 \cup S^3 \cup \cdots$

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

Positive closure:- It was introduced by Kleen mathematician

-> It is a set which contains all strings excluding null (or) empty string.

-> Denoted by S^+

-> used for regular Expression

$$\therefore S^+ = S^1 \cup S^2 \cup S^3 \cup \cdots$$

Eg :- 1) if $S=\{a\}$ then S^+

$S^+ = S^1 \cup S^2 \cup S^3 \cup S^4 \cup \cdots$

$S^1 = \{a\}$ $\quad S^4 = \{aaaa\}$

$S^2 = \{aa\}$

$S^3 = \{aaa\}$

$S^+ = S^1 \cup S^2 \cup S^3 \cup S^4 \cup \cdots$

$= \{a, aa, aaa, aaaa\}$

2) if $S=\{a, ba\}$ then find S^+

$S^1 = \{a\}$

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

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

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

$S^+ = S^1 \cup S^2 \cup S^3 \cup S^4 - \cdots$

$= \{a, aa, ba, aaa, baa, aba, aaaa, aaba, baaa, baba\}$

Note :- Relation ship btw S^* and S^+

$S^* = S^0 \cup S^+$ $\quad S^+ = S^* - S^0$

$\boxed{S^* = \epsilon \cup S^+}$ $\quad \boxed{S^+ = S^* - \epsilon}$

Parts of a string :- 1. prefix of a string
2. Suffix of a string
3. proper prefix of a string
4. proper suffix of a string.

1) prefix of a string :- The number of leading symbols of given string.

⇒ let 'x' be a string then prefix of 'x' is denoted by prefix(x).

Eg :- if $x=abc$ is a string then prefix of x is

$prefix(x) = \{\epsilon, a, ab, abc\}$

2) Suffix of a string :- The number of trailing symbols in a given string.

⇒ It is denoted by suffix(x)

Eg :- if $x=abc$ is a string then suffix of x is

$suffix(x) = \{\epsilon, c, bc, abc\}$

3) proper prefix of a string :- The no. of leading symbols except the given string.

→ It is denoted by proper prefix (x)

Eg:- if x: abc is a string then proper prefix (x)

proper prefix (x) = {∈, a, ab}

4) proper suffix of a string:- The no. of trailing symbols except the given string.

→ It is denoted by proper suffix (x)

Eg:- If x = abc is a string then proper suffix (x)

proper suffix (x) = {∈, c, bc}

Language:-

* Introduction * operations of language.

Introduction:- A language is a finite and non empty set of strings. It is denoted by L.

→ All these strings are formed from the alphabet Σ.

language notation L(M):- is a language defined by machine M that accepts a set of strings.

→ L(G) is a language defined by the Grammer "G" that recognise a set of strings.

→ L(r) is a language defined by the regular expression that represent a set of strings.

Eg:-

1) let Σ = {z} then the language of all possible strings is given by Σ = {z}

L = {∈, z, zz, zzz, zzzz}

2) The language of all possible of even length strings over Σ = {z}

$L = \{z^0, z^2, z^4, z^6, z^8 ---\}$

$L = \{z^{2n} / n \geq 0\}$

Eg:- The language of all possible strings of odd length over. Σ = {z}

$L = \{z^1 z^3 z^5 z^7 z^9 ---\}$

$L = \{ z^{2n-1} / n \geq 1 \}$

operations on language :-

1. Union
2. Intersection
3. Complementation
4. symmetric difference.
5. Reversal of language
6. polindrome language
7. Kleene closure (or) star closure of language.
8. Demorgan's law.

1) union :- It is a simple operation on two languages.

→ let L_1 & L_2 be two languages then the union is defined by $L_1 \cup L_2$.

→ Mathematically the union of two languages is defing as $L_1 \cup L_2 = \{ x / x \in L_1 \text{ or } x \in L_2 \}$

Eg :- let $L_1 = \{0, 01, 011\}$ and $L_2 = \{\epsilon, 001, 1^5\}$ then $L_1 \cup L_2 =$

$\{\epsilon, 0, 01, 011, 001, 1^5\}$

2) let $L_i = 2^i$ then $\bigcup_{i=0}^{7} L_i =$

$\bigcup_{i=0}^{7} L_i = L_0 \cup L_1 \cup L_2 \cup L_3 \cup L_4 \cup L_5 \cup L_6 \cup L_7$

$= 2^0 \cup 2^1 \cup 2^2 \cup 2^3 \cup 2^4 \cup 2^5 \cup 2^6 \cup 2^7$

$= \{\epsilon\} \cup \{2\} \cup \{22\} \cup \{222\} \cup \{2222\} \cup \{22222\} \cup \{222222\}$
$\cup \{2222222\}$

$\{\epsilon, 2, 22, 222, 2222, 22222, 222222, 2222222\}$

2) Intersection :- It is a simple operation on two languages. and l_1 & l_2 be two languages and their intersection is denoted by $L_1 \cap L_2$

→ Mathematically intersection of L_1 & L_2 is defined as

$L_1 \cap L_2 = \{ x / x \in L_1 \text{ and } x \in L_2 \}$

let $L_1 = \{\epsilon, 00, 0000, 000000, ---\}$ and

$L_2 = \{\epsilon, 0, 000, 00000, ---\}$ then $L_1 \cap L_2$

$L_1 \cap L_2 = \{\epsilon\}$

Let $L_1 = \{\epsilon\}$ and $L_2 = \{\phi\}$ $\therefore L_1 \cap L_2 = \phi$ //.

3) Complementation :-

It is a simple operation performed on single language.

→ let L be a language over Σ then the complementation of L is denoted by $\bar{L}$ (or) L^c therefore $\bar{L}$ or L^c =

$$\boxed{\bar{L} \text{ or } L^c = \Sigma^* - L}$$

→ Mathematically the complementation of L is defined as $\bar{L} = \{x | x \in \Sigma^* \text{ and } x \notin L\}$

Eg:- if $\Sigma = \{a, b\}$ and $L = \{a, b, aa\}$ then L^c = ?

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cdots$

$\Sigma^0 = \{\epsilon\}$

$\Sigma^1 = \{a\} \{b\}$

$\Sigma^2 = \{aa, ab, ba, bb\}$

$\Sigma^3 = \{aaa, aab, baa, aba, bba, bab, bbb, abb\}$

$\therefore L^c = \Sigma^* - L$

$= \{\epsilon, a, b, aaa, ab, ba, bb, aaa, aab, baa, aba, bba, bab, bbb, abb\} - \{a, b, aa\}$

$= \{\epsilon, ab, ba, bb, aaa, aab, baa, aba, bba, bab, bbb, abb\}$

2) let $L = \{\epsilon, 1, 11, 111, 1111 \cdots\}$ over $\Sigma = \{0, 1\}$ then find $\bar{L}$ =

$\bar{L} = \Sigma^* - L$

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

$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \cdots\} - \{\epsilon, 1, 11, 111, 1111 \cdots\}$

$= \{0, 00, 01, 10, 000, 001, 010, 011, 100, 101, 110 \cdots\}$

symmetric difference :-

It is a simple operation on two languages.

→ let L_1 & L_2 be two languages then the symmetric operation on L_1 & L_2 is defined as

$$L_1 \oplus L_2 = (L_1 \cup L_2) - (L_1 \cap L_2)$$

→ Here x is in L_1 or L_2 but not in both.

Eg:- let L be a language over Σ

1) $L \oplus \phi = (L \cup \phi) - (L \cap \phi)$
$= L - \phi$
$= L$

2) $L \oplus L = (L \cup L) - (L \cap L)$
$= L - L$
$= \phi$

3) $L \oplus \Sigma^* = (L \cup \Sigma^*) - (L \cap \Sigma^*)$
$= \Sigma^* - L$
$= \bar{L}$

4) $L \oplus \bar{L} = (L \cup \bar{L}) - (L \cap \bar{L})$
$= \Sigma^* - \phi$
$= \Sigma^*$

Eg:- let $L_1 = \{00, 0000, 000000, \cdots\}$ $L_2 = \{11, 1111, 111111, \cdots\}$ then

$L_1 \oplus L_2 = (L_1 \cup L_2) - (L_1 \cap L_2)$

$= \{00, 0000, 000000 \cdots, 11, 1111, 111111, \cdots\} - \{\phi\}$

$= \{00, 0000, 000000, \cdots, 11, 1111, 111111, \cdots\}$

Eg:- If $\Sigma = \{0, 1\}$ then $\Sigma^{\leq 2} \oplus \{\epsilon, 0, 00, 101, \cdots\}$

$\Sigma^{\leq 2} = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2$

$= \{\epsilon\} \cup \{0, 1\} \cup \{00, 01, 10, 11\}$

$= \{\epsilon, 0, 1, 00, 01, 10, 11\}$

$\Sigma^{\leq 2} \oplus \{\epsilon, 0, 00, 101\}$

$= (\Sigma^{\leq 2} \cup \{\epsilon, 0, 00, 101, \cdots\}) - (\Sigma^{\leq 2} \cap \{\epsilon, 0, 00, 101, \cdots\})$

$= \{\epsilon, 0, 1, 00, 01, 10, 11\} \cup \{\epsilon, 0, 00, 101, \dots\} - \{\epsilon, 0, 1, 00, 01, 10, 11\}$
$\cap \{\epsilon, 0, 00, 101\}$.

Concatenation of language:-

Concatenation of language used to combine two or more languages into a single language. It is denoted by $L_1 \circ L_2$ or $L_1 L_2$.

Mathematically it is denoted by

$$L_1 \circ L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$$

Eg:- i) Let $L_1 = \{bc, bcc, cc\}$ and $L_2 = \{cc, cccc\}$ then

i) $L_1 \circ L_2 = ?$ ii) $L_2 \circ L_1 = ?$

$L_1 \circ L_2 = \{bc, bcc, cc\} \circ \{cc, cccc\}$

$= \{bccc, bccccc, bcccc, bccccc, cccc, ccccc\}$

$L_2 \circ L_1 = \{cc, cccc\} \circ \{bc, bcc, cc\}$

$= \{ccbc, ccbcc, cc\,cc, ccccbc, ccccbcc, ccccc\}$

ii) Let $L_1 = \{0,1\}^*$ and $L_2 = \{0,1\}$ then $L_1 \circ L_2 = ?$

$L_1 = \{0,1\}^*$

$L_1 = L_1^0 \cup L_1^1 \cup L_1^2 \cup L_1^3 \cup L_1^4 \dots$

$L_1^0 = \{\epsilon\}$

$L_1^1 = \{0,1\}$

$L_1^2 = \{00, 01, 10, 11\}$

$L_1^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

$L_1 = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \dots\}$

$L_2 = \{0,1\}$

$L_1 \circ L_2 = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \dots\} \circ \{0,1\}$.

$= \{0, 00, 10, 000, 010, 110, 000, 0010, 0100, 0110, 1000, 1010, 1100, 1110,$
$1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1101, 1111\}$

3) for any language $L \cdot \epsilon = \epsilon \cdot L = L$

4) for any language $L \cdot \phi = \phi \cdot L = \phi$

Reversal of language:-

Language = set of strings.

It is similar to Reverse of strings.

It means Reverse the all strings in that language

It can be denoted by "L^R"

Mathematically it can be defined as $L^R = \{w^R / w \in L\}$

Eg:- 1) If $L = \{a_1 b, a_2 b_2, a_3 b_3\}$ then L^R

$L^R = \{a, b, b_2 a_2, b_3 a_3\}$

2) If $L = \{0, 01, 011\}$ then L^R

$L^R = \{0, 10, 110\}$

3) If $L = \{0^{i, i+1} / i \geq 0\}$ then L^R

$L^R = \{1^{i+1} 0^i / i \geq 0\}$

Kleene closure of a language:-

It is a set of all strings including null string (or) empty string.

It is denoted by L^*

Mathematically it is defined as $L^* = \cup x^i, i \geq 0$

Here, x^0, x^1 is 0 number of x.

positive closure of a language:-

It is a set of all strings excluding NULL string (or) Empty string.

It is denoted by L^+

Mathematically it is defined as $L^+ = L^1 \cup L^2 \cup L^3 \cup \cdots$

$$L^+ = \cup L^i, \; i \geq 1$$

Eg:-1 If $L = \{\epsilon, z, zz, zzz, \cdots\}$ over $\Sigma = \{z\}$ then $L^* = ?$

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$

$L^0 = \{\epsilon\}$

$L^1 = \{z\}$

$L^2 = \{zz\}$

$L^3 = \{zzz\}$

$L^* = \{\epsilon, z, zz, zzz, \cdots\}$

2) If $L = \{\epsilon, 0, 1, 00, 01, 10, 11, \cdots\}$ over $\Sigma = \{0, 1\}$ then $L^* = ?$

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$

$L^0 = \{\epsilon\}$

$L^1 = \{0, 1\}$

$L^2 = \{00, 01, 10, 11\}$

$L^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

$\therefore L^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111 \cdots\}$

Eg:-3 If $\{000\}^*$ =?

$\{000\}^* = \{000\}^0 \cup \{000\}^1 \cup \{000\}^2 \cup \{000\}^3 \cup \cdots$

$= \{\epsilon\} \cup \{000\} \cup \{000000\} \cup \{000000000\} \cup \cdots$

$= \{\epsilon, 000, 000000, 000000000\}$

4) ϕ^* =?

$\{\phi\}^* = \{\phi\}^0 \cup \{\phi\}^1 \cup \{\phi\}^2 \cup \{\phi\}^3 \cup \cdots$

$= \{\epsilon\} \cup \{\} \cup \{\} \cup \{\} \cup \cdots$

$= \{\epsilon\}$

5) If $L = \{\epsilon\}$ then L^* =?

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$

$L^0 = \{\epsilon\}$

$L^1 = \{\epsilon\}^1$

$L^2 = \{\epsilon\}^2$

$L^3 = \{\epsilon\}^3$

$\therefore L^* = \{\epsilon\} \cup \{\epsilon\}^1 \cup \{\epsilon\}^2 \cup \{\epsilon\}^3$

$= \{\epsilon\}$

//

Demorgan's laws :-

Demorgan's laws used to express the intersection of languages interms of union and different languages.

i) $L_1 - (L_2 \cup L_3) = (L_1 - L_2) \cap (L_1 - L_3)$

ii) $L_1 - (L_2 \cap L_3) = (L_1 - L_2) \cup (L_1 - L_3)$

iii) $(L_1 \cup L_2)' = L_1' \cap L_2'$

iv) $(L_1 \cap L_2)' = L_1' \cup L_2'$

v) $L_1 \cap L_2 = (L_1' \cup L_2')'$

Finite Automata :-

* Introduction
* Components of FA
* elements of FA
* Representation of FA.
* Examples.

Introduction :-

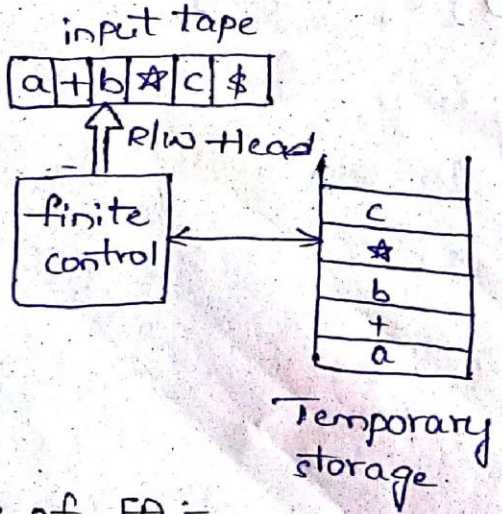
It is a self operating machine.

It is a system which obtains, transforms, transmits, and uses information to perform its functions without direct participation of humans.

Finite automata is used in mathematical analysis.

Application of finite Automata is lexical Analysis phases in compiler design.

Model of finite automata :-



components of FA :-

1) Input tape 2) R/W Head 3) Temporary storage.
4) finite control.

1) Input tape :-

It is a Memory storage used to store the input data memory area is divided into number of CELLS.

Each cell can hold only one input symbol at a time.

The input string ends with End Marker.

2) R/W Head :-

It is used by the finite control to read the input data from left side to right side in the input tape.

R/W head can move from left to right and reads only one input symbol at a time.

3) Finite control unit :-

* It controls the entire process of a system (or) machine.
* Finite control reads the input data from the input buffer tape by moving read or write head from left to right.
* It stores the reading data in temporary storage.
* It stops the reading process when the read (or) write head reaches to end marker in the input tape.

Elements of finite Automata :-

1. state 2. Transitions 3. Transition diagram / state diagram
4. Transition table.

1. state :- It is a behaviour that produce an action.

* It is a location in input buffer.
* The finite control reads the data from one state to another state in the buffer.
* states are classified into four types.
  i) initial state (or) starting state.
  ii) Final state (or) Acceptance state
  iii) Intermediate state
  iv) Invalid state (or) Dead (or) Trap state.

Initial state (or) starting state :-

The finite control can starts its reading process from a state is called initial state.

2. Finite state or acceptance state :-

Finite control can stops its reading process after complet or end of given string then it reaches a state that state is called final state.

3. Intermediate state :-

The states between starting state and final state are called Internal (or) Intermediate state.

4. Invalid state (or) Dead (or) Trap state :- It is a invalid stat when the finite control reads the input data from dead state then it always goes to dead state.

(B.) Transitions :-

* It means moving one place to another place.
* It is defined by Transition function.
* Transition function is denoted by δ.

3. Transition Diagram :-

* It is a graphical representation of finite automata.
* It is a directed graph.
* It is constructed by using the following symbols (or) Notations.

Symbol	Meaning
	state
	initial state
	final state
	dead state
	Transition

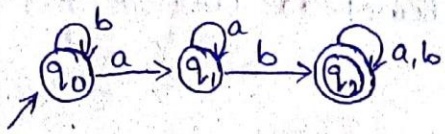
Ex :-

$\delta(q_0, a) = q_1$
$\delta(q_0, b) = q_0$
$\delta(q_1, b) = q_2$
$\delta(q_1, a) = q_1$
$\delta(q_2, a) = q_2$
$\delta_2(q_2, b) = q_2$

4. Transition tables :-

* It is a tabular representation of finite automata.
* It is combination of rows and columns. there rows represent states columns represents input symbols. The entry in between row and column is always states.

Eg :-



state	input symbols	
	a	b
→ q_0	q_1	q_0
q_1	q_1	q_2
(q_2)	q_2	q_2

** Representation of finite automata :-

Mathematically a finite automata is a 5-tuple machine like $M = (Q, \Sigma, \delta, q_0, F)$ where,

$Q \rightarrow$ set of states.

Q is a finite and Non-empty set of states.

$\Sigma \rightarrow$ It is finite and Non-empty set of input symbols.

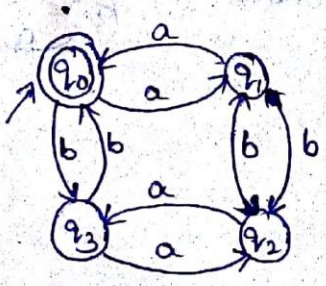
$\delta \rightarrow$ It is a transition function which is defined as $\delta: Q \times \Sigma \rightarrow Q$.

$q_0 \rightarrow$ It is a initial state or starting state and $q_0 \in Q$.

$F \rightarrow$ It is a finite set of final states and $F \subseteq Q$

Note :- * finite automata contains one initial state.
* finite automata contains one or more final states.

Ex :-



Mathematically it represented as :

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

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

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

δ is a transition function which is defined as

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

Transition table:

δ: state	input a	input b
$\rightarrow q_0$	q_1	q_3
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_2	q_0

$q_0 = \{q_0\}$

$F = \{q_0\}$

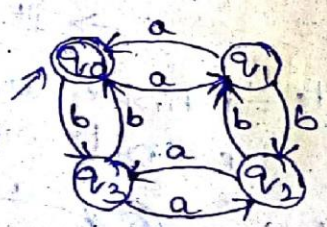
**(am) Acceptance of a string by FA:-

Let w be a given string and finite automata M contains Q states like $Q = \{q_0, q_1, q_2, \ldots q_f\}$ where q_0 is the initial state q_f is the final state. The string 'w' is accepted by machine M. If and only if $\delta(q_0, w) = q_f$.

Ex:- check whether the following strings are accepted or not by the given finite automata.

i) aabb ii) ababa iii) aabbaa iv) abababb

Sol:-



i) The given finite automata

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

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

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

δ: state	input a	input b
$\rightarrow q_0$	q_1	q_3

q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_2	q_0

$q_0 = \{q_0\}$

$F = \{q_0\}$

i) $w = aabb$

$\delta(q_0, aabb) = \delta(q_1, abb)$

$= \delta(q_0, bb)$

$= \delta(q_3, b)$

$= q_0 \in F$

∴ The given string $w = aabb$ is accepted by F.A M.

ii) $w = ababa$

$\delta(q_0, ababa) = \delta(q_1, baba)$

$= \delta(q_2, aba)$

$= \delta(q_3, ba)$

$= \delta(q_0, a)$

$= q_1$

∴ The given string $w = ababa$ is not accepted by F.A.

iii) $w = aabbaa$

$\delta(q_0, aabbaa) = \delta(q_1, abbaa)$

$= \delta(q_0, bbaa)$

$= \delta(q_3, baa)$

$= \delta(q_0, aa)$

$= \delta(q_1, a)$

$= q_0 \in F$

∴ The given string $w = aabbaa$ is accepted by F.A M.

iv) $w = abababb$

$\delta(q_0, ababbb) = \delta(q_1, babbb)$

$= \delta(q_2, abbb)$

$= \delta(q_3, bbb)$

$= \delta(q_0, bb)$

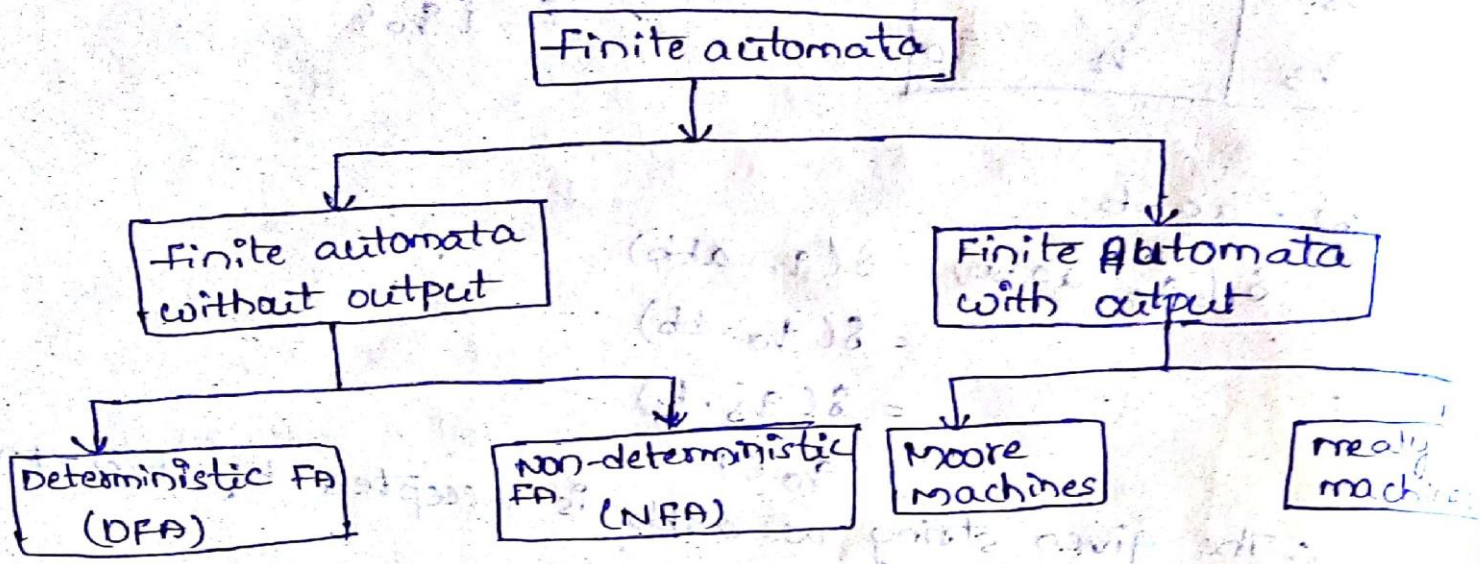
$= \delta(q_1, b)$

$= \delta(q_2, b)$

$= q_1$

∴ The given string $w = abababb$ is not accepted by F.A.

Types of FA:-



Deterministic FA:-

Introduction:- *Introduction *Representation *language Accepted by DFA

* Acceptance of DFA string by DFA

* Design of DFA.

Introduction:-

We can determine exactly what is the next state by reading a particular input symbol from a particular state then that FA is called DFA.

Definition:-

A DFA is a finite state machine where for each pair of current state and current input symbol there is a unique next state.

Representation of DFA:-

Mathematically DFA is five Tuples like

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

$Q \rightarrow$ finite and non-Empty sets of states.

$\Sigma \rightarrow$ finite and non-Empty sets of input symbols

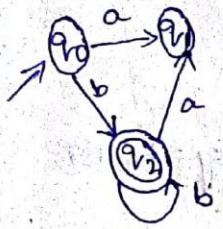
$\delta \rightarrow$ is a Transition function is defined as

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

$q_0 \rightarrow$ is the initial state.

$F \rightarrow$ is the final state.

Example of DFA:-



specification of DFA:-

1) Transition diagram 2) Transition table.

Acceptance of a string by DFA

Consider the deterministic finite Automata 'M' and the string 'w' over input Alphabet Σ. Now, the string 'w' is Accepted by 'M' if and only if $L(M) = \{w | w \in \Sigma, \delta(q_0, w) = q_f\}$

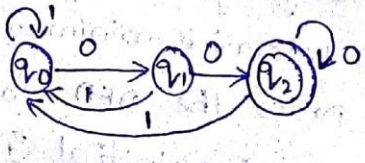
Methods for check whether the given string is accepted or not by DFA.

There are 3 methods.

1) Using sequence diagram.
2) Using extended Transition function.
3) Using $\vdash$ dash function.

eg:- 1) Consider a DFA Now check whether the following strings are accepted or not by the DFA using.

1) sequence diagram 2) Transition function 3) $\vdash$ dash function.



1) 100 2) 1101 3) 100100

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

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

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

δ:

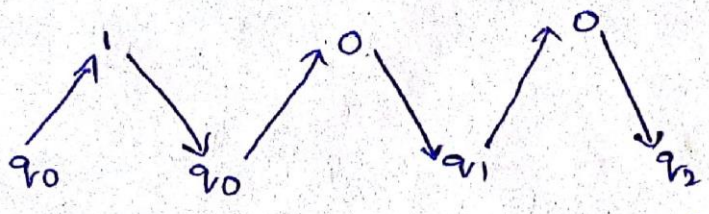
states	input	
	0	1
→ q_0	q_1	q_0
q_1	q_2	q_0
(*) q_2	q_2	q_0

$q_0 = q_0$

$F = \{q_2\}$

1. $w = 100$

1) using sequence diagram



∴ The given string $w=100$ is accepted by the DFA

2) Using Extended transition function:-

$w=100$

$\delta(q_0, w) = \delta(q_0, 100)$

$= \delta(q_0, 00)$

$= \delta(q_1, 0)$

$= q_2 \in F$

∴ The given string is accepted by FA.

3) Using ⊢ dash function:-

$w=100$

$\vdash (q_0, 100) = \vdash M(q_0, 00)$

$= \vdash M(q_1, 0)$

$= q_2 \in F$

∴ The given string is accepted by FA.

** Design of DFA:-

procedure:- 1) understanding the language which is for designing a DFA.

2) Determine minimum length string in the language.

3) Draw the DFA for minimum length string.

4) Determine initial, Intermediate, dead and final states of DFA.

5) Apply each input symbol on every state of DFA.

Eg:-

1) Design a DFA for the language which consits of set of all strings of 0's over $\Sigma = \{0\}$

let 'M' be a DFA with 5 tuples like

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

Given input $\Sigma = \{0\}$

$\therefore L(M) = \{\epsilon, 0, 00, 000 - - -\}$

Minimum string is ϵ

The next minimum string is '0'.

∴ DFA

$Q = \{q_0\}$

$\Sigma = \{0\}$

$q_0 = q_0$

$F = \{q_0\}$

δ :

state	input 0
→ q_0	q_0

2) Design a DFA for the language consist of set of all strings over accept empty string over $\Sigma = \{0\}$

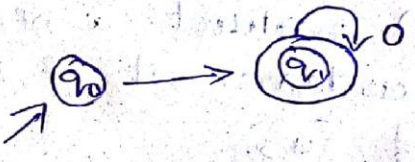
Given input $\Sigma = \{0\}$

let M be a DFA like $M = (Q, \Sigma, \delta, q_0, F)$

$\therefore L(M) = \{0, 00, 000, 0000 \ldots\}$

Minimum string is '0'.

∴ DFA



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

$\Sigma = \{0\}$

δ :

state	input 0
→ q_0	q_1
q_1	q_1

$q_0 = q_0$

$F = \{q_1\}$

3) construct a DFA that accepts a language over $\{0,1\}^*$

let 'M' be a DFA. like

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

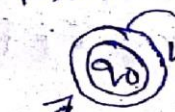
The given alphabet $\Sigma = \{0,1\}^*$

$\therefore L(M) = \{\epsilon, 0, 1, 00, 01, 10, 11\}$

Minimum string = 'ϵ'

The Next Minimum string is '0 or 1'

∴ DFA



$q_0 = q_0$

$F = \{q_0\}$

$Q = \{q_0\}$

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

δ :

state	input 0	1
→ q_0	q_0	q_0

4) construct a DFA that accepts a language over set of $\{0,1\}^+$

let M be a DFA like

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

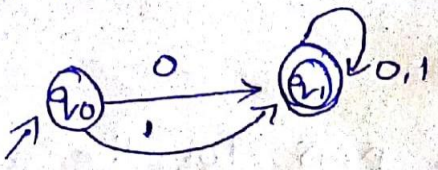
The given $\Sigma = \{0,1\}^+$

$\therefore L(M) = \{0, 1, 00, 01, 10, 11, ---\}$

The Minimum string is 0 or 1.

$\therefore$ DFA $Q = \{q_0, q_1\}$

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



δ:

state	input 0	input 1
$\rightarrow q_0$	q_1	q_1
(q_1)	q_1	q_1

$q_0 = q_0$

$F = \{q_1\}$

5) Construct a DFA that accepts a language which contains set of all strings with even number of a's over $\Sigma = \{a\}$

$\Rightarrow$ Let M be a DFA like

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

The given input $\Sigma = \{a\}$

$\therefore L(M) = \{a^0, a^2, a^4, a^6, a^8, ---\}$

$= \{\epsilon, aa, aaaa, aaaaaa, aaaaaaaa, ---\}$

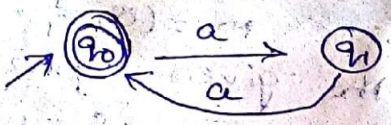
Minimum string is ϵ

Next minimum string = aa

$\therefore$ DFA

$Q = \{q_0, q_1\}$

$\Sigma = \{a\}$



δ:

state	input a
$\rightarrow (q_0)$	q_1
q_1	q_1

$q_0 = q_0$

$F = \{q_1\}$

6) Construct a DFA that accepts a language which contains set of all strings with odd number of b's over $\Sigma = \{b\}$

$\Rightarrow$ Let M be a DFA like

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

$\therefore$ The given $\Sigma = \{b\}$

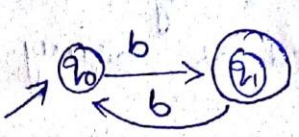
$\therefore L(M) = \{b^1, b^3, b^5, b^7, ---\}$

$= \{b, bbb, bbbbb, bbbbbbb, \ldots\}$

Minimum string is 'b'

$Q = \{q_0, q_1\}$

$\Sigma = \{b\}$



δ:

state	input b
→ q_0	q_1
(*) q_1	q_0

$q_0 = q_0$

$F = \{q_1\}$

** sum)

* Construct a DFA for a language contains the following over $\Sigma = \{0,1\}$

i) set of all strings with even no. of 0's and even no. of 1's.

ii) set of all strings with even no. of 1's and odd no. of 0's.

iii) set of all strings with odd no. of 1's and even no. of 0's.

iv) set of all strings with odd no. of 1's and odd no. of 0's.

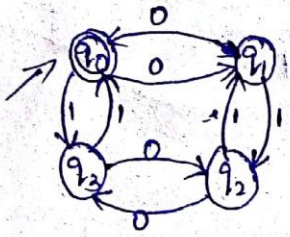
Sol:- $\Sigma = \{0,1\}$

1 → odd

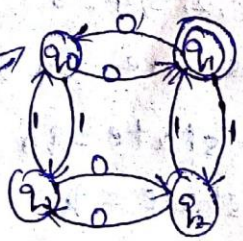
0 - even.

1	0	finalstates
0	0 = 0	q_0
0	1 = 1	q_1
1	0 = 2	q_2
1	1 = 3	q_3

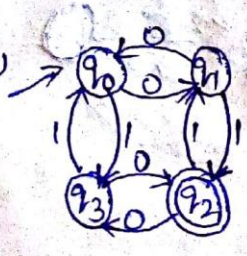
i)



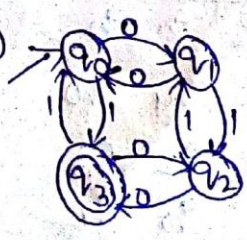
ii)



iii)



iv)



model-2

1) Design a DFA that accepts a language containing set of all strings which are divisible by 3. where string is created as binary string.

Sol:- $\Sigma = \{0, 1\}$

$L(M) = \{$ All strings divisible by 3$\}$

∴ The possible remainders are

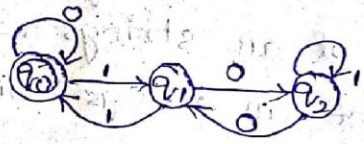
0, 1, 2

∴ The states are q_0, q_1, q_2

δ: initial state $= q_0$

final state $= q_0$

	0	1
→ q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2



2) Construct a DFA for the language $L = \{w \mid |w| \bmod 5\}$ that where string is created as ternary number.

Sol:- $\Sigma = \{0, 1, 2\}$

∴ $L(M) = \{$ All strings divisible by 5$\}$

∴ The possible remainders are

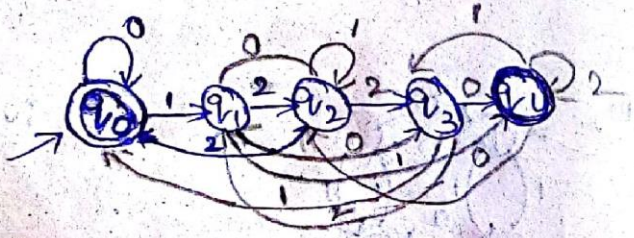
0, 1, 2, 3, 4, 等.

∴ The states are q_0, q_1, q_2, q_3, q_4

δ: initial state $= q_0$

final state $= q_0$

	0	1	2
→ q_0	q_0	q_1	q_2
q_1	q_3	q_4	q_0
q_2	q_1	q_2	q_3
q_3	q_4	q_0	q_1
q_4	q_2	q_3	q_4



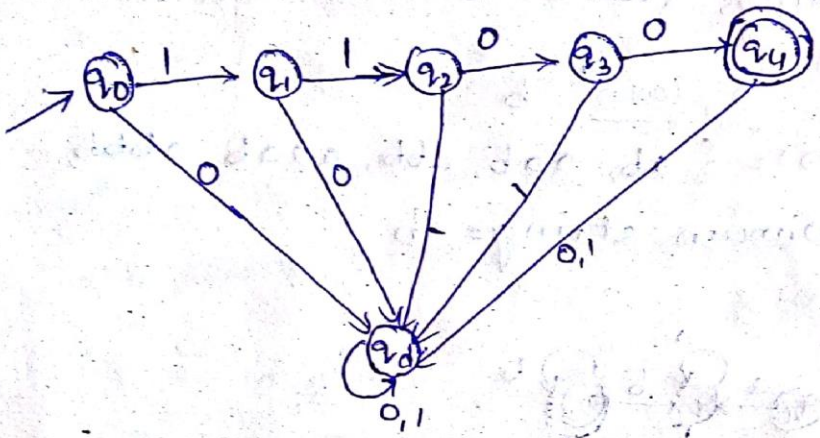
model 3

1) Construct a DFA that accepts a language which contains the string 1100 only over $\Sigma = \{0,1\}$

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

$L(M) = \{1100\}$

The minimum string = 1100



q_d is the invalid state.

2) Design a DFA that accepts set of all strings that contains 0's and 1's and Ends with 00 over $\Sigma = \{0,1\}$

Sol: Let 'M' be a DFA like

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

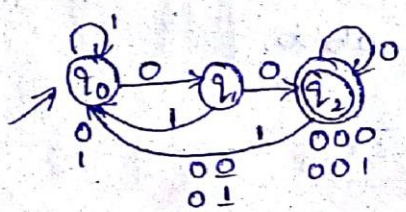
$\Sigma = \{0,1\}$

The string ends with '00'

$(0+1)^* 00$

$L(M) = \{00, 000, 100, 0000, 1100, 0100, 1000 - - -\}$

Minimum string = 00



δ:

	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_2	q_0
(q_2)	q_2	q_0

3) Design a DFA that accepts a language of set of strings which are starts with 'a' and ends with 'b' over $\Sigma = \{a, b\}$

⇒ let 'M' be a DFA like

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

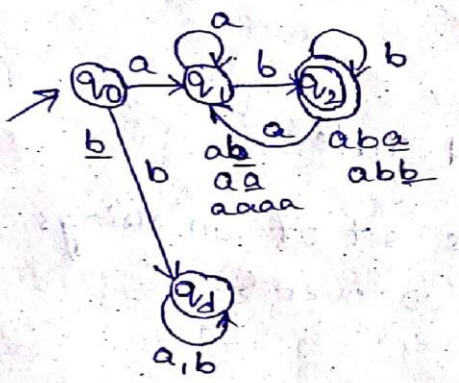
$\Sigma = \{a, b\}$

The strings starts with 'a' and ends with 'b'

$a\ \underline{(a+b)^*}\ b$

$L(M) = \{ab, aab, abb, aaab, abbb, ---\}$

minimum string = ab



δ:

	a	b
→ q_0	q_1	q_d
q_1	q_1	q_2
(*) q_2	q_1	q_2
q_d	this is a dead state	

4) Design a DFA that accepts a language of all strings which are starts with ab over $\Sigma = \{a, b\}$

Sol: Let 'M' be a DFA like

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

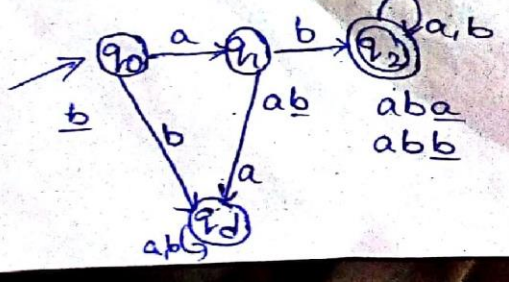
$\Sigma = \{a, b\}$

The strings starts with 'ab'

$ab\ \underline{(a+b)^*}$

$L(M) = \{ab, aba, abb, abaa, abbb, ---\}$

Minimum string = ab.



δ:

	a	b
→ q_0	q_1	q_d
q_1	q_d	q_2
(*) q_2	q_2	q_2
q_d		

Model:-4

1) Design a DFA that accepts a language of set of all strings of a's and b's. which contains 'abb' as a substring over $\Sigma = \{a, b\}$

=> Let 'M' be a DFA like

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

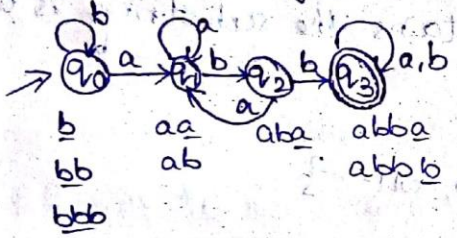
$\Sigma = \{a, b\}$

The strings with substrings as 'abb'

$(a+b)^* abb (a+b)^*$

$L(M) = \{$ abb, aabbb, babb, abba, abbb, aaabb, bbabb, abbaa, abbbbb, ---$\}$

Minimum string = abb.



δ:

	a	b	b
→ q_0	q_1	q_0	q_0
q_1	q_1	q_2	q_2
q_2	q_1	q_3	q_3
(*) q_3	q_3	q_3	q_3

2) Design a DFA over $\Sigma = \{a, b\}$ that contains set of strings of a's and b's except those containing substring bba

=> Let 'M' be a DFA like

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

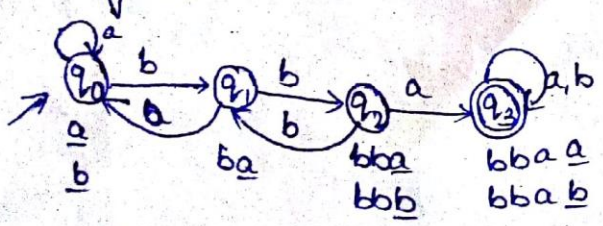
$\Sigma = \{a, b\}$

The strings with does not contains the substrings as 'bba' (each)

$(a+b)^* bba (a+b)^*$

$L(M) = \{$ bba, abba, bbba, abbaa, bbbab, ---$\}$

Minimum string = bba



δ:

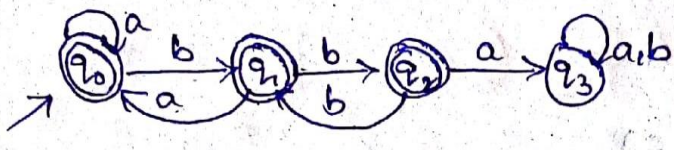
	b	b	a
→ q_0	q_1	q_1	q_0
q_1	q_2	q_2	q_0
q_2	q_1	q_1	q_3
(*) q_3	q_3	q_3	q_3

interchange

final state → non-final state

non-final state → final state

The required DFA is



3) Design a DFA over $\Sigma = \{0,1\}$ that contains set of string are 0s and 1s and those strings does not containing the substring 001

⇒ Let M be a DFA like

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

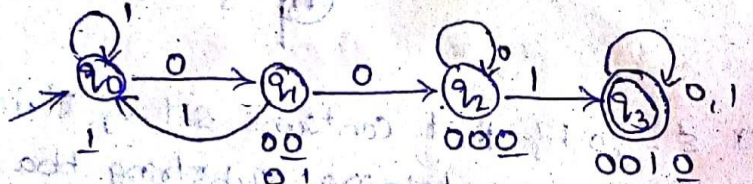
$\Sigma = \{0,1\}$

Each string does not contains the substring is 001

$(0+1)^*$ 001 $(0+1)^*$

$L(M) = \{001, 0001, 1001, 0010, 0011 - -\}$

minimum string = 001



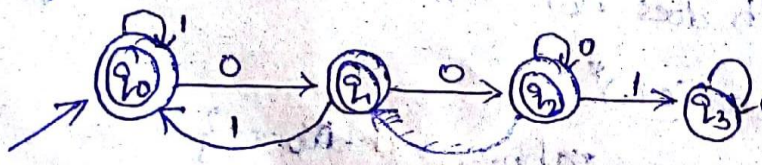
δ:	0	0	1
→ q_0	q_1	q_1	q_0
q_1	q_2	q_2	q_0
q_2	q_2	q_2	q_3
(*) q_3	q_3	q_3	q_3

interchange

final state → non final state

non final state → final state

The required DFA is



{w ∈ {a,b}* / w aa nor bb as substring}

Non-Deterministic finite Automata:-

Introduction

We cannot determine the next state exactly after reading an inputsymbol from a particular state then that FA is called NFA.

Definition:-

NFA is a finite state machine whenever each pair of current state and particular input symbol it has more than one next state.

Elements:-

1) states 2) input symbols 3) Initial state 4) final state 5) Transitions.

Representation of NFA:-

Mathematically NFA is a five Tuple like $M=(Q,\Sigma,\delta,q_0,F)$

where Q = finite and non empty set of states.

Σ = Finite and non empty set of input symbols (or) input alphabets.

δ = It is a transition function which is defined as

$$Q \times \Sigma \longrightarrow 2^Q$$

q_0 = initial state it must be belongs to Q

F = finate state and $F \in Q$.

Description of NFA:-

It is of two ways i) Transition diagram ii) Transition table

Extended transition function:-

1. $\delta(q,\epsilon)=q$ 2. $\delta(q_i,a)=q_j$ where $q_i,q_j \in Q$

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

language accepted by NFA:-

Mathematically language accepted by NFA is defined as

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

Design of NFA:—

1) procedure :- understanding the language which is for designing a NFA.
2) Determine minimum length string in the language.
3) Draw the NFA for minimum length string.
4) Determine initial, Intermediate, dead and final states of NFA.
5) Apply the each input symbol on intial and final states of NFA.

//

1) Design a NFA that accepts set of all strings over $\Sigma = \{0,1\}$ that have atleast two consecutives 0's (or) 1's.

⇒ Let 'M' be a NFA like

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

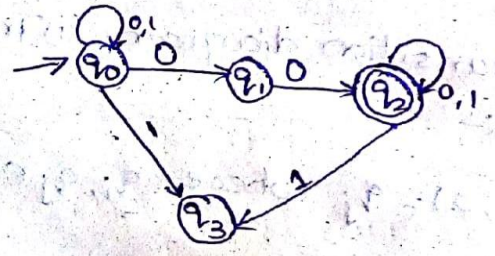
$\Sigma = \{0,1\}$

each string has atleast two consecutives 0's or 1's

$(0+1)^*(00+11)(0+0)^*$

$L(M) = \{00, 11, 000, 01, 100, 111, \ldots\}$

minimum string = 00 (or) 11



δ:

states	input 0	input 1
→ q_0	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	q_2	–
(*) q_2	q_2	q_2
q_3	–	q_2

2) Design an NFA to accept set of all strings starting with a followed by a or b and ending with a or any no. of b's over $\Sigma = \{a, b\}$

⇒ Let 'M' be NFA like

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

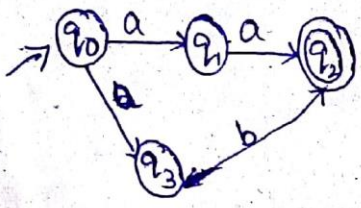
$\Sigma = \{a, b\}$

Each string starts with a and followed by 'a or b' and ending with 'a or any no. of b's'.

$a(a+b)(a+b)^*(a+b^*)$

$L(M) = \{aaa, aba, aa, ab, aaa, aaba, abaa, abab, ----\}$

Minimum string = aa or ab.



δ:

states	input a	input b
→ q_0	$\{q_1, q_3\}$	—
q_1	q_2	—
(q_2)	—	q_3 —
q_3	—	q_2

3) Design an NFA that accepts set of all strings ending in 00 over $\Sigma = \{0, 1\}$

let – M be a NFA like

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

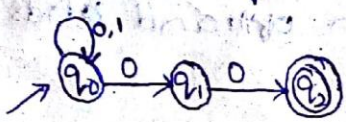
$\Sigma = \{0, 1\}$

Each string ends with 00

$(0+1)^*00$

$L(M) = \{00, 000, 100, 0000, 1100, ----\}$

Minimum string = 00



δ:

states	input 0	input 1
→ q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	—
(q_2)	—	—

4) Design an NFA that accepts set of all strings ending in aba over $\Sigma = \{a, b\}$.

=) Let – M be a NFA like

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

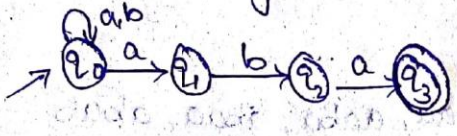
$\Sigma = \{a, b\}$

Each string ends with aba

$(a+b)^*$ aba

$L(M) = \{ aba, aaba, baba, aaaba, bbaba, --- \}$

Minimum string = aba



δ: states	input a	input b
→ q_0	$\{q_0, q_1\}$	q_0
q_1	—	q_2
q_2	q_3	—
q_3	—	—

5) Given an NFA which accepts all strings with ab over $\Sigma = \{a, b\}$:

Let M be NFA like

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

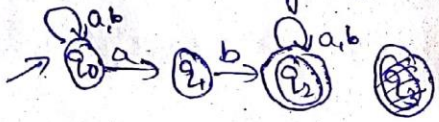
$\Sigma = \{a, b\}$

Each string have accepts all strings with ab

$(a+b)^*$ ab $(a+b)^*$

$L(M) = \{ ab, aaba, babb, --- \}$

minimum string = ab



δ: states	input a	input b
→ q_0	$\{q_0, q_1\}$	q_0
q_1	—	q_2
q_2	q_2	q_2

6) Construct an NFA that accepts all strings over $\Sigma = \{$ start with 00 and ends 01 or 10

⇒ Let M be a NFA like

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

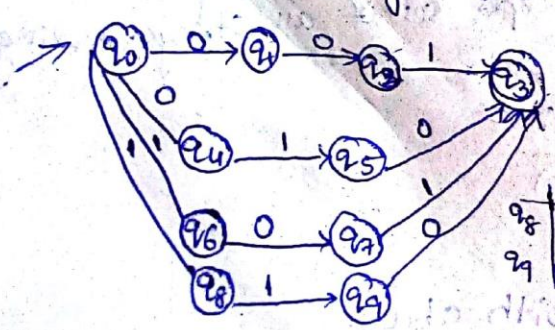
$\Sigma = \{0, 1\}$

Each string starts with 00 and ends with 01 or

$(0+1)$ $(0+1)^*$ $(01+10)$

$L(M) = \{ 001, 010, 101, 110, 0001, 0010, 0101, 0110, --- \}$

Minimum string = 001 or 010 or 101 or 110



states	Input 0	Input 1
→ q_0	$\{q_1, q_4\}$	$\{q_6, q_8\}$
q_1	q_2	—
q_2	—	q_3
q_3	—	—
q_4	—	q_5
q_5	q_3	—
q_6	q_7	—
q_7	—	q_3
q_8	—	q_9
q_9	q_3	—

7) Construct a transition system which can accept string over the alphabets a,b,c,---z containing either cat (or) RAt

⇒ Let M be a NFA like

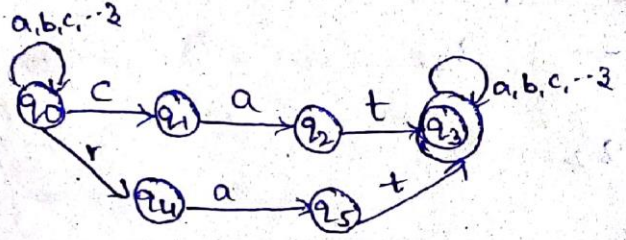
$M = (Q, \Sigma, \delta, q_0, F)$

$\Sigma = \{a,b,c,\dots z\}$

each strings containing either cat (or) RAT

$(a+b+c+\cdots+z)^* (cat + rat) (a+b+c+\cdots+z)^*$

Minimum string = cat or rat



δ:

states	input a	b	c	r	t	z
→ q_0	q_0	q_0	$\{q_0,q_1\}$	q_4	q_0	q_0
q_1	q_2	-	-	-	-	z
q_2	-	-	-	-	q_3	z
(q_3)	q_3	q_3	q_3	q_3	q_3	q_3
q_4	q_5	-	-	-	-	-
q_5	-	-	-	-	q_3	-

Conversion of NFA to DFA :-

Algorithm :-

Let D be a DFA

Let N be a NFA. This algorithm is called powerset (or) subset construction Algorithm. Because for NFA [N] with 'n' states then the corresponding DFA [D] can have 2^n states.

Subset construction Algorithm :-

step 1 :- construct the start state q_0, consisting of q_0 and all the states of NFA. that can be reached from q_0 by one or more transistions. Mark q_0 as unfinished

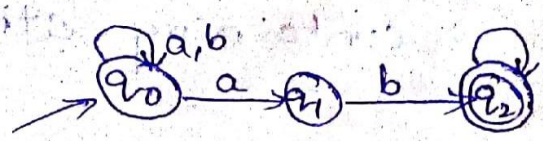
2 :- While there are unfinished states

* Take an unfinished state S.
* for each $a \in \Sigma$, $\delta(S), a = t$ is either finished (or) unfinished state.
* Mark 'S' as finished.

3 :- Mark all states that contain a final state from N. as final states of D.

Ex:- construct DFA from the Given NFA

⇒ Let



Sol:- The given NFA N is like

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

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

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

δ:

states	input a	input b
→ q_0	$\{q_0, q_1\}$	q_0
q_1	—	q_2
q_2	q_2	q_2

$q_0 = q_0$

$F = \{q_2\}$

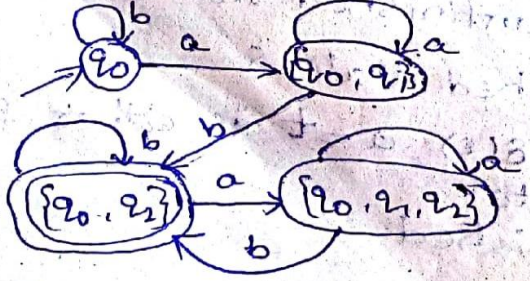
Let us consider the DFA 'D' is like

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

Apply subset construction algorithm.

states	inputs a	inputs b
→ q_0	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$

∴ The DFA is



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

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

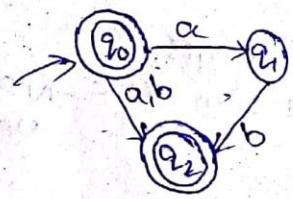
δ' =

states	Inputs a	b
→ q_0	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$

$q_0 = q_0$

$f' = \{\{q_0, q_2\}, \{q_0, q_1, q_2\}\}$.

2) construct a DFA from the given NFA



⇒ The given NFA 'N' is like

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

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

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

δ:

states	inputs a	b
→ q_0	$\{q_1, q_2\}$	q_2
q_1	ϕ	q_2
q_2	ϕ	ϕ

$q_0 = q_0$

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

Let us consider the DFA 'D' is like

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

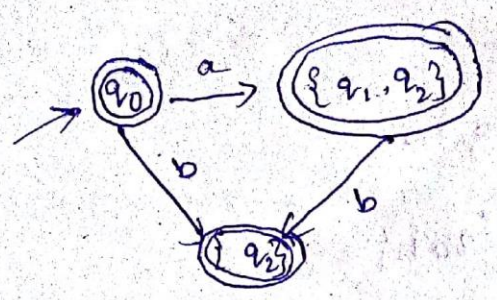
Apply subset construction algorithm

states	input a	b
→ q_0	$\{q_1, q_2\}$	q_2
$\{q_1, q_2\}$	ϕ	q_2
q_2	ϕ	ϕ

$q_0 = q_0$

$F' = \{\{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

∴ The DFA is



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

$\xi = \{a, b\}$

$q_0 = q_0$

$f' = \{ \{q_1, q_2\}, q_0, q_2 \}$

δ':

states	inputs a	b
q_0	$\{q_1, q_2\}$	q_2
$\{q_1, q_2\}$	ϕ	q_2
q_2	ϕ	ϕ

NFA with ϵ-moves:

* Introduction * Representation * Elements
* external transition function * conversion of ϵ-NFA to NFA
* ϵ-closure.

Introduction:-

* A finite state machine which contains ϵ-moves is called ϵ-NFA
* ϵ-NFA is always NFA but not DFA.

Definition:-

* ϵ-NFA is a FSM where for each pair of current state and input symbol along with epsilon have more than one next state.

Elements:-

1) states 2) inputsymbols with ϵ 3) transitions
4) initialstate 5) finalstate.

Representation:-

ϵ-NFA is a five tuple like $M = (Q, \xi, \delta, q_0, F)$,

where ~~Q is finite~~ q_0

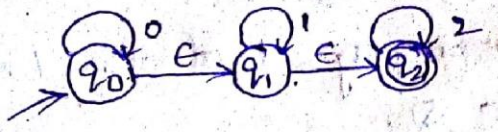
Q = is finite and non-empty set of states

ξ = finite and non-empty set of symbols.

δ = is a transition function which is defined as $\delta: Q \times \xi^* \to_2 Q$.

$Q \times \xi \cup \{\epsilon\} \to_2 Q$

Ex:-

1) Consider 

ε-closure $(q_0) = \{q_0, q_1, q_2\}$

ε-closure $(q_1) = \{q_1, q_2\}$

ε-closure $(q_2) = \{q_2\}$

***(14m) Conversion of NFA with ε-moves to NFA without ε-moves and Equivalence:-

1) Converting NFA with ε-moves to NFA without ε-moves:-

In this method we remove the ε-transitions from the given NFA and obtained NFA without ε-moves.

Algorithm:-

1) Find out all ε-transitions from each state in Q. that will be called as ε-closure of q_i, where $q_i \in Q$.

2) δ' transitions can be obtained.

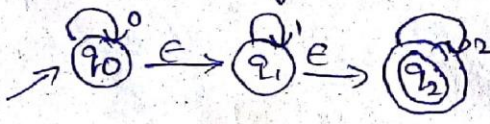
a) $\delta'(q_i, \varepsilon) = \varepsilon\text{-closure}(q)$

**b) $\delta'(q_i, a) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q), a))$

3) Repeat step 2 for each input symbol and each state given NFA.

4) Finally the resultant states of NFA without ε-moves is obtained.

Ex:- Convert the given ε-NFA to NFA.



Sol:- Let M be a given ε-NFA like.

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

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

$\Sigma = \{0, 1, 2, \varepsilon\}$

δ:- is a transition function.

states	input 0	1	2	∈
→ q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
(*) q_2	ϕ	ϕ	q_2	ϕ

$q_0 = q_0$

$F = \{q_2\}$

Now find out ∈-closure for all states in the given ∈-NFA.

∈-closure $(q_0) = \{q_0, q_1, q_2\}$

∈-closure $(q_1) = \{q_1, q_2\}$

∈-closure $(q_2) = \{q_2\}$

Compute δ'-transitions :

$\delta'(q_0, 0)$ = ∈-closure $(\delta(\text{∈-closure}(q_0), 0))$

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

= ∈-closure $(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))$

= ∈-closure $(q_0 \cup \phi \cup \phi)$

= ∈-closure (q_0)

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

$\delta'(q_0, 1)$ = ∈-closure $(\delta(\text{∈-closure}(q_0), 1))$

= ∈-closure $(\delta(\{q_0, q_1, q_2\}, 1))$

= ∈-closure $(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1))$

= ∈-closure $(\phi \cup q_1 \cup \phi)$

= ∈-closure (q_1)

= $\{q_1, q_2\}$

$\delta'(q_0, 2)$ = ∈-closure $(\delta(\text{∈-closure}(q_0), 2))$

= ∈-closure $(\delta(\{q_0, q_1, q_2\}, 2))$

= ∈-closure $(\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2))$

= ∈-closure $(\phi \cup \phi \cup q_2)$

= ∈-closure (q_2)

= $\{q_2\}$

$\delta'(q_1,0) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1),0))$

$= \epsilon\text{-closure}(\delta(\{q_1,q_2\},0))$

$= \epsilon\text{-closure}(\delta(q_1,0) \cup \delta(q_2,0))$

$= \epsilon\text{-closure}(\phi \cup \phi)$

$= \epsilon\text{-closure}(\phi)$

$= \phi$

$\delta'(q_1,1) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1),1))$

$= \epsilon\text{-closure}(\delta(\{q_1,q_2\},1))$

$= \epsilon\text{-closure}(\delta(q_1,1) \cup \delta(q_2,1))$

$= \epsilon\text{-closure}(q_1 \cup \phi)$

$= \epsilon\text{-closure}(q_1)$

$= \{q_1,q_2\}$

$\delta'(q_1,2) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2),2))$

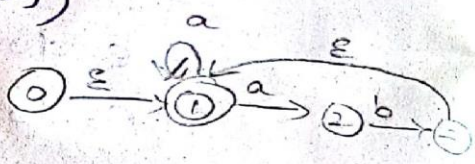
$= \epsilon\text{-closure}(\delta(\{q_1,q_2\},2))$

$= \epsilon\text{-closure}(\delta(q_1,2) \cup \delta(q_2,2))$

$= \epsilon\text{-closure}(\phi \cup q_2)$

$= \epsilon\text{-closure}(q_2)$

$= \{q_2\}$



$\delta'(q_2,0) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2),0))$

$= \epsilon\text{-closure}(\delta(\{q_2\},0))$

$= \epsilon\text{-closure}(\delta(q_2,0))$

$= \epsilon\text{-closure}(\phi)$

$= \phi$

$\delta'(q_2,1) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2),1))$

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

$= \epsilon\text{-closure}(\delta(q_2,1))$

$= \epsilon\text{-closure}(\phi)$

$= \phi$

$\delta'(q_2,2) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2),2))$

$= \epsilon\text{-closure}(\delta(\{q_2\},2))$

$= \epsilon\text{-closure}(\delta(q_2,2))$

$= \epsilon\text{-closure}(q_2)$

$= \{q_2\}$

Now the NFA without ε-moves M' is like

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

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

$\Sigma = \{0, 1, 2\}$

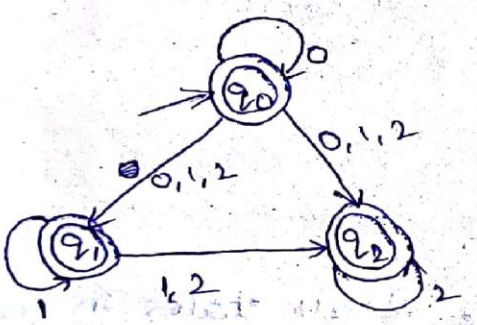
δ' :

	0	1	2
→ q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

$q_0 = q_0$

$F' = \{q_0, q_1, q_2\}$

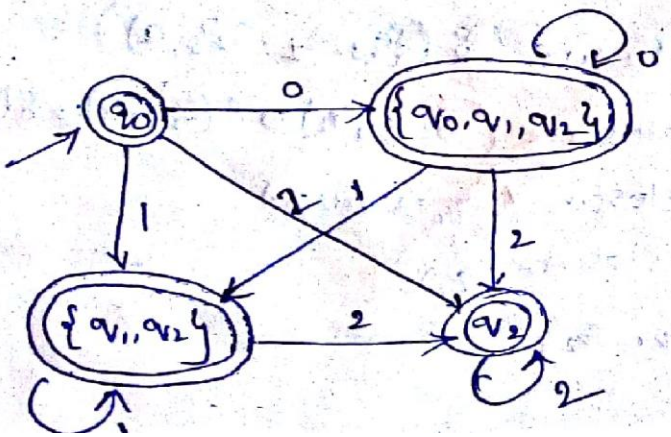
∴ NFA without ε-moves is



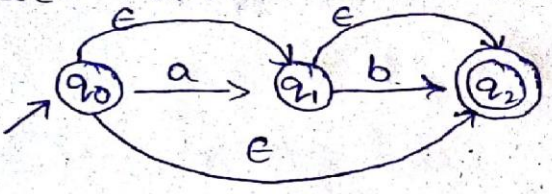
Converting NFA to DFA :-

	0	1	2
→ q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$\{q_1, q_2\}$	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	q_2

∴ The DFA is



2) Construct DFA from the given NFA with ∈ moves



→ Let M be a given NFA like

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

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

$\Sigma = \{a, b, \epsilon\}$

δ: is a transition function

δ':

state	inputs a	b	∈
→ q_0	q_1	ϕ	$\{q_1, q_2\}$
q_1	ϕ	q_2	q_2
(*) q_2	ϕ	ϕ	ϕ

$q_0 = q_0$

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

We can find out the ∈-closure of all states in the given ∈-NFA.

∈-closure of $(q_0) = \{q_0, q_1, q_2\}$

∈-closure of $(q_1) = \{q_1, q_2\}$

∈-closure of $(q_2) = \{q_2\}$

Compute δ'-transitions:-

$\delta'(q_0, a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), a))$

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

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

$= \epsilon\text{-closure}(q_1 \cup \phi \cup \phi)$

$= \epsilon\text{-closure}(q_1)$

$= \{q_1, q_2\}$

$\delta'(q_0, b) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), b))$

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

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

$= \epsilon\text{-closure}(\phi \cup q_2 \cup \phi)$

$= \epsilon\text{-closure}(q_2)$

$= \{q_2\}$

$\delta'(q_1, a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), a))$

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

$= \epsilon\text{-closure}(\delta(q_1, a) \cup \delta(q_2, a))$

$= \epsilon\text{-closure}(\phi \cup \phi)$

$= \epsilon\text{-closure}(\phi)$

$= \phi$

$\delta'(q_1, b) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), b))$

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

$= \epsilon\text{-closure}(\delta(q_1, b) \cup \delta(q_2, b))$

$= \epsilon\text{-closure}(q_2 \cup \phi)$

$= \epsilon\text{-closure}(q_2)$

$= \{q_2\}$

$\delta'(q_2, a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), a))$

$= \epsilon\text{-closure}(\delta(\{q_2\}, a))$

$= \epsilon\text{-closure}(\delta(q_2, a))$

$= \epsilon\text{-closure}(\phi)$

$= \phi$

$\delta'(q_2, b) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), b))$

$= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}\{q_2\}, b))$

$= \epsilon\text{-closure}(\delta(q_2, b))$

$= \epsilon\text{-closure}(\phi)$

$= \phi$.

Now the NFA without ε-moves M' is like.

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

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

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

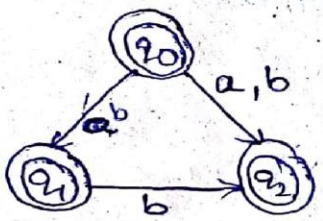
δ':

state	inputs a	b
→ q_0	$\{q_1, q_2\}$	q_2
q_1	ϕ	q_2
q_2	ϕ	ϕ

$q_0 = q_0$

$F' = \{q_0, q_1, q_2\}$

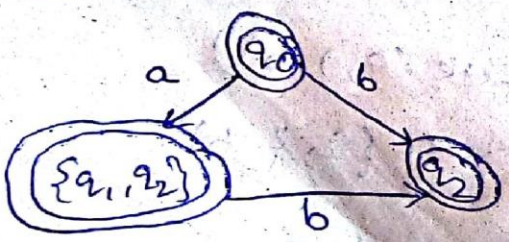
NFA without ε-moves is



Converting NFA to DFA

states	Inputs a	b
→ q_0	$\{q_1, q_2\}$	q_2
$\{q_1, q_2\}$	ϕ	q_2
q_2	ϕ	ϕ

The DFA is



Equivalence of finite state machines :-

Two finite automata's are equivalent if they accept the same set of strings over 'L'. Otherwise they are not equivalent

Algorithm :-

1) We can check the equivalence of two finite states machine by using comparison table.

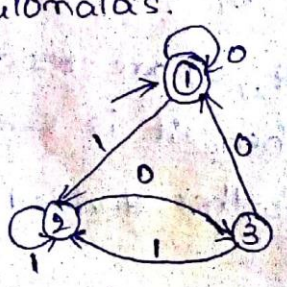
Comparison table :-

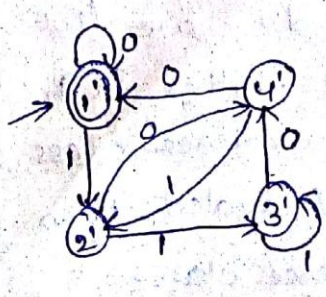
* It is similar to transition table.
* It contains (n+1) columns.
* Here, first column represents states, n column represents input symbols.
* Entries are pair of states. Here pair of states both are final states (or) both are non-final states.

Construction of Comparison table :-

* Comparison table is starts with initial states of M and M' in the first column and consider the entry as (q_0, q_0') where, $\delta(q_0, a) = q_0$ and $\delta(q_0', a) = q_0'$.
* Repeat step 1. for each input symbol on every pair of states of M and M' until no new pairs appeared.
* If you get any pair (q_0, q_0') such that q_0 is a final state and q_0' is non-final state (or) vice versa. then we terminate the process and conclude that both M and M' are not equivalent.

Ex :- check the Equivalence of the following finite automata's.





Comparison table:-

states	Input symbols. 0	1
(1,1')	(1,1')	(2,2')
(2,2')	(3,4')	(2,3')
(3,4')	(1,1')	(2,2')
(2,3')	(3,4')	(2,3')

∴ The given 2 Finite automatas are Equivalence.

Minimization (or) optimization of FA :-

The process of reducing the no.of states from given FA is called minimization (or) optimization of FA.

Equivalence states:-

The two states q_1 and q_2 are Equivalent if both $\delta(q_1,a)$ and $\delta(q_2,a)$ are Final states (or) non-final states. While minimizing finite automata we first find out which two states are Equivalent then we can represent these two states by one representative state.

Minimization Algorithm:-

* we will create a set $\Pi_0 = \{\{Q_1^0\}\ \{Q_2^0\}\}$ where $\{Q_1^0\}$ is the set of final states and $\{Q_2^0\}$ is the set of non-final states

Π_0 is 0' Equivalence class.

* Now we will construct Π_{K+1} from Π_K.

Let Q_i^K be any subset in Π_K.

If q_1 and q_2 are two states in Q_i^K.

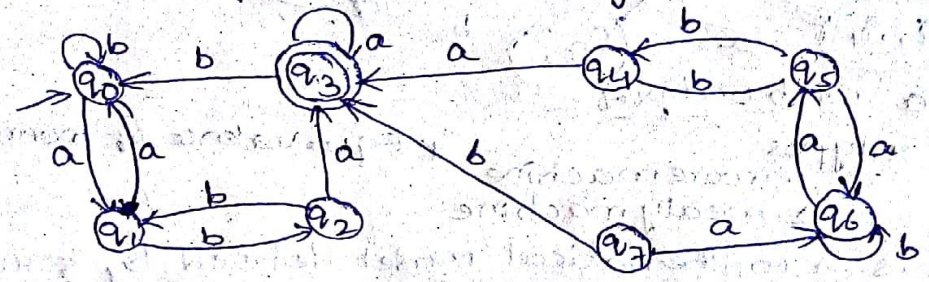
Find out whether $\delta(q_1,a)$ and $\delta(q_2,a)$ are residing in the same Equivalence class in Π_K. Then it is said that q_1, q_2 are K+1 Equivalent then Q_i^K is further divided into 'K+1' Equivalence classes.

* Repeat step 2. for every Q_i^K in Π_K and obtain all elements in Π_{K+1}.

* continue the above process until $\pi_n = \pi_{n+1}$ where $n = \{1, 2, 3, \ldots\}$
* Then replace all the equivalence states in one equivalence class find representing states.

This helps minimizing the given finite automata.

Ex:- Construct the state minimizing the Finite automata for the following transition diagram.



Sol:- The given FA, M is like

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

where

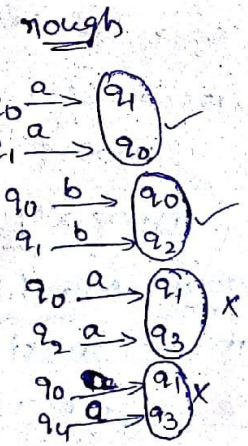
$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$

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

δ: is a transition function.

states	Input symbols a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
(final) q_3	q_3	q_0
q_4	q_3	q_5
q_5	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

rough



$\pi_0 = \{\{q_3\}, \{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}\}$

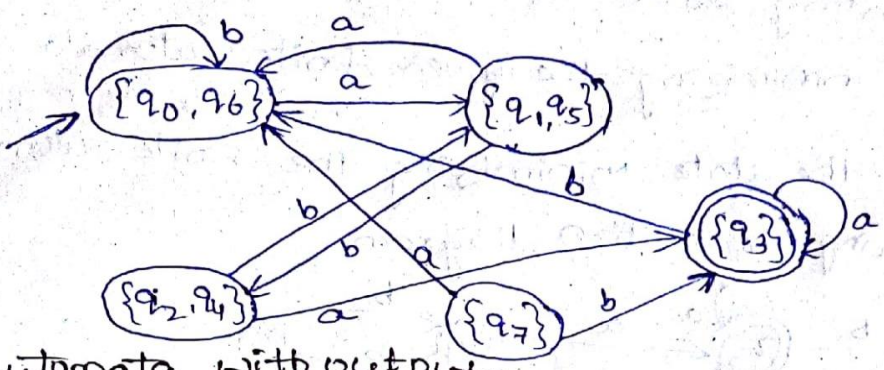
$\pi_1 = \{\{q_3\}, \{q_0, q_1, q_5, q_6\}, \{q_2, q_4, q_7\}\}$

$\pi_2 = \{\{q_3\}, \{q_0, q_6\}\ \{q_1, q_5\}, \{q_2, q_4\}\ \{q_7\}\}$

$\pi_3 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}\ \{q_7\}\}$

$\therefore \Pi_3 = \Pi_2$

∴ The minimized FA



Finite automata with output :-

* Introduction * types
  1. moore machine
  2. mealy machine
* Equivalence of moore and mealy machine.

* Introduction :-

We know FA is a mathematical model defined by 5 Tuples like $M = (Q, \Sigma, \delta, q_0, F)$.

Without information about the output.

After reading a string if finite automata reaches to the final state then the string is accepted by FA. Else the string is registered.

FA without output is called language acceptors.

Transducers :-

* the FA with output is called Transducers.
* It doesnot contain final states.
* It cannot be used for language Acceptor.
* It provides the information about output.
* It can be used for type conversion of language
* Transducers are two types.
  i) moore machine
  ii) mealy machine.

i) moore Machine :-

moore machine is a FA that contains set of states in which "the output is always depends on present state only".

present state → output.

mathematically moore machine is 6 tuples like.

$M = (Q, \Sigma, \delta, \Delta, \lambda, q_0)$

→ where Q = finite and non empty set of states.

Σ = finite and non-empty set of Input symbol.

δ = It is a transition function defined as

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

Δ = It is a finite and non-empty set of output symbols(or) output Alphabets.

λ = It is a output function which is defined as

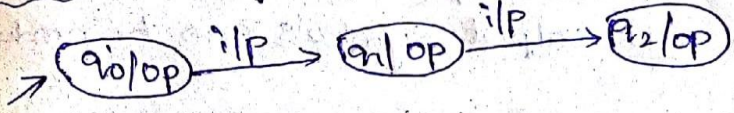
$$\lambda: Q \to \Delta$$

q_0 = It is a initial state and must be $q_0 \in Q$.

Representation of moore machine:-

i) Transition diagram ii) transition table

Transition diagram:-



Transition table:-

present states	Input symbols a	b	c	d	output

ex:- Design a moore machine for residue 13| for the input string is created as a binary number.

sol:- Given $\Sigma = \{0,1\}$

residue 13| means if any number is divisible by 3 then we get the possible remainders are 0,1,2.

$\therefore \Delta = \{0,1,2\}$

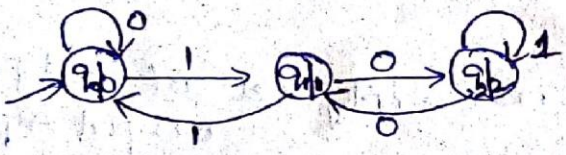
The possible states are $\{q_0, q_1, q_2\}$

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

$\therefore \delta$: Transition function is defined as

δ:	o/p | i/p 0	1
→ q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

∴ The moore machine is.



Transition table :-

present state	I/p symbols 0	I/p symbols 1	output
→ q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

Note :- In moore machine (n+1) output symbols can be produced as 'n' input symbols.

Mealy machine:-

It is a finite automata contains set of states in which "the output is always depends on present state and input symbol".

Mathematically mealy machine defined by 6 tuples like $M = (Q, \Sigma, \delta, \Delta, \lambda, q_0)$

where,

$Q \rightarrow$ finite and non-empty set of states.

$\Sigma \rightarrow$ finite and non-empty set of states and input symbols.

δ is a transition function is defined as

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

Δ is a finite and non empty set of states and output symbols.

λ is a output function is defined as,

q_0 is the Initial state must be $q_0 \in Q$.

Representation :- In two ways

1. Transition diagram 2. Transition Table.

① Transition diagram

2. Transition Table

present state	Input symbols,		Input symbols 2	
	Next state	output	Next state	output

Design a Mealy machine for residue mode 3 in which the input is treated as a binary machine.

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

Residue mode 3 means if any number is divisible by 3. then the possible remainders are 0, 1, 2

$$\Delta = \{0,1,2\}$$

The possible states are $\{q_0, q_1, q_2\}$

$\therefore \delta$ is transition function

δ: state	Input symbols 0	1
→ q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

The Mealy machine is

Transition table

present state	input symbol, (0) Next state	output	Input symbol, (1) Next state	output
→ q_0	q_0	0	q_1	1
q_1	q_2	2	q_0	0
q_2	q_1	1	q_2	2

Note:- In Mealy Machine 'n' output symbols can be produced for 'n' input symbols.

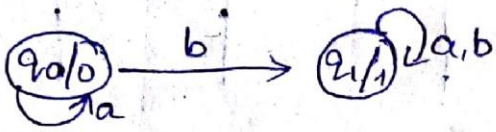
Equivalence of Mealy and Moore Machine:-

Conversion of Moore to Mealy machine

Conversion of Mealy to Moore machine

Conversion of Moore and Mealy machine:-

Convert the following Moore machine to Mealy machine



The given Moore machine is 'M' like

$M = (Q, \Sigma, \delta, \Delta, \lambda, q_0)$

$Q = \{q_0, q_1\}$

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

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

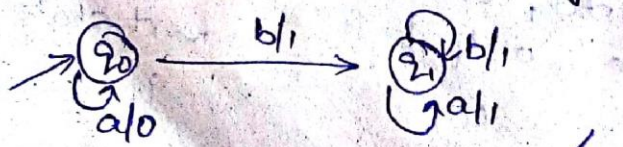
λ : output function.

present state	Input symbols a	b	output
→ q_0	q_0	q_1	0
q_1	q_1	q_1	1

Now the transition table for Mealy machine is

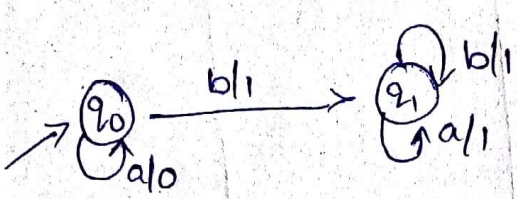
present state	Input symbol$_1$ (a)		Input symbol$_2$ (b)	
	Next state	output	Next state	output
→ q_0	q_0	0	q_1	1
q_1	q_1	1	q_1	1

Transition diagram for Mealy machine is



Conversion of Mealy to Moore Machine:-

Convert the following mealy machine to moore machine.



The given mealy machine is M like

$$M = (Q, \Sigma, \delta, \Delta, \lambda, q_0)$$

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

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

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

δ:

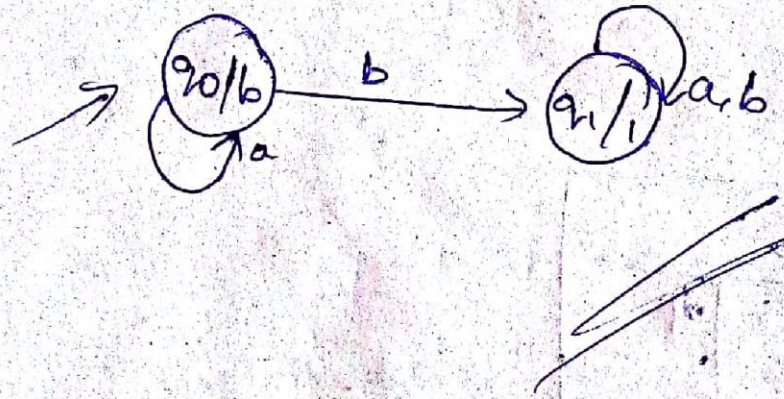
present state	a		b	
	N.S	o/p	N.S	o/p
→ q_0	q_0	0	q_1	1
q_1	q_1	1	q_1	1

δ:	a	b
q_0	q_0	q_1
q_1	q_1	q_1

Now moore machine table:

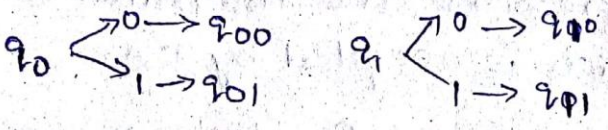
Present state	Input symbol		output
	a	b	
→ q_0	q_0	q_1	0
q_1	q_1	q_1	1

Transition diagram for moore machine.



convert the following mealy machine to moore machine

P.S	a		b	
	N.S	o/p	N.S	o/p
→q_0	q_0	0	q_1	1
q_1	q_0	1	q_1	0



Transition table for moore machine

P.S	a	b	output
→q_{00}	q_{00}	q_{11}	0
→q_{01}	q_{00}	q_{11}	1
q_{10}	q_{01}	q_{10}	0
q_{11}	q_{01}	q_{10}	1

Applications and limitations of FA:-

limitations:-

* FA contains and i/p buffer with limited (or) finite no.of locations. i.e; it has a limited amount of memory for storing i/p data.
* It recognise a finite length of i/p string.
* It can moves its read/write head in either left to right (or) right to left direction only.

Applications:-

* FA is used to design lexical analyser.
* FA is used to create text editors.
* FA is used to spell checking.
* FA is used to design sequential circuits.

INDEX

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 $\sum^0$ = {ε} – 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 $\sum^0$ = {ε} – strings of length 0,
$\sum^1$={0,1}– strings of length 1,
$\sum^2$={00,01,10,11}– strings of length 2,…

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

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 $\sum^0$ = {ε} – strings of length 0,
$\sum^1$={0,1}– strings of length 1,
$\sum^2$={00,01,10,11}– strings of length 2,…

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

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

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

Operations on languages:

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 $L_1 \cup L_2$.

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

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 $L_1 \cap L_2$.

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

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 $L_1 - L_2$.

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

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. **$L_1.L_2$={w| w1 in L_1 and w2 in L_2}**

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_1= {ε, 01,0011,000111,. .}.

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

$L_1.L_2$={ **ε.** ε , **ε.** 01, **ε.** 10, **ε.** 0011, **ε.** 0101, **ε.** 1001,…,**01**10,**01**0011,**01**0101,…}
= {ε, 01, 10, 0011, 0101, 1001, .. ., 0101,010011,010101…}

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≥1 $\mathbf{L^* = L^0}$
$\cup \mathbf{L^1} \cup \mathbf{L^2} \cup \mathbf{...}$

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

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

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^+.

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

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

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

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

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:

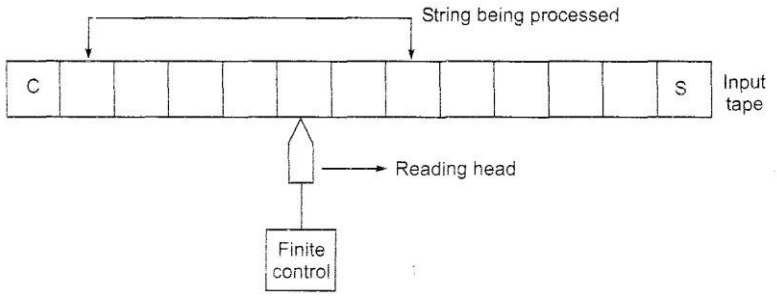
"Finite State Machine" is a model of computation which is used to simulate the behaviour of a machine or a system.

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:

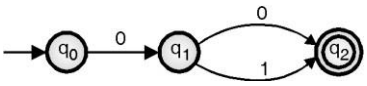
This element defines the change in state. i,e., moving from one state to another state is known as transition. Transitions are represented in 3 ways. They are 1. Transition diagram

2. Transition table

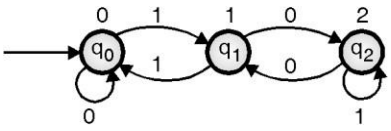
3. Transition function.

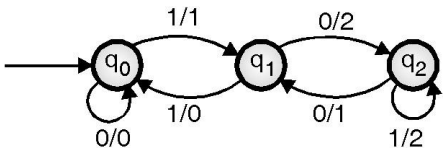
Types of transition:

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.

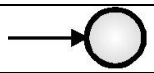
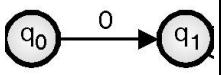


For a finite state machine, transition diagram or transition graph or a transition system is formally defined as a directed labeled graph where each vertex is a state, and directed edge is a transition between two states and edges are labeled with input/output.
Note: For DFA, NFA, and Moore machine,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

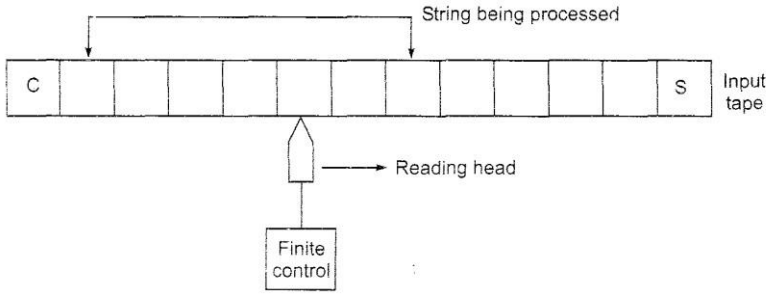
x is used to represent current input symbol and $\delta(q_i, x)$ represents the next state based on q_i and x and

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

$$M = Finite\ State\ Machine$$
$$Q - finite\ number\ of\ states$$

$\Sigma - input\ alphabet\ or\ set\ of\ input\ symbols$

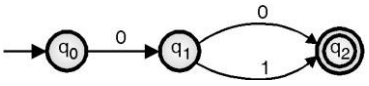
$\delta - transition\ function\ defined\ by\ \delta: QX\Sigma \rightarrow Q$

$q_0 - start\ state\ q_0 \in Q$

$F - set\ of\ final\ states\ F \subseteq Q$

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

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

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 $q_f \epsilon F$. 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	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

Solution

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

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 ↓ 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-)
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 $\boldsymbol{M = (Q, \Sigma, \delta, q_0, F)}$ where

$M = Finite\ State\ Machine$

$Q - finite\ number\ of\ states$

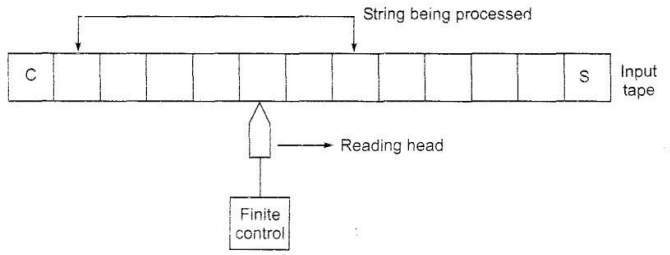
$\Sigma - input\ alphabet\ or\ set\ of\ input\ symbols$

$\delta - transition\ function\ defined\ by\ \delta: QX\Sigma \rightarrow Q$

$q_0 - start\ state\ q_0 \in Q$

$F - set\ of\ final\ states\ F \subseteq Q$

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 $\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{F})$ where

$M = Finite\ State\ Machine$

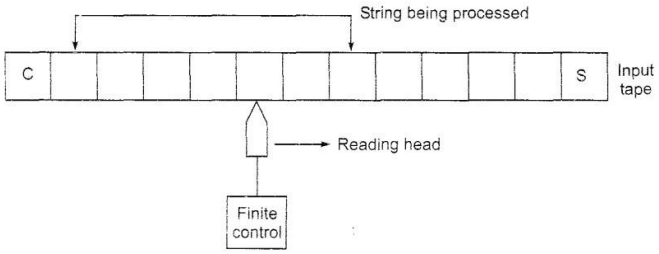
$Q - finite\ number\ of\ states$

$\Sigma - input\ alphabet\ or\ set\ of\ input\ symbols$

$\delta - transition\ function\ defined\ by\ \delta: QX\Sigma \rightarrow 2^Q\ ,\ 2^Q \subseteq Q$

$q_0 - start\ state\ q_0 \in Q$ $F - set\ of\ final\ states\ F \subseteq Q$

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)

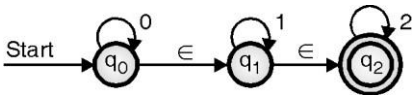
Definition:

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 $\boldsymbol{M = (Q, \Sigma, \delta, q_0, F)}$ where

$$M = Finite\ State\ Machine$$
$$Q - finite\ number\ of\ states$$

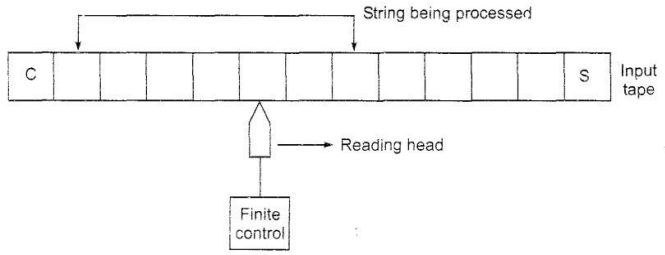
$\Sigma - input\ alphabet\ or\ set\ of\ input\ symbols$

$\delta - transition\ function\ defined\ by\ \delta$: $QX\Sigma \cup \{\varepsilon\} \rightarrow 2^Q$, $2^Q \subseteq Q$

$q_0 - start\ state\ q_0 \in Q$

$F - set\ of\ final\ states\ F \subseteq Q$

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" ($k \geq 0$) , 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 $\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\Delta}, \boldsymbol{\delta}, \boldsymbol{\lambda}, \boldsymbol{q_0})$ where

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

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 $\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\Delta}, \boldsymbol{\delta}, \boldsymbol{\lambda}, \boldsymbol{q_0})$ where

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

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 $\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{F})$ where $M =$ $Finite\ State\ Machin$ Σ $Q - finite\ number\ of\ states$ $- input\ alphabet$ $\delta \quad - transition\ function\ defined$ $\delta: QX \quad Q$ $q_0 - start\ state\ q_0 \in$ $set \quad of\ final\ states\ F \quad Q$ by $\Sigma \rightarrow$ Q $\subseteq$ $F -$	NFA is formally defined as 5-tuple $\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{F})$ where $M = Finite\ State\ Machin$ $Q - finite\ number\ of\ states$ Σ $- input\ alphabet$ $\delta \quad - transition \quad functiondefined$ $\delta: QX \quad \cup \{ \quad ,$ $q \quad - start\ state \quad q_0$ by $set \quad of\ final\ states \rightarrow \quad \} \quad Q$ $Q \quad \in \quad \Sigma \quad F -$ $\varepsilon \quad 2^Q \quad \subseteq Q \quad F \quad 2^Q \subseteq$ 0
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 $M = Moore\ Machine$ $finite\ number\ of\ states$ $Q -$ Σ $- output\ alphabet$ δ $input\ transition\ function\ defined$ $\_defined \rightarrow$ Q $function by$ $\delta: QX$ $transition by\ \delta: Q \rightarrow$ λ $- output$ $q_0 \in Q$ $state$ q $- input\ alphabet$ $_0 - start$ Δ Σ Δ	Mealy machine is formally defined as 6-tuple $\boldsymbol{M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)}$ where $M = Moore\ Machine$ $finite\ number\ of\ states$ $Q -$ $- input\ alphabet$ Σ $- output\ alphabet$ Δ δ $- input$ $transition$ $function$ $by\ \delta: QX \rightarrow$ Q $defined$ $\lambda - output$ $transition$ by $defined$ $q_0 - start$ $state$ $function \delta: QX$ $\rightarrow$ $q_0 \in$ Σ Q Σ Δ
For input ε, output is λ(□$_0$)	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’.