Chapter 2. Finite State Automata [10 hrs.][16 Marks]

2.  *Finite State Automata*

   2.1. *Sequential Circuits and Finite State Machine*
   2.2. *Finite state automata*
   2.3. *Language and grammars*
   2.4. *Non-deterministic finite state automata*
   2.5. *Language and automata*
   2.6. *Regular expression and its characteristics*

**Combinational circuit.**

Combinational circuits have no memory or in term states, their output depends only on the current values of their inputs.

**Sequential circuits**

It contains memory element or internal states.

The output depends not only on its current inputs but also on the past history of those inputs.

Finite state machines have internal states, so their output may depend not only on its current inputs but also on the past state or past history of those inputs.

A unit time delay accepts as input a bit \( x_t \) at time \( t \) and outputs \( x_{t-1} \), the bit received at time \( t-1 \).

\[
\text{delay} \quad X_t \quad X_{t-1}
\]

A serial adder accepts as input two binary numbers

\[ x = 0x_nx_{n-1} \ldots x_0 \text{ and } y = 0y_ny_{n-1} \ldots y_0 \text{ and} \]

Outputs the sum \( z_{n+1}z_n \ldots z_0 \) of \( x \) and \( y \).

The numbers \( x \) and \( y \) are input sequentially in pairs \( x_0, y_0, \ldots, x_n, y_n \).

The sum output is \( z_0, z_1, \ldots, z_{n+1} \).

\[ X = 010 \text{ and } y = 011 \]

Let \( i = 0 \) before start (this can be done by first setting \( x = y = 0 \) )
Finite State Machine

A finite state machine is an abstract model of a machine with a primitive internal memory.

Definition

A finite state machine \( M \) consists of

(a) A finite set \( I \) of input symbols.
(b) A finite set \( O \) of output symbols.
(c) A finite set \( S \) of states.
(d) A next state function \( f \) from \( S \times I \) into \( S \).
(e) An output function \( g \) from \( S \times I \) into \( O \).
(f) An initial state

We write

\[ M = (I, O, S, f, g, \sigma) \]

Example

Let \( I = \{a, b\} \), \( O = \{0, 1\} \) and \( S = \{ \sigma_0, \sigma_1 \} \)

Define the pair of functions

\( f : S \times I \rightarrow S \) and \( g : S \times I \rightarrow O \) by the rules given below.

<table>
<thead>
<tr>
<th>( S/I )</th>
<th>( f )</th>
<th>( g )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_0 )</td>
<td>( a )</td>
<td>( a )</td>
</tr>
<tr>
<td>( \sigma_0 )</td>
<td>( b )</td>
<td>( b )</td>
</tr>
<tr>
<td>( \sigma_1 )</td>
<td>( 0 )</td>
<td>( 1 )</td>
</tr>
<tr>
<td>( \sigma_1 )</td>
<td>( 1 )</td>
<td>( 0 )</td>
</tr>
</tbody>
</table>
Then M=(I, O, S, f, g, σ₀) is a FSM

\[ f(σ₀, a) = σ₀ \quad g(σ₀, a) = 0 \]
\[ f(σ₀, b) = σ₁ \quad g(σ₀, b) = 1 \]
\[ f(σ₁, a) = σ₁ \quad g(σ₁, a) = 1 \]
\[ f(σ₁, b) = σ₂ \quad g(σ₁, b) = 0 \]

**Transition diagram**

The transition diagram is a digraph. The vertices are the states. The initial state is indicated by an arrow. If we are in state \( σ \) and inputting I causes output O and moves us to state \( σ' \), we draw a directed edge from vertex \( σ \) to vertex \( σ' \) and label it i/o.

Let M={I,O,S, f, g, σ } be a FSM. The transition diagram of M is a digraph G whose vertices are the members S, an arrow designates the initial state. A directed edge \((σ₁, σ₂)\) exist in G. If there exists an input I with \( f(σ₁, i) = σ₂ \), In this case, if \( g(σ₁, i) = 0 \). The edge \((σ₁, σ₂)\) is labeled i/o.

**Definition**

Let M=(I, O, S, f, g, σ) be a FSM. An I/P string for M is a string over I.

The string

\[ y₁ \ldots yₙ \]

is the o/p string for M corresponding to the i/p string

\[ x₁ \ldots xₙ \]

If there exists states \( σ₀, \ldots σₙ \in S \)

With \( σ₀ = σ \)

\( σᵢ = f(σᵢ₋₁, xᵢ) \) for \( i=1, \ldots n \)

\( yᵢ = g(σᵢ₋₁, xᵢ) \) for \( i=1, \ldots n \)
Example

1) Design a FSM that performs serial addition

serial adder accepts pair of bits

The i/p set will be {00,01,10,11}
The o/p set is {0,1}

Given an i/p x, y
i) either add x and y or
ii) add x, y and 1 depending on whether the carry bit was 0 or 1. So, we have two states c (carry) and NC (no carry). The initial state is NC.

Transition diagram:

The SR flip flop

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Not allowed</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
| 0 | 0 | 1 if s was last equal to 1
|   |   | 0 if R was last equal to 1|

The SR flip flop
**Finite State Automata**

Finite state automate is a special kind of FSM. It has no output and no output function but has accepting or final state.

Definition:

A FSA A={I, O, s, f, g, σ} is a FSM in which the set of o/p symbols is \{0,1\} and where the current state determines the last output. Those states for which the last output was 1 are called accepting states.

E.g. The initial state is $\sigma_0$

<table>
<thead>
<tr>
<th></th>
<th>$f$</th>
<th>$g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s/I$</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>$\sigma_0$</td>
<td>$\sigma_1$</td>
<td>$\sigma_0$</td>
</tr>
<tr>
<td>$\sigma_1$</td>
<td>$\sigma_2$</td>
<td>$\sigma_0$</td>
</tr>
<tr>
<td>$\sigma_2$</td>
<td>$\sigma_2$</td>
<td>$\sigma_0$</td>
</tr>
</tbody>
</table>

If we are in state $\sigma_0$, the last output was 0. If we are in either state $\sigma_1$ or $\sigma_2$, the last o/p was 1. Thus A is FSA. The accepting states are $\sigma_1$ and $\sigma_2$. 
Q) Draw the transition diagram of the FSA as a transition diagram of a FSM

\[ Q \]

\[ FSA \]

\[ FSM \]

\( \sigma_2 \) is an accepting state
Its incoming edges with a and b.
The other states \( \sigma_0 \) and \( \sigma_1 \) are not accepting state so that they have incoming along with only a.

**Finite State Automata**

A finite state automata is similar to finite state machine but with no output, and with a set of states called accepting or final states. FSA can be regarded as.

1) A finite set \( I \) of input symbols.
2) A finite set \( S \) of input symbols.
3) A next state function \( f \) from \( SXI \) into \( S \).
4) A subset \( A \) of \( S \) of accepting states
5) An initial state \( \sigma \in S \)

\( A= (I, S, f, A, \sigma) \)

**Example**

Transition diagram of FSA

\( A= (I, S, f, A, \sigma) \) where

\( I=\{a, b\}, S=\{ \sigma_0, \sigma_1, \sigma_3 \}, A=\{ \sigma_2 \} \)

\( \sigma=\sigma_0 \) and \( f \) is given by table below

<table>
<thead>
<tr>
<th>s/I</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_0 )</td>
<td>( \sigma_0 )</td>
<td>( \sigma_1 )</td>
</tr>
<tr>
<td>( \sigma_1 )</td>
<td>( \sigma_0 )</td>
<td>( \sigma_2 )</td>
</tr>
<tr>
<td>( \sigma_2 )</td>
<td>( \sigma_0 )</td>
<td>( \sigma_2 )</td>
</tr>
</tbody>
</table>
Incoming is i/p to a FSA, we will end at either an accepting or a non accepting state. The states of this final state determines whether the string is accepted by FSA.

Definition
Let \( A(I, S, f, A, \sigma) \) be a FSA. Let \( \alpha = x_1, \ldots, x_n \) be a string over \( I \). If there exists states \( \sigma_0, \ldots, \sigma_n \) satisfying,

\[
\begin{align*}
& a) \quad \sigma_0 = \sigma \\
& b) \quad f(\sigma_{i-1}, x_i) = \sigma_i \text{ for } i=1, \ldots, n \\
& c) \quad \sigma_n \in A
\end{align*}
\]

we say that \( \alpha \) is accepted by \( A \). The null string is accepted if \( f \sigma \in A \).

Let \( \text{Ac}(A) \) denoted the set of string accepted by \( A \) and we say that \( A \) accepted \( \text{Ac}(A) \).

Is the string \( abaa \) accepted by FSA

Yes

Is the string \( abbabba \) accepted by FSA

No
Q) Design FSA that accepts precisely those strings over \{a,b\} that contain an odd number of a's.

Two states
Odd A = (OA)
Even A = (EA)

OA is accepted state
EA is the initial state

If two FSA accept precisely the same things, we say that the automata are equivalent. FSA A and B are equivalent if \( \text{Ac}(A) = \text{Ac}(B) \)

**Algorithm**

This algorithm determines whether a string over \{a, b\} is accepted by FSA.

Input : \( n \), the length of the string (\( n=0 \) designation the null string); \( s_1 s_2 \ldots s_n \), the string.

Output : ‘Accept’ if the string is accepted  
‘Reject’ if the string is not accepted

procedure fsa (s, n)

State : ’E’

For \( i=1 \) to \( n \) do

begin

If state = ‘E’ and \( S_i = \text{a} \) then

State = ‘0’

If state = ’0’ and \( S_i = \text{a} \) then


state = 'E'
End
If state = '0' then
Return ("Accept")
Else
Return ("Reject")
End fsa

**Language and Grammar**

**Alphabet or vocabulary**

Alphabet or vocabulary is a finite non empty set of systems.
It is denoted by $\sum$ or A or v

$V = \{0,1\}$ is a binary alphabet

$V = \{a, b, \ldots, z\}$ is a set of lower case letter.

**String(w)**

A finite sequence of symbols taken from some alphabet is called string.

Word or sentence is string of finite length of element of symbols.

0100 is string from alphabet $v = \{0,1\}$

Concatenation (vw): vw is string v, then string w.

$W^n$: string w repeated n times

$V^*$: all possible string using v.
Empty string (\( \lambda \) or \( \in \) or e)

Zero occurrence of symbols is null or empty string.

Length of string

The number of symbols present in the string.

\[ W = 0100 \]

\[ |w| = 4 \]

\[ |\in| = 0 \]

Language

Language is a body of words and method of combining words used to understand by a considerable community. Such languages are often called natural to distinguish them from formal languages, which are used to model natural languages and to communicate with computers.

The rules of a natural language are very complex and difficult to characterize completely.

It is possible to specify completely the rules by which certain formal languages are constructed.

Definition

The set of all words or string including zero or \( v \) is denoted by \( V^* \)

Language is set of strings all of which are taken from \( V^* \)

A formal language over \( v \) is subset of \( V^* \)

\( L \subseteq V^* \)

\( V = \{0,1\} \)

\( V^0 = \{\} \)

\( V^1 = \{0,1\} \)

\( V^2 = VUV \)

\( = \{0,1\} U \{0,1\} \)

\( = \{00,01,10,11\} \)
\[ V^* = \bigcup_{i=0}^{\infty} V^i \]
\[ V^+ = \bigcup_{i=1}^{\infty} V^i \]

\( V^+ \) consist of all words formed by concatenating a finite number but not null.

**Grammar**

A way to determine the structure of a language is with a grammar. In order to define a grammar, we need two kinds of symbols: non terminal, used to represent given subsets of the language, and terminal, the final systems that occur in the strings of the language.

A grammar defining formal language \( l \) is a quadruple

\[ G = (N,T,R,S) \]

\( N = \) set of non terminal

\( T = \) set of terminals

\( R = \) set of production

\( S = \) start symbol

\( T \) is alphabet

A grammar is a way to specify the set of all legal sentences of a language.

We write \( G = (N,T,P,\sigma) \). A production \( (A,B \in P) \) is usually written \( A \rightarrow B \)

A must include at least one non terminal symbol whereas

B can consists of any combination of non terminal and terminal symbols.

Let,

\( N = \{ \sigma, S \} \)

\( T = \{ a, b \} \)

\( P = \{ \sigma \rightarrow b\sigma, \sigma \rightarrow as, s \rightarrow bs, s \rightarrow b \} \)

Then \( G = (N,T,P,\sigma) \) is a grammar.
The language generated by G, written L(G) consists of all strings over T derivable from \( \sigma \)

For above grammar

\[ \sigma = b \sigma = b b \sigma = b b \sigma = b b a = b b a b \]

The only derivations from \( \sigma \) are

\[ \sigma = b \sigma \ldots = b^n \sigma = b^n a (n \geq 0) \]

\[ = b^n a b^m \quad (n \geq 0, m \geq 1) \]

Thus, \( L(G) \) consists of the strings over \{a, b\} containing precisely one a that end with b.

**Types of grammar (Chomsky hierarchy of grammar)**

1) **Unrestricted grammar (Type 0)**

If there is no restriction in the every production of the grammar, it is called unrestricted grammar.

Production rule

\[ \alpha \rightarrow \beta \]

Where,

\[ \alpha \in (N U T)^* \]

\[ \beta \in (N U T)^* \]

It is recognized by Turing machine

2) **Context Sensitive Grammar (Type 1)**

If every production is of the form

\[ \alpha A \beta \rightarrow \alpha \gamma \beta \]

(we may replace A with \( \gamma \) in the context of \( \alpha \) and \( \beta \))
\( \alpha, \beta \in (\text{NUT})^* \)

\( A \in \text{N} \)

The language generated by this grammar is recognized by Linearly Bounded Automata.

3) **Context Free Grammar (Type 2)**

If the production is of the form

\[ A \rightarrow \gamma \]

Where \( A \in \text{N} \)

\( \gamma \in (\text{NUT})^* \)

Left side = single non terminal
Right side = a word in one or more symbols.

\( \text{N} = \{S, A, B\} \)
\( \text{T} = \{a, b\} \)
\( \text{P} = \{S \rightarrow aA, A \rightarrow aAB, B \rightarrow b, A \rightarrow a\} \)

Start symbol \( S \).
The language generated by this grammar is recognized by Push down Automata.

4) **Regular Grammar (Type 3)**

If every production is of the form

\[ A \rightarrow a \] or \[ A \rightarrow aB \] or \[ A \rightarrow \gamma \]

Where, \( A, B \in \text{N} \)

\( a \in \text{T} \)

Left side = single non terminal
Right side = either single terminal or terminal followed by a non terminal.

Recognized by Finite State Automata.

\( \text{N} = \{S, A\} \)
\( \text{T} = \{a, b\} \)
\( \text{P} = \{S \rightarrow bS, S \rightarrow aA, A \rightarrow aS, A \rightarrow bA, A \rightarrow a, S \rightarrow b\} \)

Start symbol \( S \)
BNF

An alternative way to state or represent the productions context free grammar is using

1) Backus naur form and
2) Derivation tree

In BNF the non terminal symbols typically being with “<” and end with “>” the production
S→T is written S::<T

Production of the from S::<T₁, S::<T₂…… S::<Tₙ may be combined as S::<T₁[T₂|...........Tₙ

Grammar for integers

<digit>::=0|1|2|3|4|5|6|7|8|9

<integer>::=<signed integer>|< unsigned integer>

<signed interger>::=+-<digit>-<unsigned integer>

<unsigned integer>::=<digit>|<digit><unsigned integer>

Derivation of -108

<integer>::=<signed integer>::=<unsigned integer>
A regular grammar is a context free grammar and that a context free grammar with no production of the form \( A \rightarrow \lambda \) is a context sensitive grammar.

If every production is of the form

\[ A \rightarrow \alpha \] or \[ A \rightarrow \alpha B \] or \[ A \rightarrow \lambda \]

Where \( A, B \in N \), \( \alpha \in T^* - \{ \lambda \} \) then \( G \) is also a regular grammar.

\[ Q) \text{ The grammar } G \text{ defined by } T=\{a,b,c\}, \]

\[ N=\{ \sigma , A, B, C, D, E \}, \text{ with productions} \]

\[ \sigma \rightarrow aAB, \quad \sigma \rightarrow aB, \quad A \rightarrow aAC, \quad A \rightarrow ac, \quad B \rightarrow DC, \quad D \rightarrow b, \quad CD \rightarrow CE, \quad CE \rightarrow DE, \quad DE \rightarrow DC, \quadCc \rightarrow Dc \]

and starting symbol \( \sigma \) is context sensitive

\[ CD = CE = DE = DC \text{ (derivation of } dC \text{ from } CD) \]

The string \( a^3b^3c^3 \) is in \( L(G) \)

\[ aAB \rightarrow aaACB = aaaCCDc = aaaCDCc \]

\[ = aaaDCCc = aaaDCCDCc = aaaDDCCccc \]

\[ = aaaDDDCccc = aaabbbccc \]

The string \( a^3b^3c^3 \) is in \( L(G) \)

It can be show that

\[ L(G) = \{ a^n b^n c^n | n=1,2, \ldots \} \]
(Q) Write the regular grammar given by FSA given below

The terminal symbols are the input symbols \{a, b\}. The states E and O become the non terminal symbols. The initial state E becomes the starting symbol. The production correspond to the directed edges.

If there is an edge labeled x from S to S',

We write

\[ S \rightarrow xS' \]

In addition, if S is an accepting state we includes the production

\[ S \rightarrow \lambda \]

In case

\[ E \rightarrow aO, E \rightarrow bE, O \rightarrow aE, O \rightarrow bO \]

since O is accepting state we also include \[ O \rightarrow \lambda \]

Then the grammar \( G = (N, T, P, E) \) with \( N = \{O, E\}, T = \{a, b\} \) and \( P \) consisting of above productions generate the language \( L(G) \) which is the same as the set of string accepted by above FSA.

**Theorem**

Let \( A \) be a FSA given as a transition diagram. Let \( \sigma \) be the initial state. Let \( T \) be the set of input symbols and \( N \) be the set of states.

Let \( P \) be the set of productions.

\[ S \rightarrow xS' \]

If there is an edge labeled \( x \) from \( S \) to \( S' \) and

\[ S \rightarrow S' \]
If S is an accepting state let G be the regular grammar.

\[ G = (N,T,P,\sigma) \]

Then set of strings accepted by A is equal to \( L(G) \).

**Reverse case**

Given a regular grammar G, we want to construct a FSA to that \( L(G) \) is precisely the set of string accepted by A.

E.g.

Consider the regular grammar defined by

\[ T = \{a, b\} \]
\[ N = \{\sigma, c\} \]

With productions,

\[ \sigma \rightarrow b\sigma, \quad \sigma \rightarrow aC, \quad C \rightarrow bC, \quad C \rightarrow b \]

and starting symbol \( \sigma \)

The non terminal become state with \( \sigma \) as the initial state. For each production of the form

\[ S \rightarrow xS' \]

we draw an edge from state s to S' and label it x.

The productions:

\[ \sigma \rightarrow b\sigma, \quad \sigma \rightarrow aC, \quad C \rightarrow bC \]

give the graph as shown below

![Graph](image)

The production \( C \rightarrow b \) is equivalent to two productions.

\[ C \rightarrow bF \text{ and } F \rightarrow \lambda \]

where F is an additional non terminal symbol.
The production $F \to \lambda$ tells us that $F$ should be an accepting state.

This is not FSA

(i) vertex $C$ has no outgoing edge label $a$.

(ii) vertex $F$ has no outgoing at all

(iii) vertex $C$ has two outgoing edges labeled $b$

This is NDFA (Non Deterministic Finite Automata)

**Definition:**

NDFSA ‘$A$’ consists of

a) A finite set $I$ of input symbols.

b) A finite set $S$ of states

c) A next-state function $f$ from $S \times I$ into $P(S)$ (subset state $S$)

d) A subset $A$ of $S$ accepting states

e) An initial state $\sigma \in S$

We write $A = (I, S, f, A, \sigma)$

In DFSA, the next-state function takes us to a uniquely defined state, whereas in NDFSA, the next-state function takes us to a set of states.

**Example**

$I = \{a, b\}$

$S = \{\sigma, C, F\}$

$A = \{F\}$
The string $\alpha = bbabb$ is accepted because there is at least one path representing $\alpha$ that ends at an accepting state.

(Q) Convert NDFSA to DFSA
g (\{ \sigma \}, a)=\{ C \}
g (\{ \sigma \}, b)=\{ \sigma \}
g (\{ C \}, a)=\emptyset
g (\{ C \}, b)=\text{E}(C) \cup \text{E}(F)=\{ C, F \}
g (\{ C, F \}, a)=\emptyset
g (\{ C, F \}, b)=\text{E}(C) \cup \text{E}(F)=\{ C, F \}
g (\{ F \}, a)=\emptyset
g (\{ F \}, b)=\emptyset

(Q) Change N DFA to DFA
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow q_0$</td>
<td>${ q_0 }$</td>
<td>${ q_1, q_2 }$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>${ q_1, q_2 }$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

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

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

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

$= \emptyset \cup \{ q_1, q_2 \}$

$= \{ q_1, q_2 \}$ old state

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

$= \emptyset \cup \emptyset$

$= \emptyset$

(Q) Let $L$ be the set of strings accepted by the FSA shown below construct a FSA that accepts the strings

$L^R = \{ x_n \ldots x_1 | x_1 \ldots x_n \in L \}$
Reserve all arrows in above FSA and make $\sigma_3$ the starting state and $\sigma_1$ the accepting state. The result is a NDFSA that accepts $L^R$.

Change into DFSA

$E(\sigma_3)=\{\sigma_3\}, E(\sigma_2)=\{\sigma_2\}, E(\sigma_1)=\{\sigma_1\}$

Transitions

$(\sigma_3, b, \sigma_3) \quad (\sigma_2, b, \sigma_1) \quad (\sigma_1, a, \sigma_1)$

$(\sigma_3, b, \sigma_2) \quad (\sigma_1, a, \sigma_2)$

$(\sigma_1, b, \sigma_3)$

$g(\{(\sigma_3), a\}) = \emptyset$

$g(\{(\sigma_3), b\}) = E(\sigma_3)UE(\sigma_2)=\{\sigma_3, \sigma_2\}$

$g(\{(\sigma_3, \sigma_2), a\}) = \emptyset$

$g(\{(\sigma_3, \sigma_2), b\}) = E(\sigma_3)UE(\sigma_2)UE((\sigma_1)=\{\sigma_3, \sigma_2, \sigma_1\}$

$g(\{(\sigma_3, \sigma_2, \sigma_1), a\}) = E(\sigma_1)UE(\sigma_2)UE((\sigma_3)=\{\sigma_3, \sigma_2, \sigma_1\}$

$g(\{(\sigma_3, \sigma_2, \sigma_1), b\}) = E(\sigma_1)UE(\sigma_2)UE((\sigma_3)=\{\sigma_3, \sigma_2, \sigma_1\}$

$g(\{\emptyset\}, a) = \emptyset$

$g(\{\emptyset\}, b) = \emptyset$
Non Deterministic Finite State Automata

A non-deterministic finite automaton is a generalization of a finite state automaton so that at each state there might be several possible choices for the “next state” instead of just one.

Convert NDFSA to DFSA
**Regular expressions**

Regular expressions are useful for representing certain set of strings in an algebraic manner. Actually this describes the language accepted by finite state automata.

The regular expressions over an alphabet $\sum$ are the strings over the alphabet $\sum \cup \{, , \}$ such that the following hold ( $\sum$ consists of alphabet $\sum=\{a,b,\ldots\}$ /p symbols)

1) The empty set $\emptyset$ and each member of $\sum$ is a regular expression.
2) If $\alpha$ and $\beta$ are regular expressions then their concatenation $\alpha\beta$ is also a regular expression.
3) If $\alpha$ and $\beta$ are regular expressions then their union ($\alpha \cup \beta$) is also a regular expressions.
4) If $\alpha$ is a regular expression then so is the closure (iteration) $\alpha^*$. If $\alpha$ is R.E ($\alpha$) is also RE.
5) Nothing is a regular expression unless it follows from (1) through (4).

Regular languages can be characterized as languages defined by regular expressions.

The language represented by regular expression is given a function $L$, such that if $\alpha$ is any regular expressions then $L(\alpha)$ is the language represented by $\alpha$.

1) $L(\emptyset)=\emptyset$, $L(a)=\{a\}$, for each $a \in \sum$
2) If $\alpha$ and $\beta$ are regular expressions then $L(\alpha\beta)=L(\alpha).L(\beta)$
3) If $\alpha$ and $\beta$ are regular expressions

Then

$L(\alpha\cup\beta)=L(\alpha)\cup L(\beta)$

4) If $\alpha$ is a regular expression

Then

$L(\alpha^*)=L(\alpha)^*$

**What are the strings specified by the regular expression?**

Regular expression is **language generator** it generates the set of all strings for finite automata.

(FSA, pumping lemma are **language recognizer**.)

1) $10^* = A \text{ ‘1’ followed by any no. of 0’s including non zero.}$
2) $(10)^* = \text{Any number of copies of 10 (including the null string)}$
3) \(0U01 = \text{The string 0 or the string 01.}\)
4) \(0(0U1)^* = \text{Any string not beginning with 0.}\)
5) \((0^*1)^* = \text{Any string not ending with 0.}\)
6) \(\Sigma^* = \{0,1\}^* = \{\varepsilon, 0, 00, 01, 10, 11, \ldots\}\)
7) \(0^* + 1^* = \{\varepsilon, 0, 00, 000, \ldots\}\) (can it give combination only either 0 or 1.)
8) \(0^*1^* = \{\varepsilon, 0, 01, 001, 0011, \ldots\}\) (zero never come after 1)
9) \((01)^* = \{\varepsilon, 01, 0101, 010101, \ldots\}\)
10) \(01^* + 10^* = \text{either single 0 followed by any number of 1 or vice versa.}\)
11) \((0+1)^*.00\)

A language is regular iff it is accepted by FSA.

There are three operations on language that the operators of regular expression represent.

These operations are:

1) **Union:**

The union of two languages L and M denoted by \(L + M\) is the set of strings that are in either L or M or both.

E.g. \(L = \{001, 10, 111\}\) and \(M = \{\varepsilon, 001\}\)

Then,

\(L + M = \{\varepsilon, 10, 001, 111\}\)

2) **Concatenation:**

The concatenation of language L and M is the set of strings that can be formed by taking any string in L and concatenating it with any string in M and is denoted by \(L \cdot M\).

E.g. \(L = \{001, 10, 111\}\), \(M = \{\varepsilon, 001\}\)

Then,

\(L \cdot M = \{001, 10, 111, 001001, 10001, 111001\}\)

3) **The closure (or star or kleene star) of language**

L is denoted by \(L\) and represents the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions (i.e. the same string may be selected more than once) and concatenating all of them.

E.g. \(L = \{0, 1\}\) then \(L^*\) is the set of all the strings of 0’s and is with \(\varepsilon\) or \(^*\)

i.e. \(L = \{0, 1\}\)

\(L^* = \{0, 1\}^* = \{\varepsilon, 0, 01, 10, 11, 111, \ldots\}\)
The class of languages accepted by finite state automata (i.e. R.L.) is closed under.
1) Union
2) Concatenation
3) Kleene star
4) Complementation
5) Intersection

An automata with no final sets accepts or recognizes $\emptyset$

Empty string

Accepts the alphabet $a$

1) Union:

Let $L_1$ and $L_2$ be languages accepted by non-deterministic automata $M_1$ and $M_2$ respectively. Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$. We assume that $Q_1$ and $Q_2$ are disjoint sets.

We construct NDFSA $M$ that accepts $L(M_1)U \cup L(M_2)$ as follows

$M=(Q, \Sigma, \delta, q, F)$ where $q$ is new state not in $q_1$ or $Q_2$

$Q= Q_1UQ_2U(q)$ [Set of states of $M$]

$F= F_1UF_2$ [Final states of $M$]

Transition function of $M= \delta_1U\delta_2U\{(q, \epsilon)\rightarrow((q, \epsilon)\rightarrow q_2)\}$

Or,

$F_1UF_2\{(s,e,s_1),(S,e,s_2)\}$
2) **Concatenation** :
Let $M_1$ and $M_2$ be NDFA. We construct a NDFA $M$ such that $L(M) = L(M_1) \rightarrow L(M_2)$. The construction is shown in the figure.

$$M$$ operates by simultaneeously $M_1$ for a while and then jumping non-deterministically from a finite state of $M_1$ to the initial state of $M_2$. Therefore $M$ imitates $M_2$.

Formally,
Let,
$M_1 = (S'_1, I, f_1, s_1, F_1)$ or $Q_1, , q_1, F_1)$ or $(Q_1, \delta_1, q_1, F_1)$
$M_2=(S'_2, I, f_2, s_2, F_2)$  $s_1$ and $s_2$ are disjoint
$L(M) = L(M_1).L(M_2)$
$M= (S, I, f, \sigma ,F)$
Set of states of $M=${$S'=S'_1US'_2=union of M_1 and M_2$
Final states (F)= final states of $M_2$ (F_2)
Transition $S = Union of M_1 and M_2 plus (F_1\times e \times \{s_2\})$
$F=F_1UF_2U\{(F_1,c)\rightarrow S_2\}$

3) **Kleene star** :
Let $M_1$ be a NDFA we construct a NDFA $M$ such that $L(M) = L(M_1)^*$. The construction is show below.
L(M)=L(M_{1})
States of M=states of M_{1} plus s'_{1}
Final states of M= Final states of M_{1} plus s'_{1}
Transition of M= Transition of M_{1} plus (F \{e\} \X \{s_{1}\})
F=F_{1}U F_{2}U \{(F,e,S_{1})\}

**Pumping lemma (Theorem) for Regular Language**

Pumping lemma is a powerful technique for showing certain language to be non-regular.

**Statement**

Let L be a regular language. Then there exit a constant n (which depend on L) such that for every string w in L, such that \(|w| \geq n\).

We can break w into three sub strings

1) y ≠ \epsilon
2) |xy| ≤ n
3) For all i≥0, string xy'z is also in L

Technique to identify the class of language for finite automata is pumping lemma.

Show that the language
L={a^{n}b^{n} \ n>0} is not regular
w\in L
w=a^{p}b^{p}

According to the pumping lemma
x=a^{q}
y=a^{r}
z=a^{p-(q+r)}b^{p}

Since,
xy\in L
Now,
xy^{2}z
\[ a^q(a^r)^2(a^{p-(q+r)}b^p) \]
\[ = a^{q+2r}a^{p-(q+r)}b^p \]
\[ = a^{p+r}b^p L \]

Since \( xy^2z \) is not the from \( a^pb^p \) therefore \( xy^2z \in L \). It is contradiction. Hence \( L \) is not regular.

**Closure properties of Regular Language**

If \( L \) and \( M \) be regular language. Then following languages are all regular.

1) Union : \( L \cup M, L+M \)
2) Intersection : \( L \cap M \)
3) Complement : \( \bar{N}, \bar{L} \) and \( \bar{M} \), \( i = \sum^* \backslash L \)
4) Difference : \( L \backslash M = L \cap \bar{M} \)
5) Reversal \( L^R = W^R : w \in L \)
6) Closure: \( L^* \)
7) Concatenation : \( L.M \)
8) Homomorphism
   \[ h(L) = \{ h(w) \mid w \in L, h \text{ is homomorphism} \} \]
9) Inverse homomorphism
   \[ h^{-1}(L) = \{ w \in \mid h(w)L, h: \Sigma \rightarrow \Delta \text{ is a homomorphism} \} \]

\[
\text{homomorphism}\n\text{h: } \Sigma^* \rightarrow \Theta^*  \text{ (alphabet } \Sigma, \Theta) \text{ } \\
w = a_1a_2 \ldots \ldots a_n \in \Sigma^* \\
h(w) = h(a_1)h(a_2) \ldots \ldots h(a_n) \\
h(L) = \{ h(w) \mid w \in L \}
\]

**Example**

Let \( h: \{0,1\}^* \rightarrow \{a,b\}^* \) be defined by

\[ h(0) = a \ b \text{ and } h(1) = \epsilon \]

Now,

\[ h(0011) = abab \]
\[ h(L(10^*1)) = L((ab^*)) \]