UNIT - 1

Automata -

Basic Machine or Automation:

An automation is defined as a system where energy, materials, and information are transformed, transmitted, and used for performing some functions without direct participation of man. E.g.: Automatic Machine tools, Automatic Packing Machines, automatic photo printing machines, etc.

\[ I_1 \rightarrow \text{Automation} \rightarrow O_1 \]
\[ I_2 \rightarrow \text{Automation} \rightarrow O_2 \]
\[ I_p \rightarrow \text{States} \rightarrow O_p \]

Characteristics of automation -

1. Input \( I_1, I_2, \ldots, I_p \)
2. Output \( O_1, O_2, \ldots, O_p \)
3. States \( q_1, q_2, \ldots, q_p \)
4. State transition
5. Output relation

Finite Automaton or Finite State Machine:

A finite automation or finite state machine can be represented by 5-tuple \( (Q, \Sigma, \delta, q_0, F) \) where:

- \( Q \) \text{ - finite nonempty set of states}
- \( \Sigma \) \text{ - finite nonempty set of inputs}
- \( \delta \) \text{ - Transition function} \( (Q \times \Sigma) \to Q \)
- \( q_0 \) \text{ - Initial State} \( (q_0) \to Q \)
- \( F \) \text{ - Set of final states} \( (F) \to Q \)

Block Diagram of finite automation
Transition Graph (Transition System):
A transition graph or a transition system is a finite directed labelled graph in which each vertex (or node) represents a state and the directed edges indicate the transition of a state and the edges are labelled with input/output.
A transition system is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\)

Transition Matrix (Transition Table):
To show a transition graph or a transition system in the form of a matrix or table is known as transition matrix or transition table.

Deterministic FSM or DFA:
A deterministic finite state machine or finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where \(\delta\) - transition function \((Q \times \Sigma \rightarrow Q)\)

Non-deterministic FSM or NFA:
A non-deterministic finite state machine or finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where \(\delta\) - transition function \((Q \times \Sigma \rightarrow 2^Q)\)

Equivalence of DFA and N DFA:
For every N DFA, there exists a DFA which simulates the behaviour of N DFA. Alternatively, if \(L\) is the set of accepted by N DFA, then there exists a DFA which also accepts \(L\)

-> Conversion of N DFA to DFA

Mealy and Moore machine:
A Mealy machine is a 6-tuple \((Q, \Sigma, \Delta, \delta, \lambda, q_0)\) where
\(Q\rightarrow\) finite set of states
\(\Sigma\rightarrow\) Input alphabet
\(\Delta\rightarrow\) Output alphabet
\(\delta\rightarrow\) Transition function \((\Sigma \times \Delta \rightarrow Q)\) \((Q \times \Delta \rightarrow \Omega)\)
1. Output function: $\delta: Q \times \Sigma \rightarrow \Delta$ (where $\Sigma$ and $\Delta$ are finite alphabets).

2. Initial state: $q_0 \rightarrow$ Initial state

3. Moore machine is a 6-tuple $(Q, \Sigma, \Delta, \delta, q_0)$ where all symbols except $\Sigma$ have the same meaning as in the Mealy machine.


5. Mealy machine to Moore machine (Add $\delta(q, \sigma)$ for all $q \in Q$).

6. Minimization of finite automata:

   - Definition 1: Two states $q_1$ and $q_2$ are $\equiv$ equivalent (denoted $q_1 \equiv q_2$) if both $\delta(q_1, \sigma)$ and $\delta(q_2, \sigma)$ are final states or both are non-final states for all $\sigma \in \Sigma$.

   - Definition 2: Two states $q_1$ and $q_2$ are $K$-equivalent ($K \geq 0$) if both $\delta(q_1, \sigma)$ and $\delta(q_2, \sigma)$ are final states or both non-final states for all strings $\sigma$ of length $K$ or less. In particular, any two states are 0-equivalent and any two non-final states are also 0-equivalent.

   - Take, $\Pi_0 = \{ q_i | i = 1 \}$, all other states $q_i$ then final $\Pi_1, \Pi_2, \ldots$.

7. Two-way finite automata:

   - It is like a deterministic finite automata except that the reading head can go backwards as well as forwards on the infinite tape.

   - It is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where $\delta$ is a function from $Q \times \Sigma \rightarrow Q \times \{ L, R \}$ indicates the direction of head movement, and $M(Q, \Sigma, \delta, q_0, F)$.

8. Start state:

9. Configuration $(p, u, v)$ indicates that the machine is in state $p$ with the head on the first symbol of $v$ and with $u$ to the left of the head.

   - If $v = \lambda$, configuration $(p, u, \lambda)$ means that it has completed its operation on $u$ and ended up in state $p$. 
\[ R = \lambda + RP \]
\[ = \lambda + (\lambda P^*) P \]
\[ = \lambda (\lambda + P^* P) \]
\[ = \lambda P^* \]
\[ \lambda = \epsilon \]

**Regular Sets and Regular Grammars**

1. Regular Expression:
   - The regular expressions are useful for representing certain set of strings in an algebraic fashion. Actually then describe the languages accepted by finite state automata.

2. Regular Set:
   - Any set represented by a regular expression is called a regular set.

3. Identities for regular expression:
   - (1) \( \phi + R = R \)
   - (2) \( \phi R = R \phi = \phi \)
   - (3) \( \lambda R = R \lambda = \lambda R \)
   - (4) \( \lambda^* = \lambda \) and \( \phi^* = \lambda \)
   - (5) \( R + R = R \)
   - (6) \( R^* R^* = R^* \)
   - (7) \( RR^* = R^* R \)
   - (8) \( (R^*)^* = R^* \)
   - (9) \( \lambda + RR^* = R^* = \lambda + R^* R \)
   - (10) \( (PQ)^* RP = P(QP)^* \)
   - (11) \( (P + Q)^* = (P^* Q^*)^* = (P^* + Q^*)^* \)
   - (12) \( (P + Q) R = PR + QR \) and \( R (P + Q) = RP + RQ \)

**Theorem:**

Let \( P \) and \( Q \) be two regular expressions over \( \epsilon \). If \( P \) does not contain \( \lambda \), then the following equation on \( R \), namely
\[ R = \lambda + RP \]
has a unique solution given by
\[ R = \lambda P^* \]

4. Finite automata and regular expression:

Finite automaton with \( \lambda \)-move into without \( \lambda \)-move:

Suppose we want to replace a \( \lambda \)-move from vertex \( v_1 \) to vertex \( v_2 \). Then we proceed as follows:
**STEP 1:** Find all edges starting from $V_2$.

**STEP 2:** Duplicate all these edges starting from $V_1$, without changing the edge labels.

**STEP 3:** If $V_1$ is an initial state, make $V_2$ also as initial state.

**STEP 4:** If $V_2$ is a final state, make $V_1$ also as the final state.

→ **Algorithm method using Dudeney's theorem:**

(1) Find $q_i = q_i \cdot a_1^* + q_j \cdot a_2^* + \ldots + q_m \cdot a_n^*$, for all states $q_i$, $q_j$, $q_2$, ..., $q_m$, $\rightarrow$ simulation present in $a_1$,$a_2$,$\ldots$, $a_n$ $\rightarrow$ label on edges = transition system.

(2) Reduce the unknowns by repeated substitution.

(3) Find the equation of final state.

→ Used in construction of regular expression from state diagram.

→ **Constructions of finite automata equivalent to a regular expression**

(1) $r = a + b$

(2) $r = a \cdot b$

(3) $r = a^+$

(4) $r = a^*$
5. **Myhill-Nerode Theorem:**

It can be stated as follows:

The following three statements are equivalent:

1. A language L is regular.
2. L is the union of some of the equivalence classes of a right-invariant equivalence relation of finite index.
3. L is of finite index.

An equivalence relation R on $\Sigma^*$ is said to be right invariant if for every $x, y \in \Sigma^*$, if $xRx$, then for every $z \in \Sigma^*$, $xzRyz$.

Also, an equivalence relation is said to be of finite index, if the set of its equivalence classes is finite.

6. **Pumping Lemma for regular sets:**

Let $M = (Q, \Sigma, S, q_0, F)$ be a finite automaton with n states. Let L be the regular set accepted by M. Let $w \in L$ and $|w| \geq m$. If $m \geq n$, then there exist $x, y, z$ such that $w = xyz$, $y \neq \lambda$ and $xy^iz \in L$ for each $i \geq 0$.

7. **Application of Pumping Lemma:**

It is used to prove that certain sets are not regular. Steps needed for proving that a given set is not regular:

**STEP 1:** Assume that L is regular.

**STEP 2:** Choose a string $w$ such that $|w| \geq n$ when $n$ be the number of states. Using pumping lemma, $w = xyz$.

**STEP 3:** Find a suitable integer $i$ such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

8. **Closure Properties of Regular Sets:**

1. Set union
2. Set intersection
3. Concatenation
4. Transpose
5. Complementation
6. Closure (intersection)