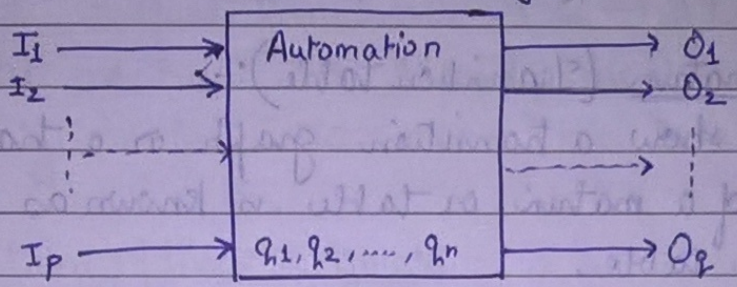
UNIT - 1

Automata -

① **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. Eg:- Automatic Machine tools, Automatic Packing Machines, automatic photo printing machines etc.



Model of discrete automation

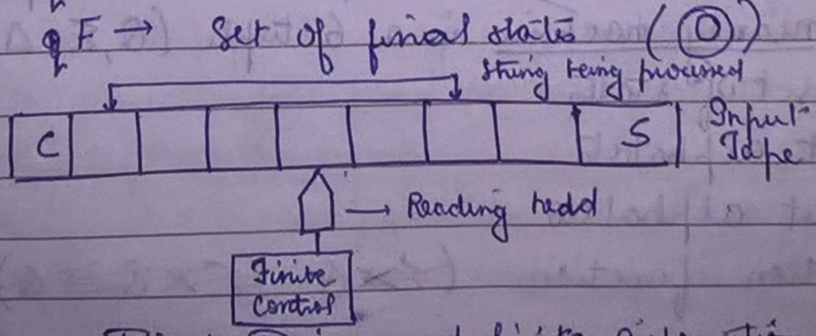
Characteristics of automation -

(1) Input $\rightarrow$ $I_1, I_2, \ldots, I_p$
(2) Output $\rightarrow$ $O_1, O_2, \ldots, O_p$
(3) States $\rightarrow$ $q_1, q_2, \ldots, q_p$
(4) state relation
(5) Output relation

② **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.

1 $Q \rightarrow$ finite nonempty set of states
$\Sigma \rightarrow$ finite non empty set of inputs
$\delta \rightarrow$ Transition function $(Q \times \Sigma \rightarrow Q)$
$q_0 \rightarrow$ Initial state ($\longrightarrow q_0$)
$F \rightarrow$ Set of final states (◎)



Block Diagram of finite automation

③ **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 $(Q, \Sigma, \delta, q_0, F)$

④ **Transition matrix (Transition table):-**

To show a transition graph or a transition system in form of a matrix or table is known as transition matrix or transition table.

FSM without output

⑤ **Deterministic FSM or FA :-**

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

Non-deterministic FSM or FA :-

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

⑥ **Equivalence of DFA and NDFA :-**

For every NDFA, there exists a DFA which simulates the behaviour of NDFA. Alternatively, if L is the set ~~of~~ accepted by NDFA, then there exists a DFA which also accepts L.

→ Conversion of NDFA to DFA

⑦ **Mealy and Moore machines :-** → (n inputs → n outputs)

A mealy machine is a 6-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

$Q \rightarrow$ finite set of states

$\Sigma \rightarrow$ Input alphabets

$\Delta \rightarrow$ Output alphabets

$\delta \rightarrow$ transition function ~~($\Sigma \times Q$~~ ~~($\Sigma \times Q \rightarrow Q$)~~ $(Q \times \Sigma \rightarrow Q)$

$\lambda \rightarrow$ Output function ~~($Q \to \Delta$)~~ ~~($\Sigma \times Q \to \Delta$)~~ ($Q \times \Sigma \to \Delta$)

$q_0 \rightarrow$ Initial state

$\rightarrow$ (n inputs $\rightarrow$ $n+1$ outputs)

A <u>moore machine</u> is a 6-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where all symbols except λ have the same meaning as in the ~~Mealy~~ machine.

$\lambda \rightarrow$ Output function $(Q \rightarrow \Delta)$

→ Moore machine to Meanly machine

→ Meanly machine to moore machine (Find $Q \times \Delta$, then Q/r)

(8) <u>Minimization of finite automata</u> :-

<u>Definition 1</u> :- Two states q_1 and q_2 are equivalent (denoted $q_1 \equiv q_2$) if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or both of them are non-final states for all $x \in \Sigma^*$.

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

→ Take, $\pi_0 = \{\{F\}, \{\text{all other states}\}\}$ then find $\pi_1, \pi_2, \ldots$ so on.

(9) <u>Two-way Finite automata</u> :- (read only Turing machine)

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

It is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where δ is a function from $Q \times \Sigma$ into $Q \times \{\leftarrow, \rightarrow\}$; $\leftarrow$ or $\rightarrow$ indicates the direction of head movement. or <u>$M(Q, \Sigma, \vdash, \dashv, \delta, s, t, r)$</u>.

s → start state
t → accept state
r → reject state

A 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, Λ) means that it has completed its operation on u and ended up in state p.

$$R = Q + RP$$
$$= Q + (QP^*)P$$
$$= Q(\Lambda + P^*P) \qquad \Lambda \equiv \epsilon$$
$$= QP^*$$

Regular Sets and Regular Grammers -

① Regular Expression :-

The regular expressions are useful for representing certain set of strings in an algebric fashion. Actually these 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) $\Lambda R = R\Lambda = R$

$\Lambda . \epsilon$ (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) $(PQ)^*P = P(QP)^*$

(11) $(P+Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$

(12) $(P+Q)R = PR + QR$ and $R(P+Q) = RP + RQ$

Arden's Theorem :-

Let P and Q be two regular expressions over Σ. If P does not contain Λ, then the following equation in R, namely $\boxed{R = Q + RP}$

has a unique solution given by $\boxed{R = QP^*}$

④ Finite automata and regular expression :-

→ Finite automaton with Λ-moves into without Λ-moves :-

STEP 1:- Suppose we want to replace a Λ-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

→ **Algebraic method using Arden's theorem**:-

(1) Find $q_i = q_i\alpha_1 + q_j\alpha_2 + \dots + q_m\alpha_n$ for all states present in transition system

$q_i, q_j, \dots, q_m$ → state

$\alpha_1, \alpha_2, \dots, \alpha_n$ → label on edges

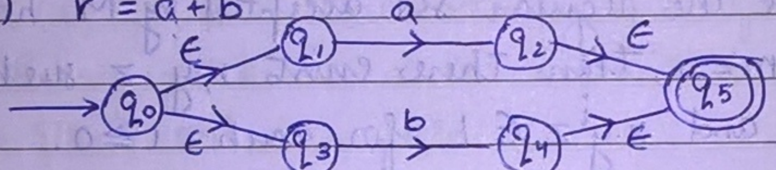
(2) Reduce the unknowns by repeated substitution

(3) Find the equation of final state.

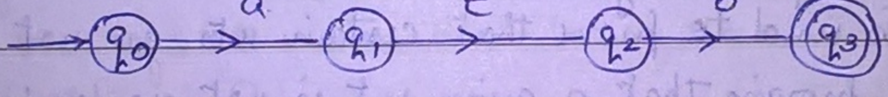
→ Used in construction of regular expression from state diagram

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

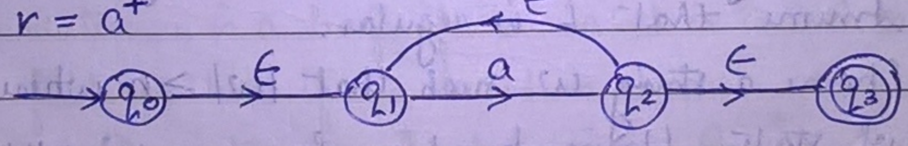
(1) $r = a + b$



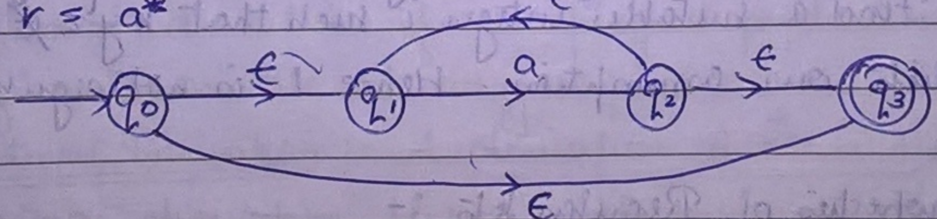
(2) $r = a.b$



(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 equivalent relation of finite index.

(3) I_L is of finite index.

An equivalence relation R on Σ^* is said to be right invariant if for every $x, y \in \Sigma^*$, if xRy 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 $M = (Q, \Sigma, \delta, q_0, F)$ 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 exists 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 $|w| \geq n$ where 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 (iteration)