THEORY OF AUTOMATA & FORMAL LANGUAGES

S.B.VERMA

Peter linz
K.L.P Mishra
Aho ullman

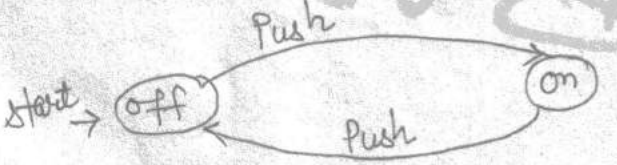
COMPUTE_TRANSITION_FUNCTION (P, Σ)

```
1- m ← length[P]
2- for q ← 0 to m
3-     do for each character a ∈ Σ
4-         do K ← min(m+1, q+2)
5-            repeat K ← K-1          ← FALSE
6-            until P_K ] P_q a       ←
7-            δ(q,a) ← K              ← TRUE
8- return δ
```

$P_K] P_q a \Rightarrow P_K$ is a suffix of $P_q a$.

Finite Automata:- finite automata are useful model for many important kinds of H/W and S/W. some important kinds are—

① - S/W for designing and checking the behavior of digital circuits.
② - The "lexical analyzer" of a typical compiler.
③ - To find occurence of words.
④ - To verify systems of all types that have a finite no. of distinct states, such as comm. protocols or protocols for secure exchange of information.



* The purpose of a state is to remember the relevant portion of the system's history. Since there are only a finite no. of states, the entire ~~entry~~ history can not be remembered, so the system must be designed carefully, to remember what is important and forget what is not.

STRINGS :- A string (or word) is a finite sequence of symbols from Σ. for example, a, b and c are symbols and abcb is a string. $|w|$ denotes the length of the string. The empty string, denoted by ϵ, is the string consisting of zero symbols. Thus $|\epsilon| = 0$.

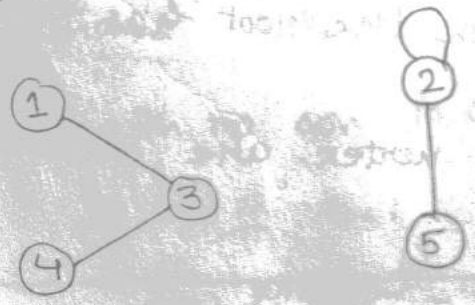
PREFIX :- A prefix of a string is any number of leading symbols of that string.

SUFFIX :- A suffix is any number of trailing symbols.

ALPHABET :- An alphabet is a finite set of symbols. A formal (language) is a set of strings of symbols from one alphabet. The empty set, ϕ and the set consisting of the empty string $\{\epsilon\}$ are languages.

⇒ The set of palindromes (string that read the same forward & backward) over the alphabet $\{0,1\}$ is an infinite language. ie 10101, 101, ϵ, 0, 1.

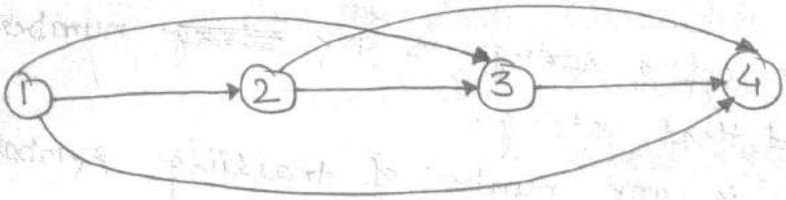
GRAPH :- A graph, denoted $G = (V, E)$ consists of a finite set of vertices V and a set of pairs of vertices E called edges. ie - $V = \{1,2,3,4,5\}$ and $E = \{(n,m) | n+m = 4$ or $n+m = 7\}$.



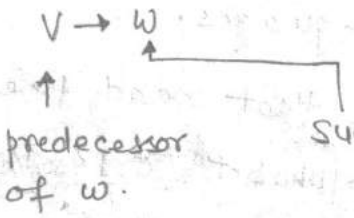
⇒ A path in a graph is a sequence of vertices $v_1, v_2 \dots v_K$, $K \geq 1$, such that there is an edge (v_i, v_{i+1}) for each i, $1 \leq i \leq K$. The length of the path is $(K-1)$.

Directed graph :-

A directed graph or (dia graph), consists of a finite set of vertices V and a set of ordered pairs of vertices E called arcs, denoted as $V \rightarrow W$ from V to W.

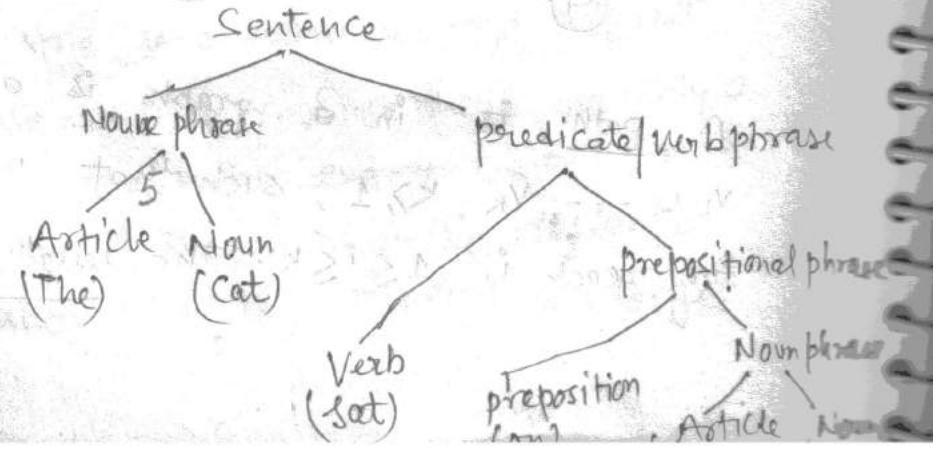


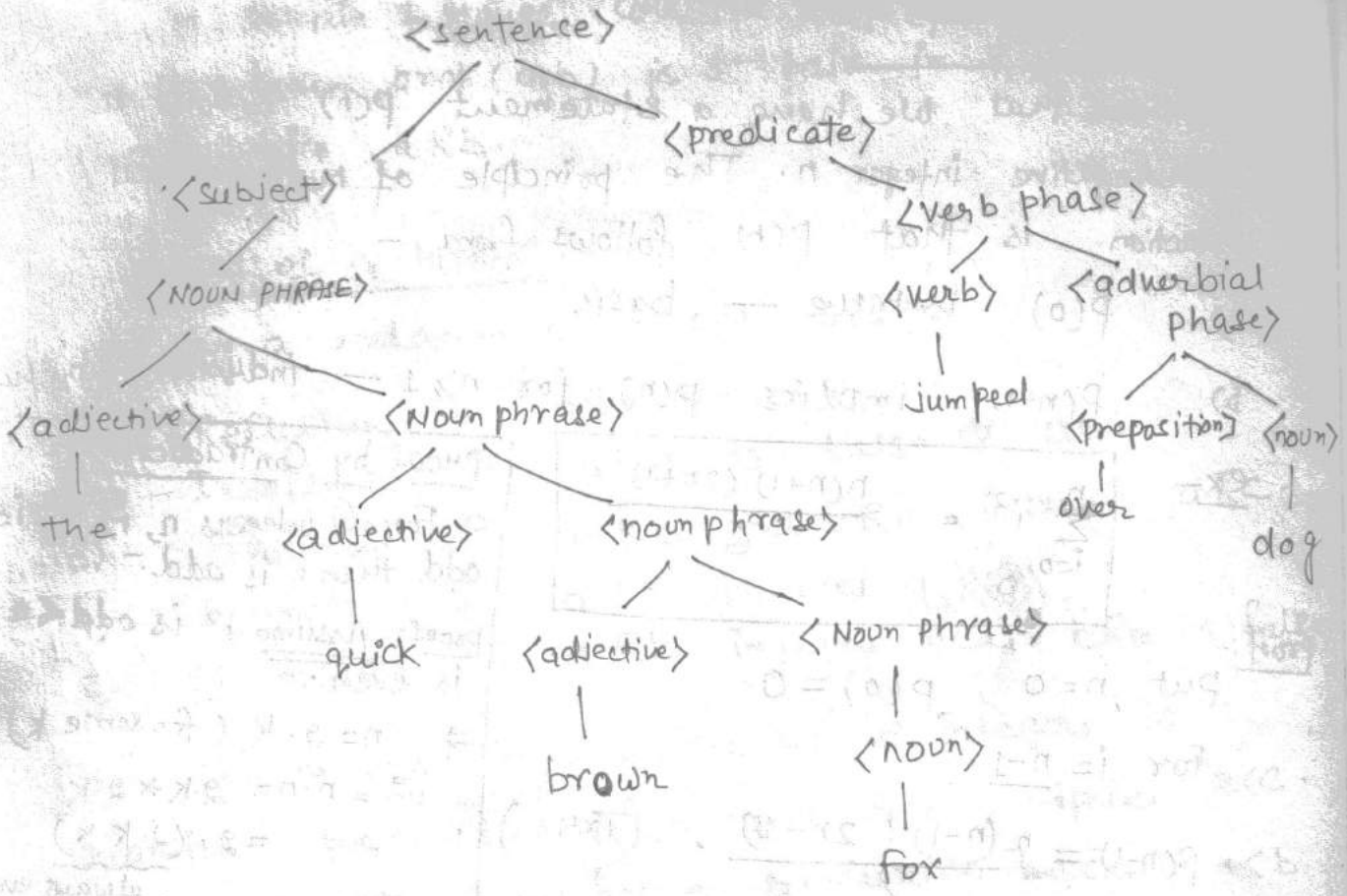
A path in a digraph is a sequence of vertices $V_1, V_2 \cdots V_K$, $K \geq 1$, such that $V_i \rightarrow V_{i+1}$ is an arc for each $i, 1 \leq i \leq K$.



TREE :- A tree is a digraph with the following properties-

1) – There is one vertex, called the root, that has no predecessors and from which there is a path to every vertex.
2) – Each vertex other than the root has exactly one predecessor.
3) – The successors of each vertex are ordered "from the left".





⇒ A successor of a vertex is called a son, and the predecessor is called the father. If there is a path from vertex V_1 to vertex V_2, then V_1 is said to be an ancestor of V_2, and V_2 is said to be a descendant of V_1.

⇒ Vertex with no son is called the leaf.

Finite Automata (FA) is the simplest machine to recognize patterns. The finite automata of FSM is an abstract machine that has five elements. It has a set of states and rules from one state to another but it depends upon the applied input symbol. Basically it is an abstract model of a digital computer.

Inductive Proofs :-

Suppose that we have a statement P(n) about a non negative integer n. The principle of mathematical induction is that P(n) follows from -

a) P(0) is true — basis

b) P(n-1) implies P(n) for $n \geq 1$ — induction hypothesis.

ex- $$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Proof :-

Put n=0, P(0) = 0.

for i = n-1.

$$P(n-1) = \frac{n(n-1)(2n-1)}{6} \quad \text{(True)}$$

$$P(n) = P(n-1) + n^2$$

$$= \frac{n(n-1)(2n-1)}{6} + n^2$$

$$P(n) = \frac{n(n+1)(2n+1)}{6}$$

Proof by Contradiction :-

ex- For all integers n, if n^2 is odd, then n is odd.

Proof :- Assume n^2 is odd & n is even.

$\Rightarrow$ $n = 2 \cdot K$ (for some K)

$n^2 = n \cdot n = 2 \cdot K * 2 \cdot K$

$= 2 \cdot (2 \cdot K \cdot K)$

(always even)

$n^2 = 2 \cdot$ (some even integer)

(even)

$n^2 \Rightarrow$ is always even

So, since n is even, n^2 which is the product of n with itself, is also even. This ~~is the~~ Contradicts the assumption.

■

Relations :- A binary relation is a set of pairs. The first component of each pair is chosen from a set called the domain & second component is chosen from a set called the range.

⇒ If the domain & range are the same set S, and if R is a relation and (a,b) is a pair in R, then we often write aRb.

Properties of Relations :-

We say a relation R on set S is—

1) – Reflexive if aRa ∀ a∈S.
2) – Irreflexive if aRa is false ∀ a∈S.
3) – Transitive if aRb and bRc implies aRc.
4) – Symmetric if aRb implies bRa.
5) – Asymmetric if aRb implies that bRa is false.

Ex. The relation (<) on the set of integers is transitive because a<b and b<c implies a<c. It is asymmetric and hence irreflexive because a<b implies b<a is false.

EQUIVALENCE RELATION :-

A Relation (R) that is reflexive, symmetric, and transitive is said to be equivalence relation.

Important ⇒ An equivalence relation R on a set S partitions S into disjoint nonempty equivalence classes. ie- $S = S_1 \cup S_2 \cdots$, where for each i & j with $i \neq j$:-

1) – $S_i \cap S_j = \phi$
2) – for each a and b in S_i, aRb is true.
3) – for each a in S_i and b in S_j, aRb is false.

⇒ S_i are equivalent classes, number of classes may be infinite.

Ex:- Congruence modulo an integer m is an equivalence relation.

$$i \equiv j \bmod m$$

$$\Downarrow$$

$$i \bmod n = j \bmod n$$

or we may write $i \equiv_m j$

⇒ The equivalence classes of $\equiv_m$ are m in number:-

$$\{\ldots\ -m, 0, m, 2m \ldots\}$$

$$\{\ldots\ (-m-1), 1, m+1, 2m+1, \ldots\}$$

$$\vdots$$

$$\{\ldots\ -1, m-1, 2m-1, 3m-1 \ldots\}$$

Exercise.

1)- prove that $\sum_{i=0}^{n} i = \frac{n(n+1)}{2} = S(n)$

proof:- put $n=0$

$S(0) = 0$

Let $S(n-1)$ is true ie- $\frac{n(n-1)}{2} = S(n-1)$

$$S(n) = S(n-1) + n$$

$$= \frac{n(n-1)}{2} + n = \frac{n(n+1)}{2}$$

proved.

Definition of an Automaton:- An automaton is defined as a system where energy, materials and information are tranformed, transmitted and used for performing some functions without direct participation of man. Ex- automatic machine tools, automatic packing machines and automatic photo printing machines. In computer science term 'automaton' means 'discrete automaton' and is defined in a more abstract way show in figure — 9

FINITE AUTOMATA

A finite automata consists of a finite set of states and a set of transitions from one state to another state that occur on input symbols chosen from an alphabet Σ.

We formally denote a finite automaton by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite nonempty set of states.

Σ is a finite nonempty input alphabet

$q_0 \in Q$ is the initial state

$F \subseteq Q$ is the set of final states.

δ is the transition function mapping from $Q \times \Sigma \to Q$

ie $\delta(q, a)$ is a state for each q and input symbol a.

Extended transition function :- $(\hat{\delta})$.

The extended transition function $\hat{\delta}$ is applied to a state and a string rather than a symbol.

$$\hat{\delta}: Q \times \Sigma^* \longrightarrow Q$$

formally we define $\hat{\delta}$ as follows:

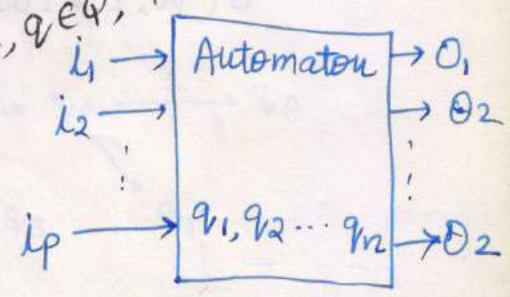
1) - $\hat{\delta}(q, \epsilon) = q$ → (without reading an I/P symbol, FA can't change its state)

2) - $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$, for all string w and input symbols a.

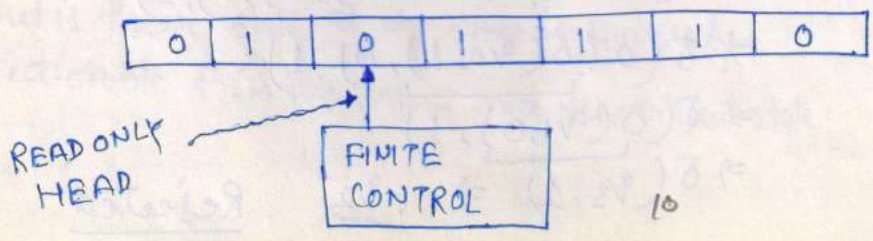
3). $\hat{\delta}(q, a) = p$ for some $a \in \Sigma, q \in Q, p \in Q$.

⇒ Since $\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \epsilon), a)$

$$\boxed{\hat{\delta}(q, a) = \delta(q, a)}$$



(Discrete automaton)

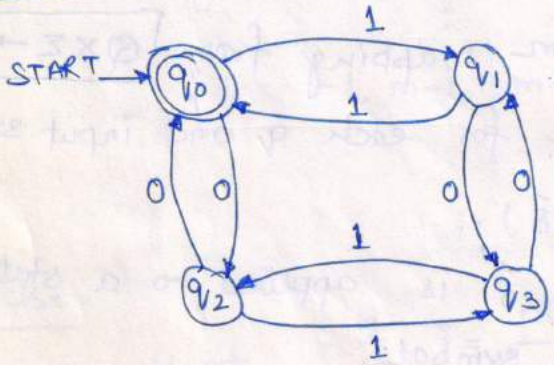


⇒ A string x is said to be accepted by a finite automaton $M=(Q,\Sigma,\delta,q_0,F)$ if $\hat{\delta}(q_0,x)=p$, for some p in F.

⇒ The Language accepted by M, designated by $L(M)$, is the set $\boxed{\{x \mid \hat{\delta}(q_0,x) \in F\}}$

⇒ A language is a regular set if it is the set accepted by some finite automaton.

Example:- Even No. of 0's & 1's.



States	Inputs 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

The FA is ⟶ $M=(Q,\Sigma,\delta,q_0,F)$ where $Q=\{q_0,q_1,q_2,q_3\}$

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

$F=\{q_0\}$

Suppose $\omega = 11\bullet101$ is input to M

$\hat{\delta}(q_0, 11\bullet101) = \delta(\hat{\delta}(q_0, 11\bullet10), 1)$

$\Rightarrow \delta(\delta(\hat{\delta}(q_0, 11\bullet1), 0), 1)$

$\Rightarrow \delta(\delta(\delta(\hat{\delta}(q_0, 11), 1), 0), 1)$

$\Rightarrow \delta(\delta(\delta(\delta(\hat{\delta}(q_0, 1), 1), 1), 0), 1)$

$\Rightarrow \delta(\delta(\delta(\delta(q_1, 1), 1), 0), 1)$

$\Rightarrow \delta(\delta(\delta(q_0, 1), 0), 1)$

$\Rightarrow \delta(\delta(q_1, 0), 1)$

$\Rightarrow \delta(q_3, 1) \Rightarrow q_2$ Rejected

Let $w = 00$

$$\hat{\delta}(q_0, 00) = \delta(\hat{\delta}(q_0, 0), 0)$$

$$\Rightarrow \delta(q_2, 0)$$

$$\Rightarrow q_0 \quad \text{(accepted)}$$

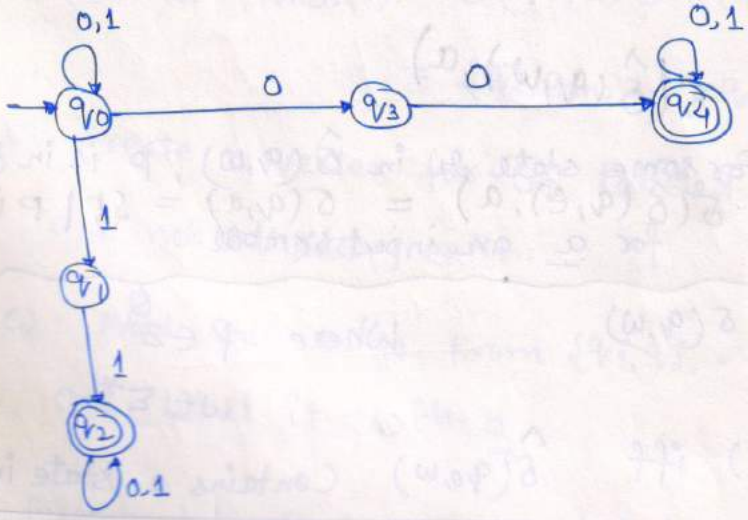
Final state.

$\Rightarrow$ Thus $w = 00$ is in $L(M)$. We can say that $L(M)$ is the of strings with an even no. of 0's and even no. of 1's.

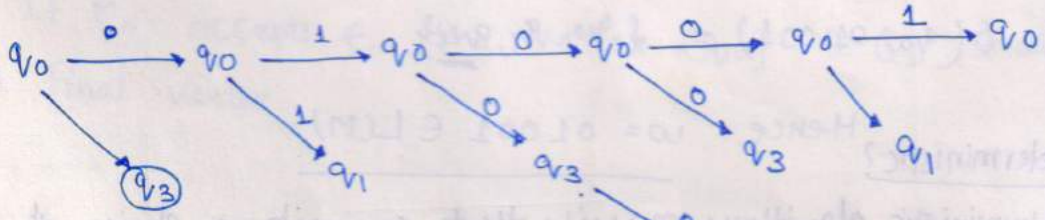
NON-DETERMINISTIC FINITE AUTOMATA :-

A F.A that allow zero, one or more transitions from a state on the same input.

ex-



Let $w = 01001$.



Final state

* Digital Computers are completely deterministic; that is their state at any time is uniquely predictable from the input and the present state.

Formally we denote that a NFA is a 5 tuple $(Q, \Sigma, \delta, q_0, F)$ where Q, Σ, q_0 & F have the same meaning as for a DFA, but δ is a map from $\boxed{Q \times \Sigma \to 2^Q}$

ie $\boxed{\delta(q,a) = \{p \mid p \in Q \text{ and } \delta(q,a)\}}$

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

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

⇒ The function δ can be extended to a function $\hat{\delta}$ mapping $\boxed{Q \times \Sigma^* \text{ to } 2^Q}$ and reflecting sequences of inputs as follows:—

a) – $\hat{\delta}(q, \epsilon) = \{q\}$ $\quad \delta(\hat{\delta}(q,w), a)$

b) – $\hat{\delta}(q, wa) = \{p \mid$ for some state r in $\hat{\delta}(q,w)$, p is in $\delta(r,a)\}$ $\quad \delta(\hat{\delta}(q,\epsilon), a) = \delta(q,a) = \{p \mid p \text{ in } \delta(q,a)\}$

c) – $\hat{\delta}(q,a) = \delta(q,a)$ for a an input symbol.

d) – $\delta(P, w) = \bigcup_{q \text{ in } P} \delta(q,w)$ Where $P \in 2^Q$, $w \in \Sigma^*$

⇒ A string $w \in L(M)$ iff $\hat{\delta}(q_0, w)$ contains a state in F.

Consider the previous example –

$\delta(q_0, 01001) = \{q_0, q_1, q_4\}$

Hence $w = 01001 \in L(M)$

Why Non-deterministic?

⇒ Many deterministic algorithms require that one make a choice at some stage. A typical ex. is a game-playing program. frequently, the best move is not known, but can be found using an exhausive search with back-tracking. When several moves are possible, we choose one and follow it until it becomes clear whether or not it was best. If not, we retreat to the last decision point & explore the other choices.

EQUIVALENCE OF DFA'S and NFA'S :-

Theorem:- Let L be the language accepted by a NFA $M_N(Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a DFA $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that $\boxed{L = L(M_D)}$

NFA-to-DFA :-

1) - Create a graph G_D with vertex $\{q_0\}$ as initial state.

2) - Repeat the following steps until no more edge are missing.

a) - Take any vertex $\{q_i, q_j \cdots q_K\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\hat{\delta}(\{q_i, q_j \cdots q_K\}, a)$

$$\hat{\delta}(\{q_i, q_j, \cdots q_K\}, a) = \hat{\delta}(q_i, a) \cup \hat{\delta}(q_j, a) \cup \cdots \cup \hat{\delta}(q_K, a)$$

$$= \{q_l, q_m, \cdots q_n\}$$

b) - Create a vertex for G_D labeled $\{q_l, q_m, \cdots q_n\}$ if it does not already exist.

c) - Add an edge from $\{q_i, q_j, \cdots q_K\}$ to $\{q_l, q_m \cdots q_n\}$ and label it with a.

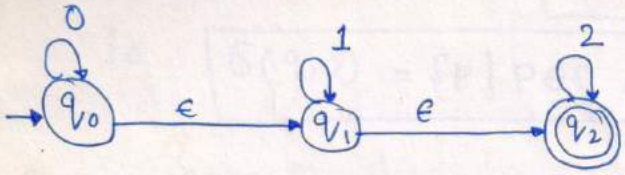
3) - Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.

4) - If M_N accepts ϵ, the vertex $\{q_0\}$ in G_D is also made a final vertex.

⇒ A nondeterministic algorithm that can make the best choice would be able to solve the problem without backtracking, but a deterministic one can simulate nondeterminism with some extra work.

Finite automata with ϵ-moves :-

Extend our model of the NFA to include transition on the empty input ϵ.



A NFA-with ϵ-moves is a quintuple $(Q, \Sigma, \delta, q_0, F)$ with all components as before, but δ, the transition function, maps

$$Q \times (\Sigma \cup \epsilon) \text{ to } 2^Q$$

⇒ The intention is that $\delta(q, a)$ will consist of states p such that there is a transition labeled a from q to p, where a is either ϵ or symbol in Σ.

⇒ Extend the transition function δ to a function $\hat{\delta}$ that maps

$$Q \times \Sigma^* \text{ to } 2^Q$$

, our expactation is that $\hat{\delta}(q, w)$ will be all states p such that one can go from q to p along a path labeled w, perhaps including edges labeled ϵ.

ϵ-closure :- We use ϵ-closure (q) to denote the set of all vertices p such that there is a path from q to p labeled ϵ.

ϵ-closure $(q_0) = \{q_0, q_1, q_2\}$

⇒ if P is a set of states. Then the ϵ-closure (P) is

$$\bigcup_{q \text{ in } P} \epsilon\text{-closure}(q)$$

Now $\hat{\delta}$ define as follows:-

1)- $\hat{\delta}(q,\epsilon) = \epsilon\text{-Closure}(q)$

2)- for w in Σ^* and $a \in \Sigma$,

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

where $A = \{p \mid \text{for some } r \text{ in } \hat{\delta}(q,w),\ p \text{ is in } \delta(r,a)\}$

3. $\hat{\delta}(q,a) = \delta(\hat{\delta}(q,\epsilon), a)$
$= \epsilon\text{-closure}\{\delta(\epsilon\text{-closure}(q), a)\}$

It is convenient to extend δ and $\hat{\delta}$ to sets of states by

3)- $\delta(R,a) = \bigcup_{q \text{ in } R} \delta(q,a)$

4- $\hat{\delta}(R,w) = \bigcup_{q \text{ in } R} \hat{\delta}(q,w)$

$\Rightarrow$ The language accepted by $M = (Q, \Sigma, \delta, q_0, F)$ is

$L(M) = \{w \mid \hat{\delta}(q_0,w)$ contains a state in $F\}$.

$\hat{\delta}(q_0,0) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0,\epsilon), 0))$ $\rightarrow \epsilon\text{-closure}(q_0)$

$\Rightarrow \epsilon\text{-closure}(\delta(\{q_0,q_1,q_2\}, 0))$

$\Rightarrow \epsilon\text{-closure}(\delta(q_0,0) \cup \delta(q_1,0) \cup \delta(q_2,0))$

$\Rightarrow \epsilon\text{-closure}(\{q_0\} \cup \phi \cup \phi)$

$\Rightarrow \epsilon\text{-closure}(\{q_0\})$

$\Rightarrow \{q_0,q_1,q_2\}$ final state (accepted)

$\hat{\delta}(q_0,01) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0,0), 1))$

$\Rightarrow \epsilon\text{-closure}(\delta(\{q_0,q_1,q_2\}, 1))$

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

$\Rightarrow \{q_1,q_2\}$

accepted.

Equivalence of NFA's with and without ε-moves :-

Theorem :- If L is accepted by an NFA with ε-transitions, then L is accepted by an NFA without ε-transitions.

Proof :- Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA with ε-transitions then construct $M' = (Q, \Sigma, \delta', q_0, F')$ where

$$F' = \begin{cases} F \cup \{q_0\} & \text{if } \epsilon\text{-closure}(q_0) \text{ contains a state of } F, \\ F & \text{otherwise,} \end{cases}$$

and $\delta'(q, a)$ is $\hat{\delta}(q, a)$ for $q \in Q$ and $a \in \Sigma$.

We wish to show by induction on $|x|$ that

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

Basis :- $|x| = 1$. Then x is a symbol a and

$$\delta(q_0, a) = \hat{\delta}(q_0, a).$$

Induction :- $|x| > 1$. Let $x = wa$ for symbol a in Σ. Then

$$\delta'(q_0, wa) = \delta'(\delta'(q_0, w), a)$$

By the inductive hypothesis, $\delta'(q_0, w) = \hat{\delta}(q_0, w)$.

Let $\hat{\delta}(q_0, w) = P$. We must show that

$$\delta'(P, a) = \hat{\delta}(q_0, wa). \text{ But}$$

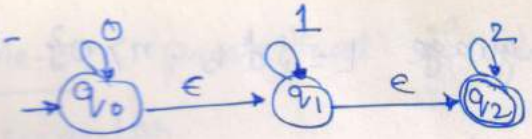
$$\delta'(P, a) = \bigcup_{q \text{ in } P} \delta'(q, a) = \bigcup_{q \text{ in } P} \hat{\delta}(q, a)$$

$$\bigcup_{q \text{ in } P} \hat{\delta}(q, a) = \hat{\delta}(q_0, wa)$$

$$\Rightarrow \boxed{\delta'(q_0, wa) = \hat{\delta}(q_0, wa)}$$

proved.

ex:-



procedure :- NFA with ϵ-moves to without ϵ-moves :-

1)- Compute ϵ-closure(q) $\forall$ $q \in Q$. If ϵ-closure(q) contains a state from F(finalstate) then make it final state.

2)- compute $\delta(q,a)$ $\forall$ $q \in Q, a \in \Sigma$ such that

$$\delta(q,a) = \hat{\delta}(q,a) \Rightarrow \epsilon\text{-closure}\left(\delta(\hat{\delta}(q,\epsilon),a)\right)$$

$$\Rightarrow \{q_i, q_j \cdots q_k\}$$

3)- Show the transition from q to q_i, q_j & q_k labeled a.

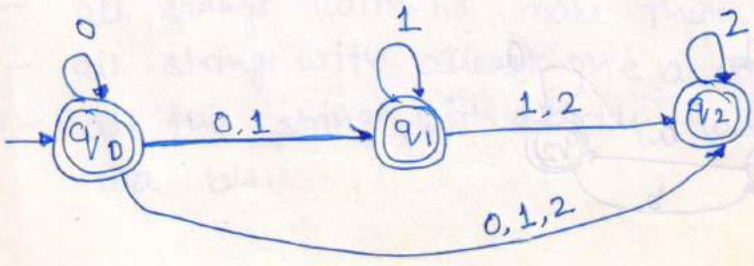
4)- Repeat step 2 & 3 until all transition completed.

solution:-

ϵ-closure(q_0) = $\{q_0, q_1, q_2\}$

ϵ-closure(q_1) = $\{q_1, q_2\}$

ϵ-closure(q_2) = $\{q_2\}$

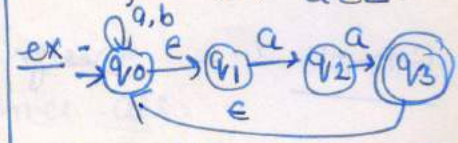


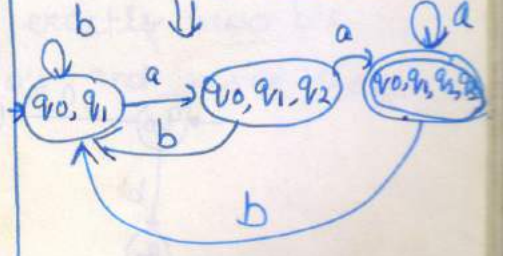
CONVERSION FROM NFA WITH ϵ-MOVES INTO DFA.

1) $q_0' = \epsilon$-closure(q_0) and

2) $\delta'(P,a) = \epsilon$-closure$(\delta(P,a))$ where $P = \{q_i, q_j \cdots q_k\}$ for all $a \in \Sigma$.

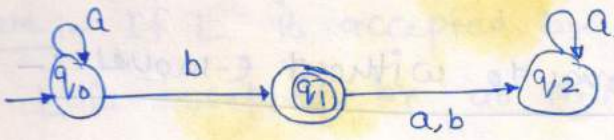
ex -



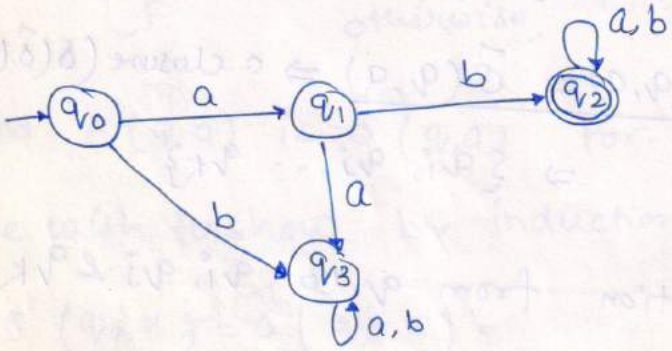


DFA

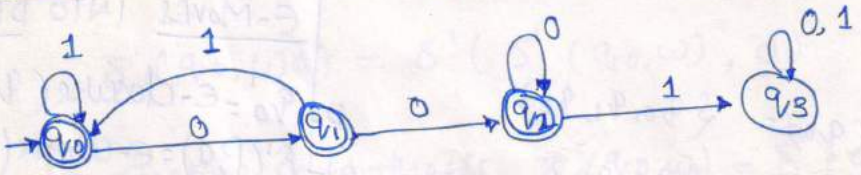
Example-1 :- Design DFA for language $L = \{a^n b : n \geq 0\}$



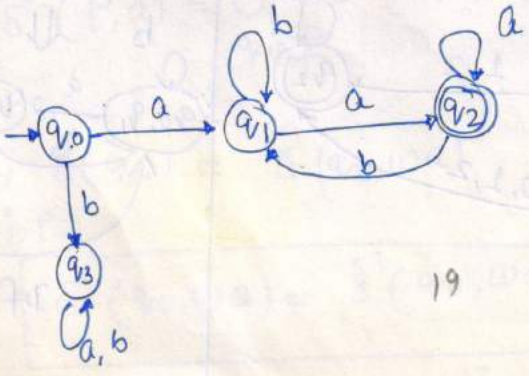
Example-2 :- find a DFA that recognise the set of all strings on $\Sigma = \{a, b\}$ starting with the prefix ab.



Example-3 :- Find a DFA that accepts all the strings on $\{0, 1\}$, except those containing the substring 001.



Example-4:- Design a DFA for the language $L = \{awa : w \in \{a, b\}^*\}$



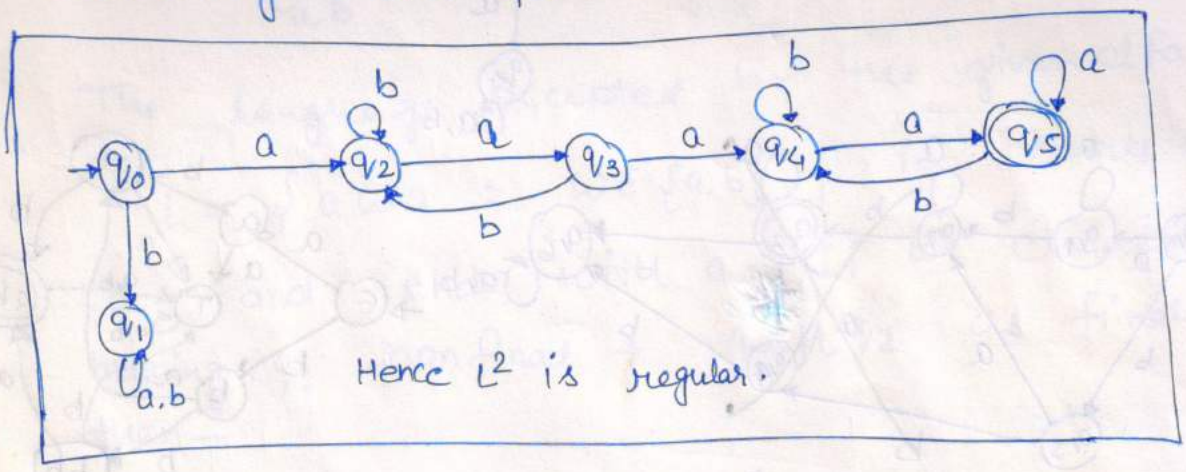
Example-5:- Prove that if L is regular then L^2 is also regular.

Solution:- As we know that if a language is regular then there exists a DFA M, such that $L = L(M)$.

$L^2 = L \cdot L$

for example. $L = \{awa : w \in \{a,b\}^*\}$

As we have saw in example4 language $L = \{awa : w \in \{a,b\}^*\}$ is regular. We have show that L^2 is regular by constructing a DFA for L^2.



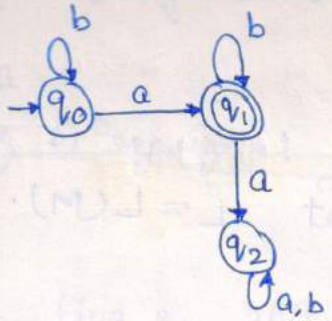
Hence L^2 is regular.

Example-6:- For $\Sigma = \{a,b\}$, construct DFA's that accept the sets consisting of —

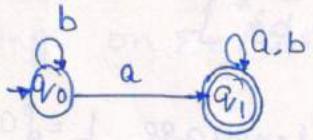
a)- all strings with exactly one a.
b)- all strings with at least one a.
c)- all strings with no more than three a's.
d)- all strings with atleast one a and exactly two b's.
e)- all the strings with exactly two a's and more than two b's.

Solution :-

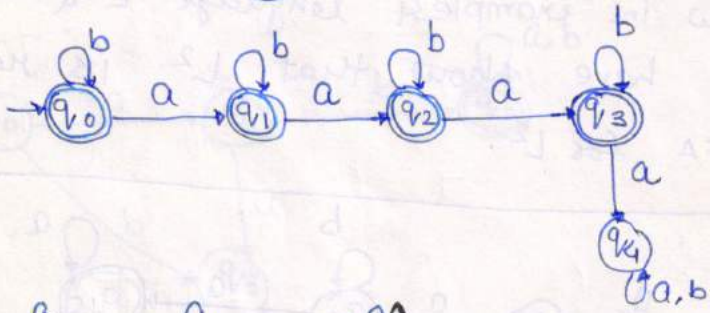
a)-



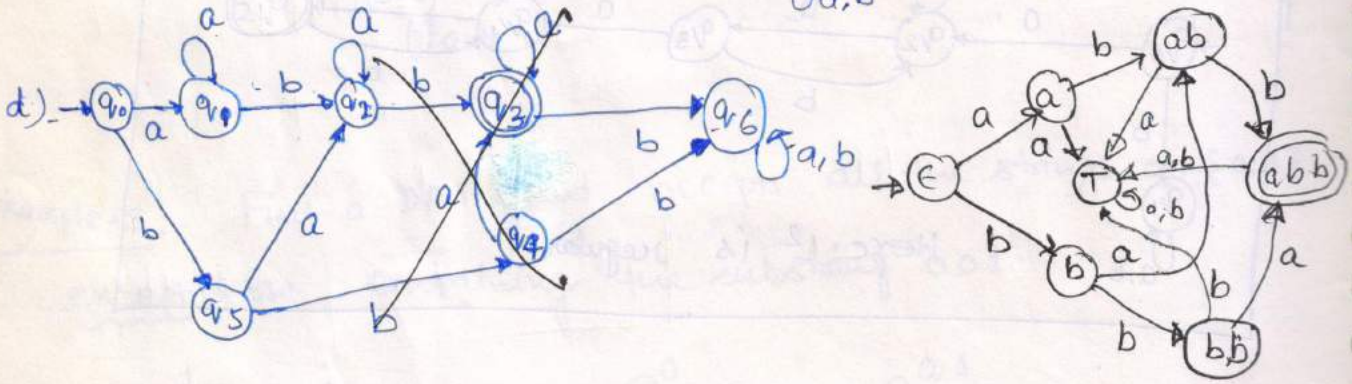
b)-



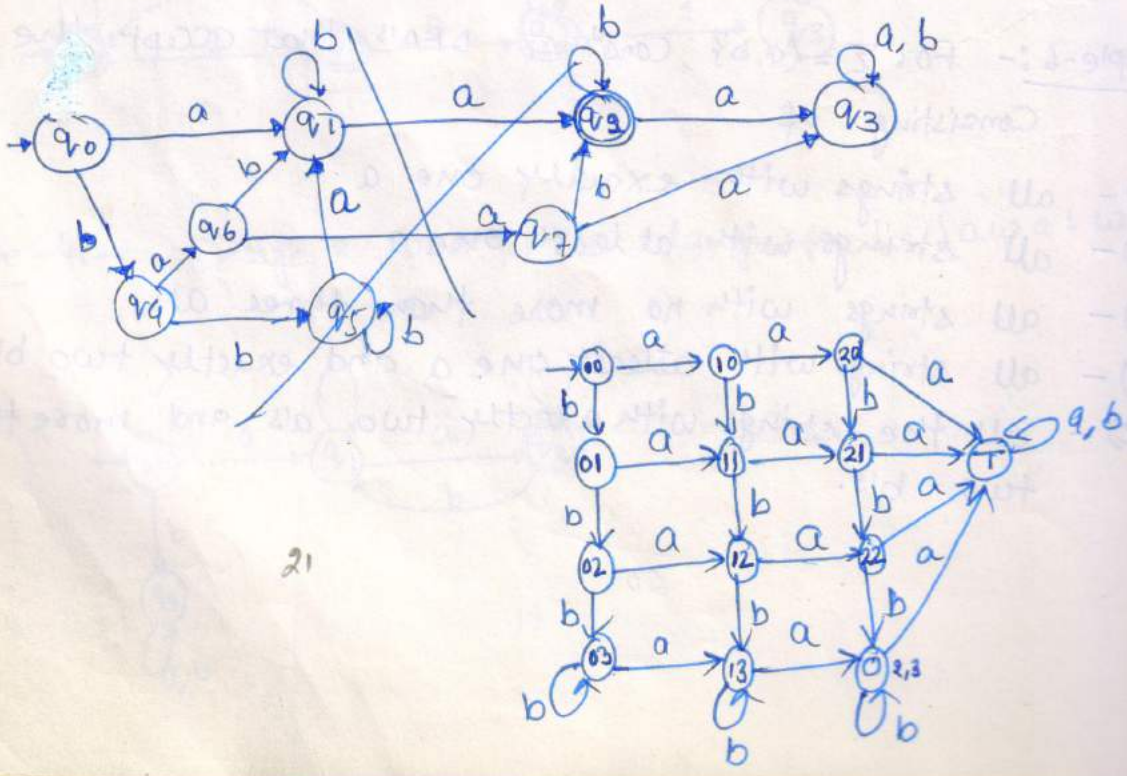
c)-



d)-

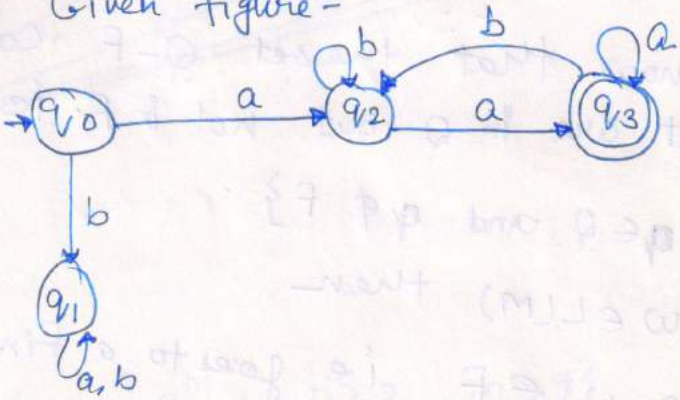


e)-

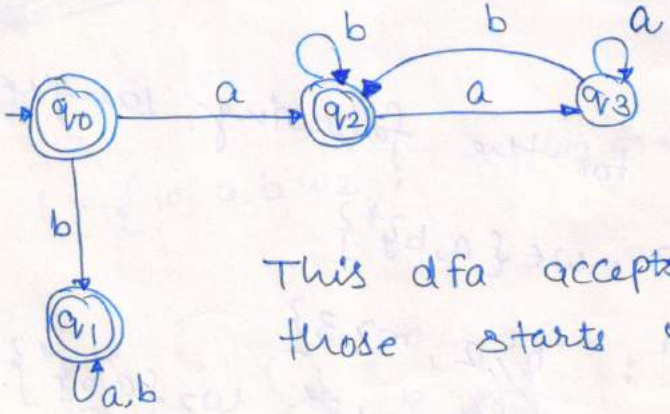


Question :- Show that if we change the following figure - making q_3 a nonfinal state and making q_0, q_1, q_2 Final states, the resulting dfa accepts $\bar{L}$.

Solution :- Given figure -



The language accepted by the given dfa is - $L = \{awa : w \in \{a,b\}^*\}$. ie starts with a and ends with a. If we change by making q_3 nonfinal & q_0, q_1, q_2 as final state then -



This dfa accepts all strings excepts those starts with a and ends with a.

Hence the language accepted by the above dfa is $\bar{L}$.

Question:- Show that if $M = (Q, \Sigma, \delta, q_0, F)$ and $\bar{M} = (Q, \Sigma, \delta, q_0, Q-F)$ are two dfas, then $\overline{L(M)} = L(\bar{M})$.

proof:- As we know that the set $Q-F$ contains all states that are in Q but not in F. ie

$Q-F = \{q : q \in Q \text{ and } q \notin F\}$.

if a string $w \in L(M)$ then

$\hat{\delta}(q_0, w) \in F$ ie goes to a final state.

Then $\overline{L(M)} = \{x : \hat{\delta}(q_0, x) \in Q-F\}$

ie if a string belongs to the set $\overline{L(M)}$ then it is accepted by a dfa $\bar{M}$.

Hence $\boxed{\overline{L(M)} = L(\bar{M})}$.

Question:- Give dfas for the following languages -

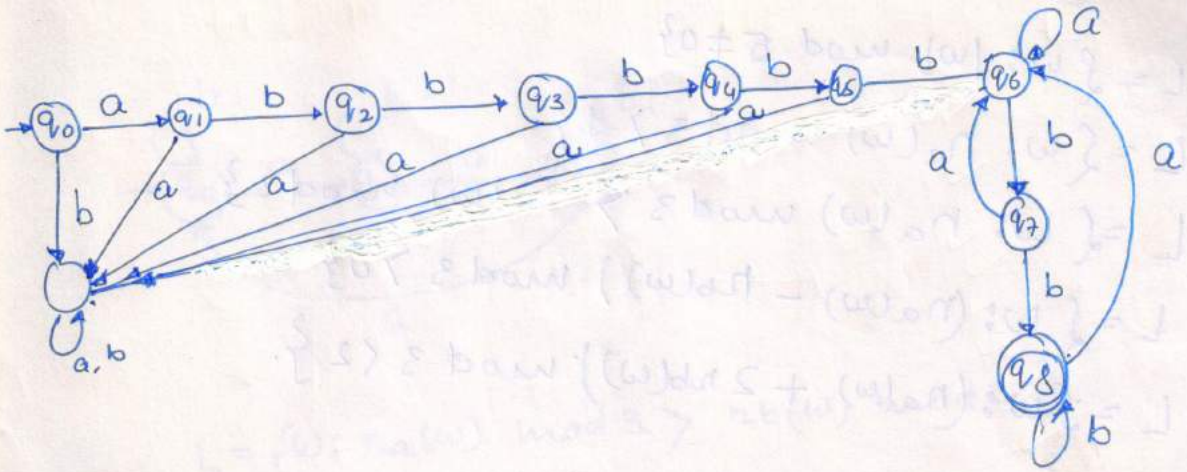
a) - $L = \{ab^5 w b^2 : w \in \{a,b\}^*\}$

b) - $L = \{ab^n a^m : n \geq 2, m \geq 3\}$

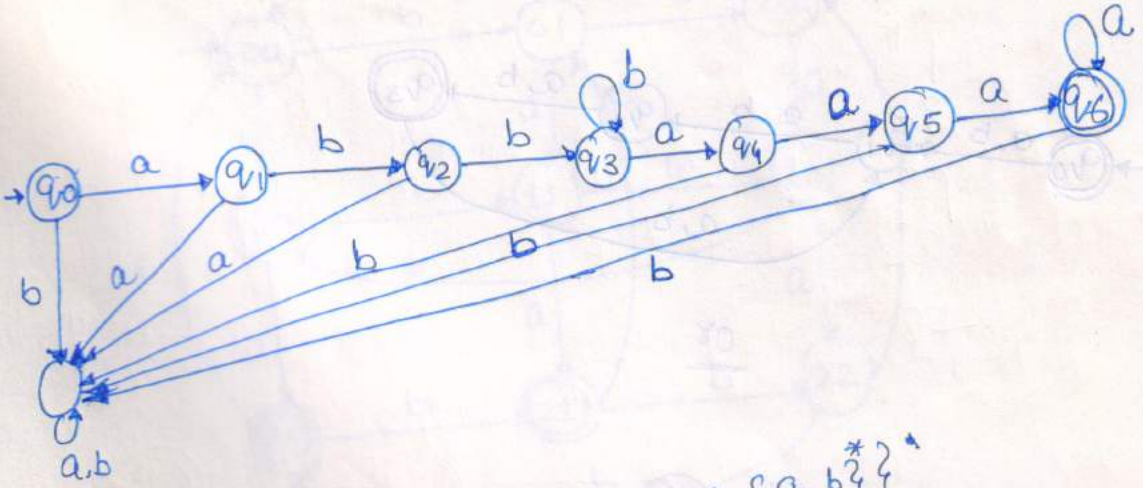
c - $L = \{w_1 ab w_2 : w_1 \in \{a,b\}^*, w_2 \{a,b\}^*\}$.

Solution :- $L = \{ ab^5 w b^2 : w \in \Sigma^* \}$.

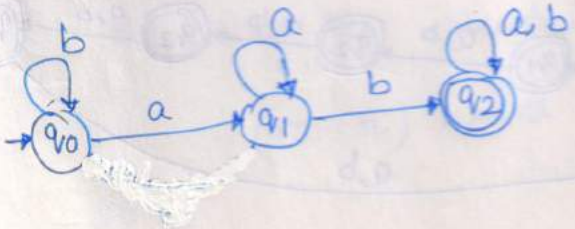
(a) -



(b) - $L = \{ a b^n a^m : n \geq 2, m \geq 3 \}$

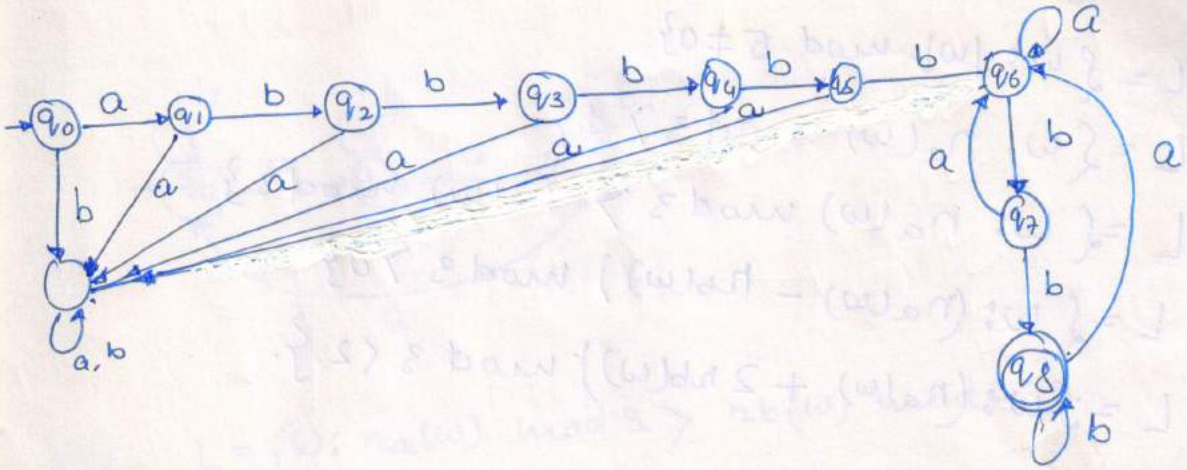


(c) - $L = \{ w_1 ab w_2 : w_1, w_2 \in \{a, b\}^* \}$

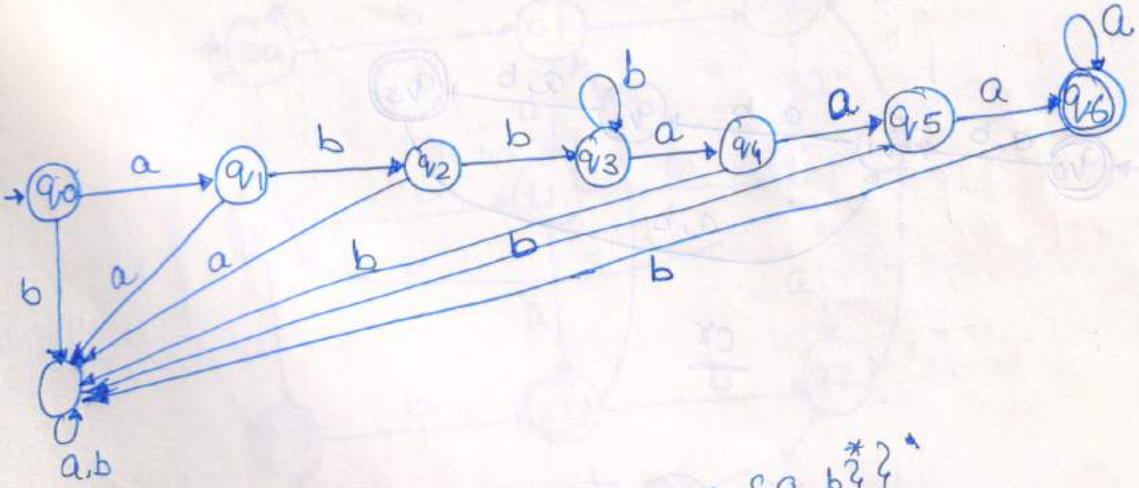


Solution :- $L = \{ab^5 w b^2 : w \in \Sigma^*\}$.

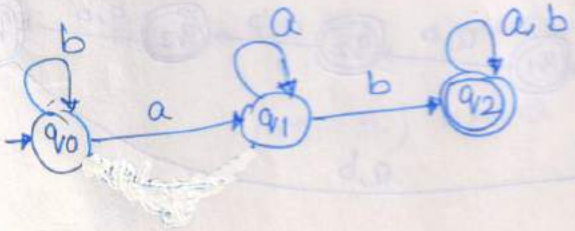
(a) -



(b) - $L = \{a b^n a^m : n \geq 2, m \geq 3\}$



(c) - $L = \{w_1 ab w_2 : w_1, w_2 \in \{a, b\}^*\}$



Question:- Find dfas for the following languages over $\Sigma = \{a, b\}$

(a) - $L = \{w : |w| \bmod 3 = 0\}$

(b) - $L = \{w : |w| \bmod 5 \neq 0\}$

(c) - $L = \{w : n_a(w) \bmod 3 > 1\}$

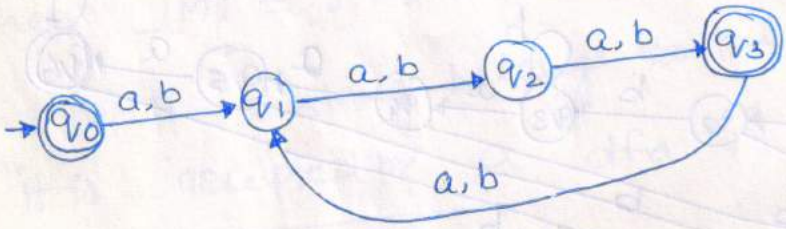
(d) - $L = \{w : n_a(w) \bmod 3 > n_b(w) \bmod 3\}$

(e) - $L = \{w : (n_a(w) - n_b(w)) \bmod 3 > 0\}$

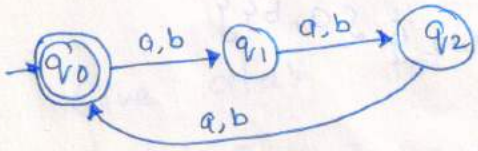
(f) - $L = \{w : (n_a(w) + 2n_b(w)) \cdot \bmod 3 < 2\}$.

Solution:-

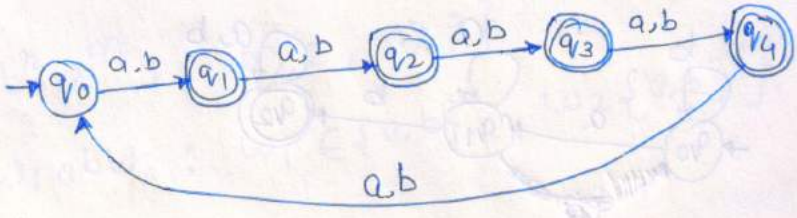
(a) -



OR

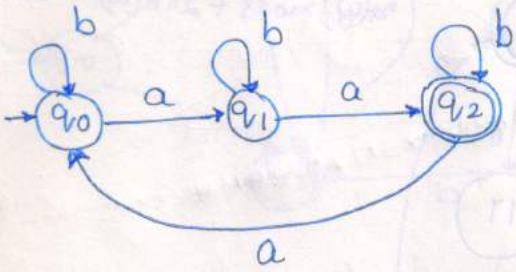


(b) -

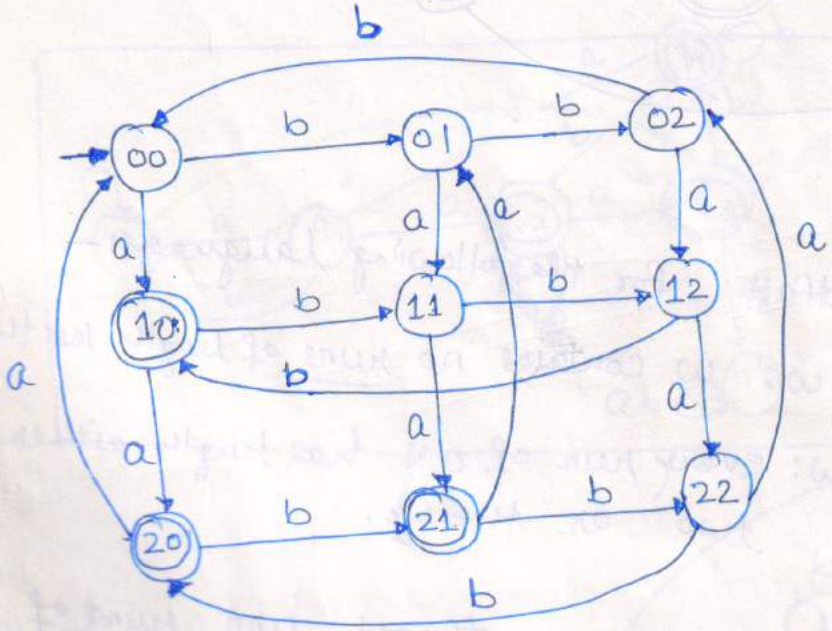


(c) - $L = \{w : n_a(w) \bmod 3 > 1\}$

As we know any no. [redacted] $n \bmod 3 > 1$ are 2.



(d) - $L = \{w : n_a(w) \bmod 3 > n_b(w) \bmod 3\}$



(e) - $L = \{w : (n_a(w) - n_b(w)) \bmod 3 > 0\}$

According to modular arithmatic -

$(m-n) \bmod P = m \bmod p - n \bmod p$

So

$(n_a(w) - n_b w) \bmod 3 \Rightarrow n_a(w) \bmod 3 - n_b(w) \bmod 3$

$\Rightarrow n_a(w) \bmod 3 > n_b(w) \bmod 3$

26

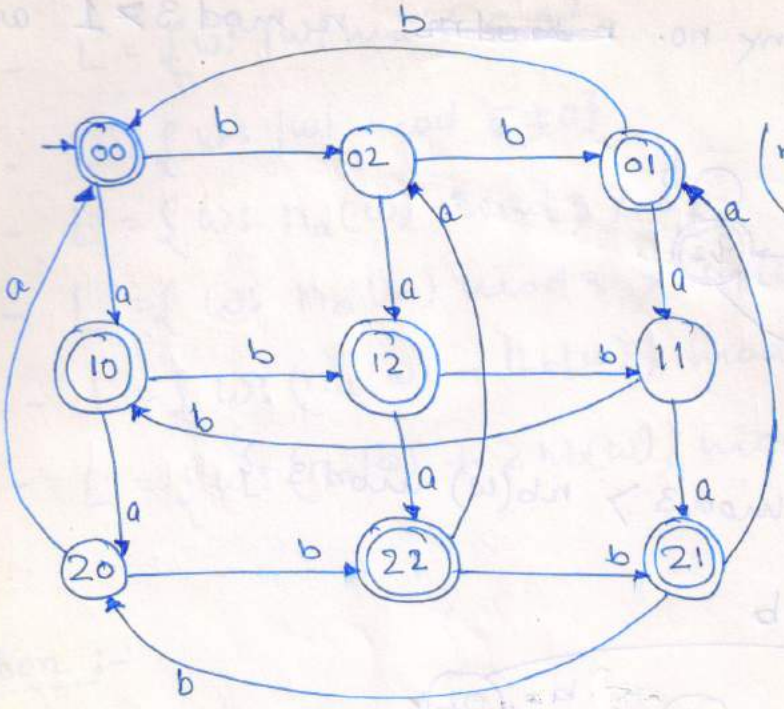
So the dfa ~~for~~ this language is like above dfa.

(f) - $L = \{w : (n_a(w) + 2n_b(w)) \bmod 3 < 2\}$

$n_a(w) \bmod 3 + 2 n_b(w) \bmod 3 < 2$

$\Downarrow$

$(n_a(w) \bmod 3 + 2 n_b(w) \bmod 3) \bmod 3 < 2$



Question:- find dfas for the following languages—

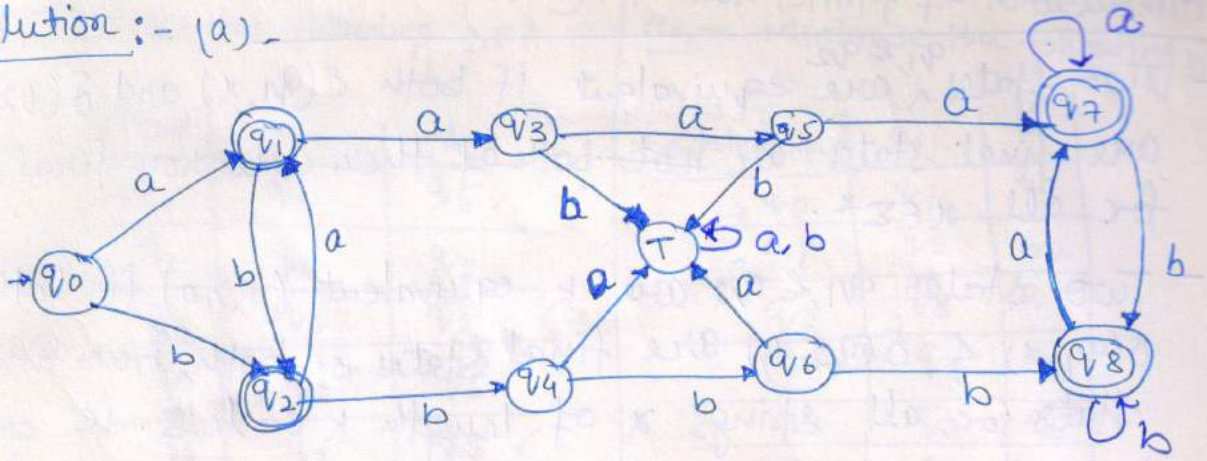
(a) - $L = \{w :$ w contains no runs of length less than four$\}$.

(b) - $L = \{w :$ every run of a's had length either two or three$\}$.

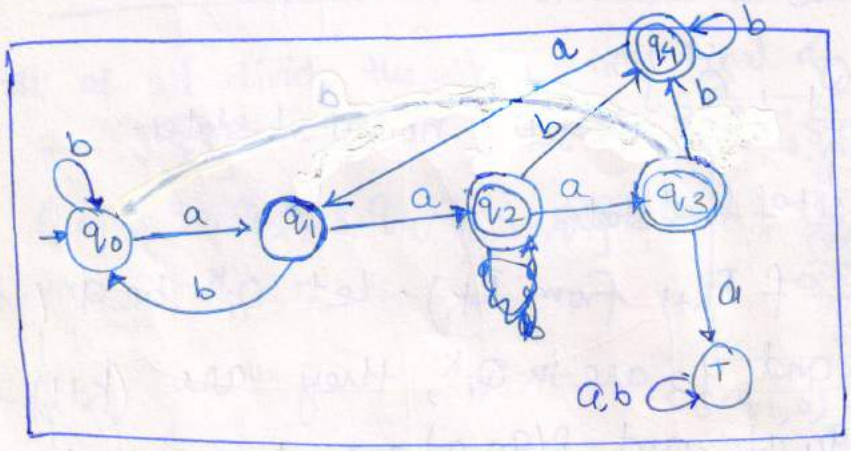
(c) - $L = \{w :$ there are atmost two runs of a's of length three$\}$.

(d) - $L = \{w :$ there are exactly two runs of a's of of length 3$\}$.

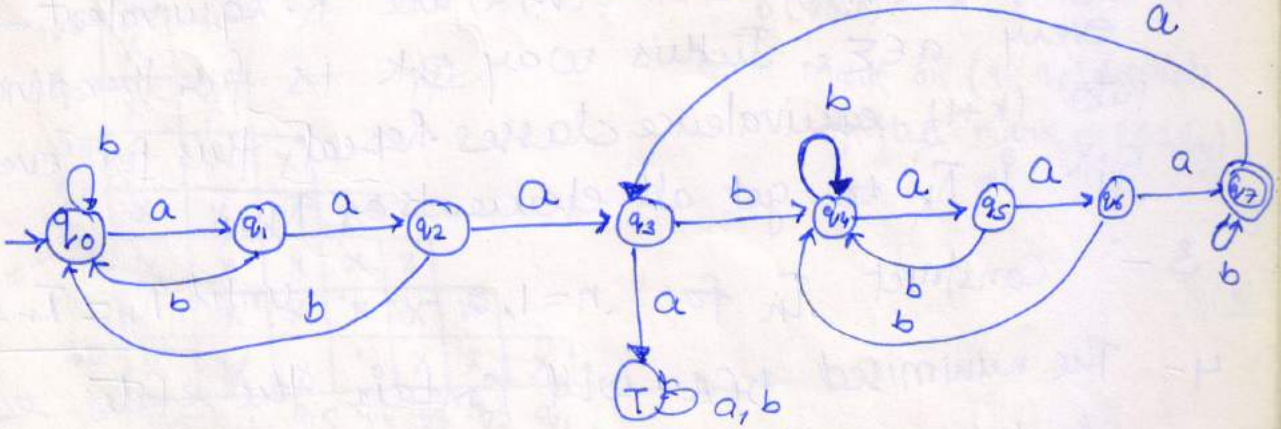
Solution :- (a) -



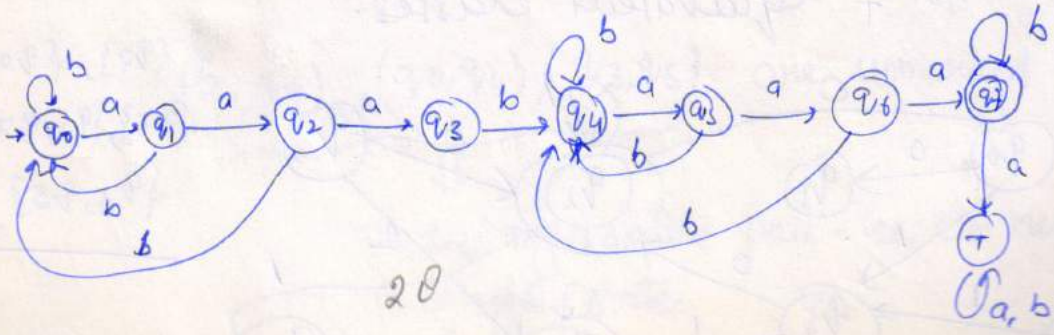
(b) -



(c) -



(d) -



Minimization of finite Automata (K.LP.MISHRA)

1- Two states q_1 & q_2 are equivalent if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or both of them are non-final states for all $x \in \Sigma^*$.

2- Two states q_1 & q_2 are k-equivalent ($k > 0$) if both $\delta(q_1, x)$ & $\delta(q_2, x)$ are final states or both are non-final states for all strings x of length k or less. ie any two final states are 0-equivalent & any two non final states are 0-equivalent.

Procedure - 1) - Construct π_0 ie 0-equivalent

$\pi_0 = \{Q_1^0, Q_2^0\}$

Q_1^0 → set of final states

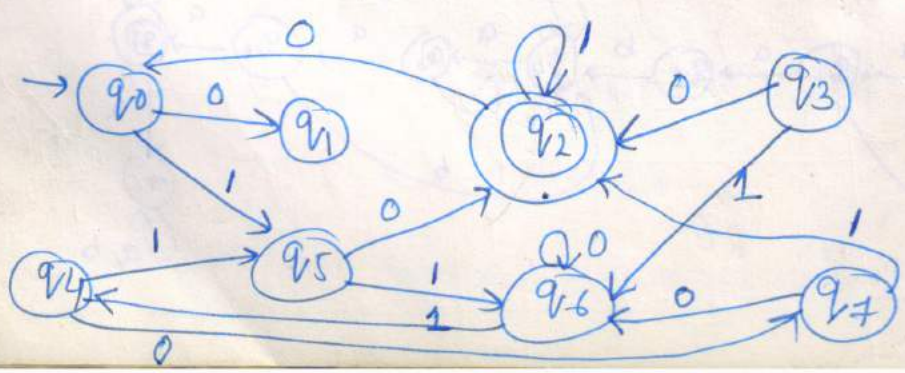
Q_2^0 → set of non-final states.

2- (Construction of π_{k+1} from π_k). Let Q_i^k be any subset in π_k. If q_1 and q_2 are in Q_i^k, they are (k+1) equivalent if (provided) $\delta(q_1, a)$ and $\delta(q_2, a)$ are k-equivalent for every $a \in \Sigma$. In this way Q_i^k is further divided into (k+1)-equivalence classes. Repeat this for every Q_i^k in π_k to get all elements of π_{k+1}.

3- Construct π_n for $n = 1, 2 - - -$ until $\pi_n = \pi_{n+1}$.

4- The minimized DFA will contain the states equal to the no. of equivalent classes.

Ex-



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

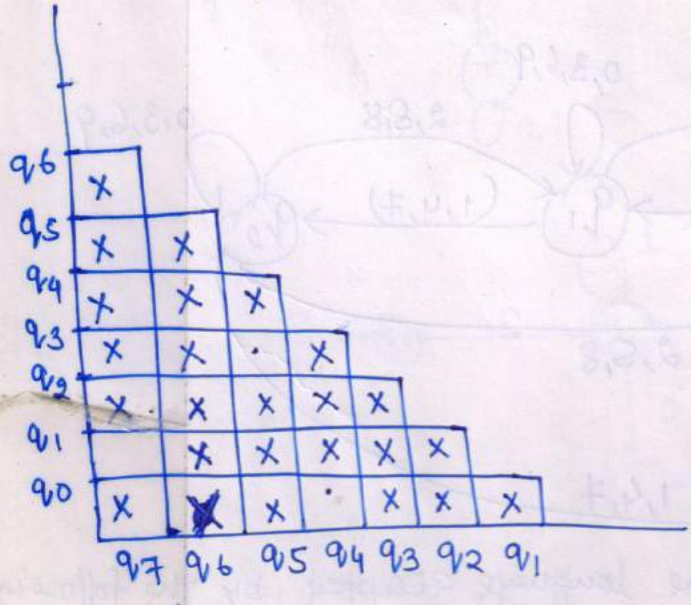
eg. Minimize the following DFA.

state/input	0	1
→q_0	q_1	q_5
q_1	q_6	q_2
q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

first of all divide the states into final & Non-final states

$F = \{q_2\}$

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



(q_1, q_7), (q_0, q_4), (q_3, q_5) are unmarked cells.

⇓

Indistinguishable pair. We can Merge them into a single state.

ex- Minimize the following DFA

state/input	a	b
→q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
(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

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

ex- P.S

P.S	a	b	
→q_0	q_1	q_2	0
q_1	q_4	q_3	(1,2)
q_2	q_4	q_3	(3,4)
(q_3)	q_5	q_6	(5,7)
(q_4)	q_7	q_6	6
q_5	q_3	q_6	
q_6	q_6	q_6	
q_7	q_4	q_6	

$\delta(q_1, 0) = q_6$ $\delta(q_1, 1) = q_2$

$\delta(q_6, 0) = q_6$ $\delta(q_6, 1) = q_4$

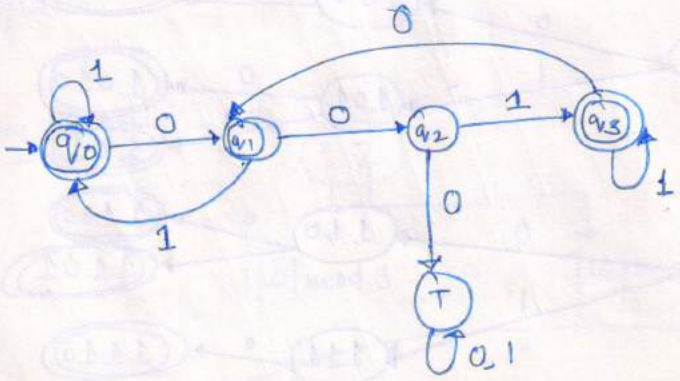
Mark on (q_1, q_6) depends on the mark on (q_2, q_4)

Question:- Construct a dfa for the following ~~lan~~—

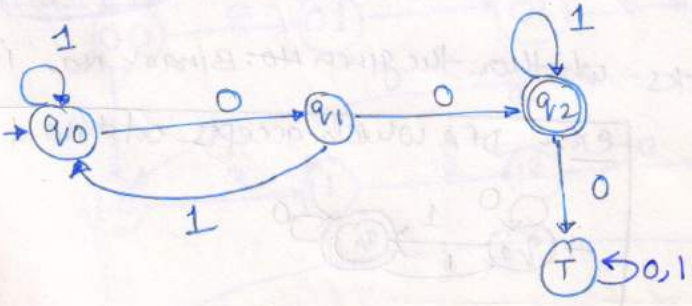
(a)- Every 00 is followed immediately by 1.
(b)- All string containing 00 but not 000.
(c)- The leftmost symbol differs from the rightmost one.
(d)- Every substring of four symbols has atmost two 0's.
(e)- All strings of length five or more in which the forth symbol from the right end is different from the leftmost symbol.

Solution:-

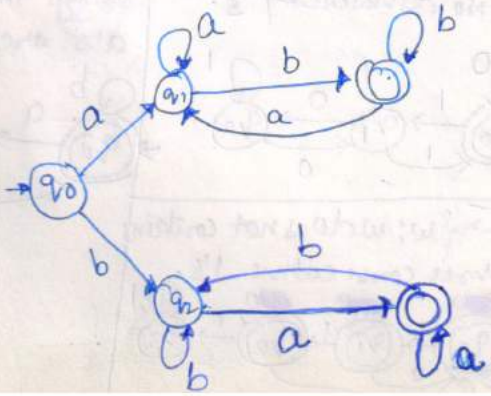
(a) -



(b)-



(c)-



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

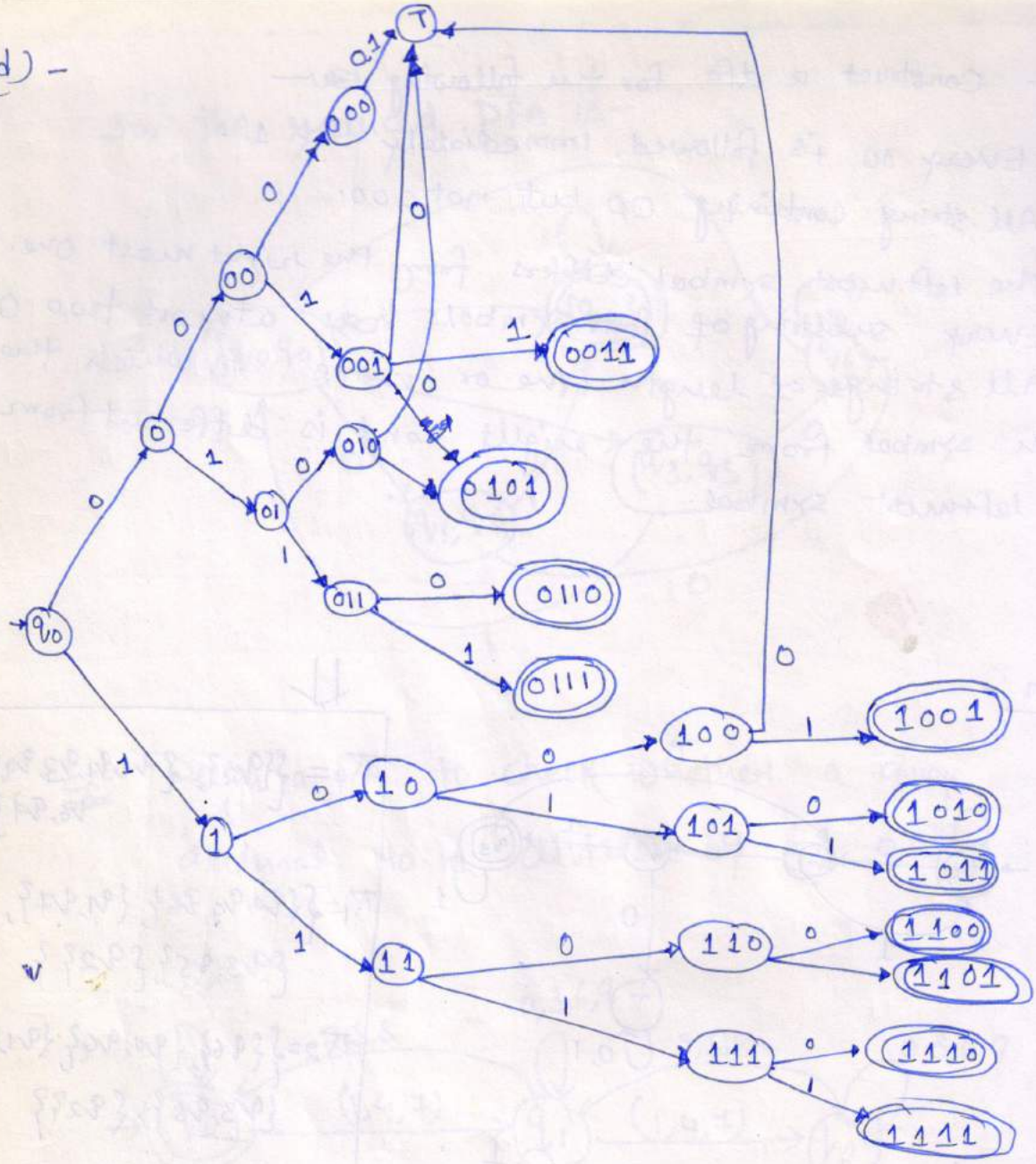
$\pi_1 = \{\{q_0, q_5, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}, \{q_2\}\}$

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

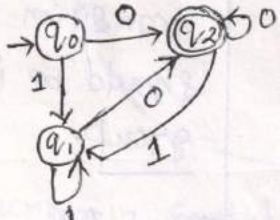
* for each $a \in \Sigma$ transition of $q_i \& q_j$ should be in the same group.

$\pi_3 = \pi_2$

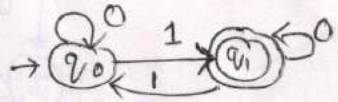
(d) –



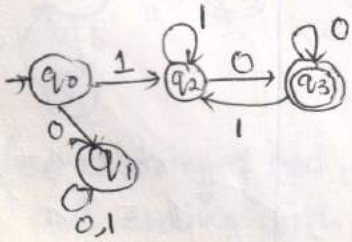
(Ex) – DFA, which ~~accepts~~ checks whether the given ~~NO.~~ Binary NO. is even or not?



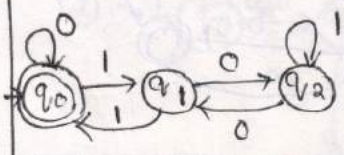
ex:- DFA which accepts 'odd NO. of 1's.



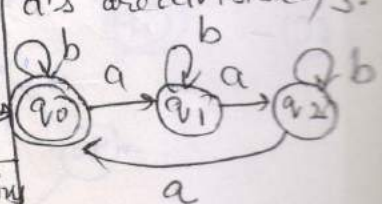
ex- $L = \{ 1w0 : w \in \{0,1\}^* \}$



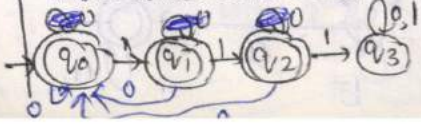
ex: DFA to check a binary NO. divisible by 3.



ex- DFA to accepts strings in which a NO. of a's are divisible by 3.

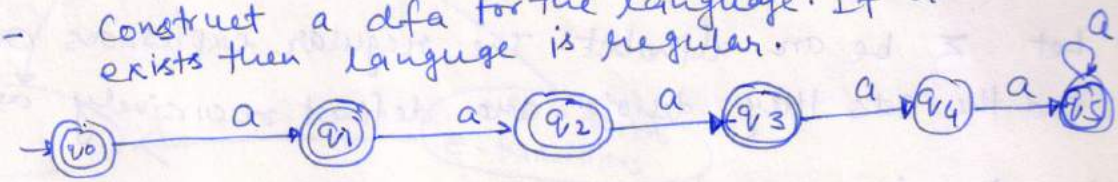


ex: $L = \{ w : w$ does not contain three consecutive 1's. $\}$



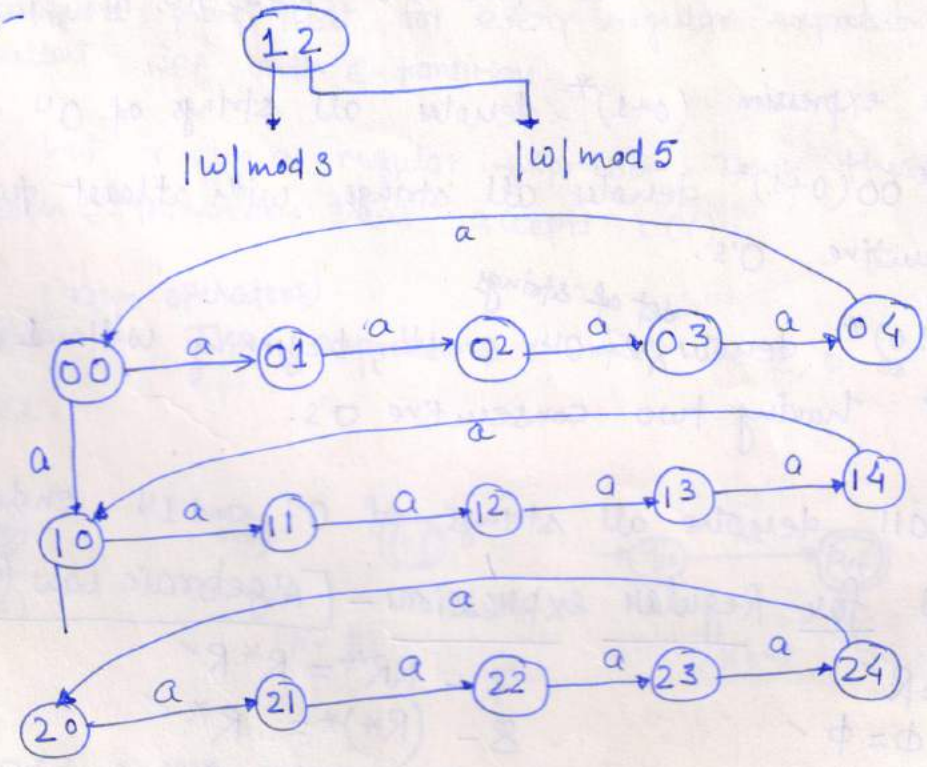
Question:- Show that the language $L = \{a^n : n \geq 0, n \neq 4\}$ is regular.

Solution:- Construct a dfa for the language. If dfa exists then language is regular.



Question:- Show that the language $L = \{a^n : n$ is a multiple of three, but not a multiple of 5$\}$ is regular.

Solution:-



REGULAR EXPRESSION

The language accepted by finite automata are easily described by simple expressions called **regular expression**.

Let Σ be an alphabet. The regular expressions over Σ and the sets they denote are defined recursively as follows:-

1)- ϕ is a regular expression and denote the empty set
2)- ϵ is a regular expression and denote the set $\{\epsilon\}$
3)- For each a in Σ, a is a regular expression and denote the set $\{a\}$.
4)- If r and s are regular expressions denoting the set R & S respectively, then $(r+s)$, (rs), (r^*) are regular expressions that denote the sets $R \cup S$, RS, R^*, respectively.

exp- The expression $(0+1)^*$ denotes all strings of 0's and 1's.

⇒ $(0+1)^* 00 (0+1)^*$ denotes all strings with atleast two consecutive 0's.

⇒ $(1+10)^*$ denotes set of strings of 0's & 1's, beginning with 1, and not having two consecutive 0's.

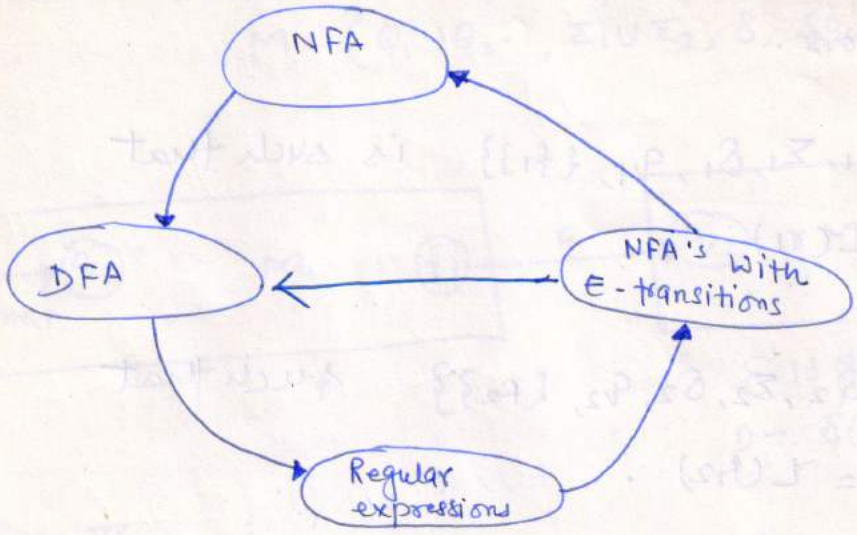
⇒ $(0+1)^*011$ denotes all strings of 0's and 1's ending in 011.

Identities for Regular expressions - [Algebraic Law for R.E]

1- $\phi + R = R$ ✓
2- $\phi R = R\phi = \phi$ ✓
3- $\epsilon R = R \cdot \epsilon = R$ ✓
4- $\epsilon^* = \epsilon$ and $\phi^* = \epsilon$ ✓
5- $R + R = R$ ✓
6- $R^* R^* = R^*$ ✓
7- $RR^* = R^* R$ ✓
8- $(R^*)^* = R^*$ ✓
9- $\epsilon + RR^* = R^* = \epsilon + R^* R$ ✓
10- $(PQ)^* P = P(QP)^*$ ✓
11- $(P+Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$ ✓
12- $(P+Q)R = PR + QR$ and ✓
$R(P+Q) = RP + RQ$

* Two regular expressions P and Q are equivalent if P & Q represent the same set of strings.

EQUIVALENCE of FA's and REGULAR EXPRESSIONS



⇒ The languages accepted by finite automata are precisely the language denoted by regular expressions.

We proceed to prove that for every regular expression there is an equivalent NFA with ε-transitions.

Theorem:- Let r be a regular expression. Then there exists an NFA with ε-transitions that accepts $L(r)$.

Proof:- (zero operators)

(Basis) – The expression r must be ϵ, ϕ or a for some $a \in \Sigma$.

Induction :- (one or more operators) :-

Assume that the theorem is true for regular expressions with fewer than i operators, $i \geq 1$.

Case I :- $r = r_1 + r_2$. Both r_1 & r_2 have fewer than i operators.

Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ is such that

$L(M_1) = L(r_1)$

&

$M_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$ such that

$L(M_2) = L(r_2)$.

$Q_1 \cap Q_2 = \phi$

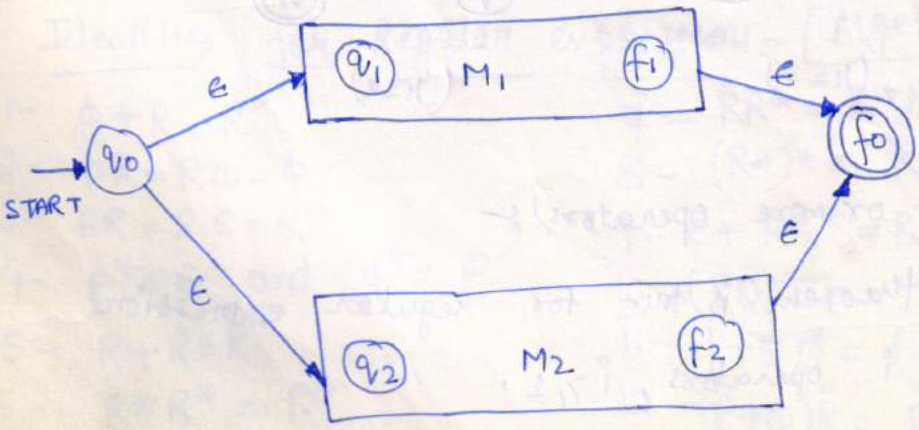
Let q_0 be a new initial state & f_0 a new final state. Construct $M = (Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma_1 \cup \Sigma_2, \delta, q_0, f_0)$ where δ is defined as -

(i) - $\delta(q_0, \epsilon) = \{q_1, q_2\}$

(ii) - $\delta(q, a) = \delta_1(q, a)$ for $q \in Q_1 - \{f_1\}$ & $a \in \Sigma_1 \cup \{\epsilon\}$

(iii) - $\delta(q, a) = \delta_2(q, a)$ for $q \in Q_2 - \{f_2\}$ & $a \in \Sigma_2 \cup \{\epsilon\}$

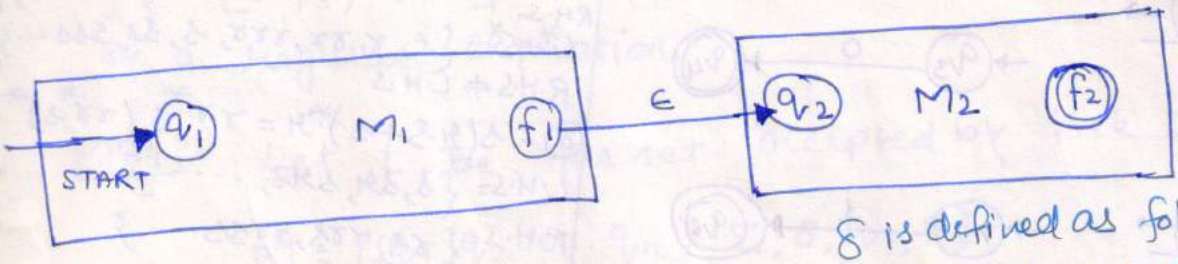
(iv) - $\delta(f_1, \epsilon) = \delta(f_2, \epsilon) = \{f_0\}$



Case-II

$r = r_1 r_2$

$M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, \{q_1\}, \{f_2\})$



δ is defined as follows :-

1) $\delta(q,a) = \delta_1(q,a)$ for $q \in Q_1 - \{f_1\}$, $a \in \Sigma_1 \cup \{\epsilon\}$

2) $\delta(q,a) = \delta_2(q,a)$ for $q \in Q_2$, $a \in \Sigma \cup \{\epsilon\}$

3) $\delta(f_1, \epsilon) = \{q_2\}$

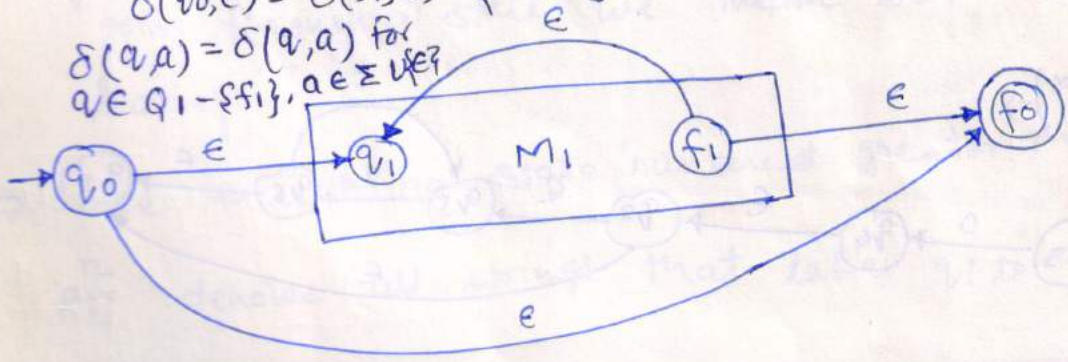
Case-III

$r = (r_1)^*$

Let $M_1 = (Q, \Sigma, \delta, q_1, f_1)$ be the FA such that $L(M_1) = L(r_1)$. Now construct a FA, $M = (Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\})$

$\delta(q_0, \epsilon) = \delta(f_1, \epsilon) = \{q_1, f_0\}$

$\delta(q,a) = \delta(q,a)$ for $q \in Q_1 - \{f_1\}$, $a \in \Sigma \cup \{\epsilon\}$



Example :- Construct an NFA for the regular expression $01^* + 1$.

$\Downarrow$

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

r_1 r_2^* r_3

Construct Regular Exp

Ex :- 1 - All strings ending with 00 $\Rightarrow (0+1)^*00$

2 - starts with 1 & ends with 0 $\Rightarrow 1(0+1)^*0$

3 - starts with a but not having consecutive b's

$(a + ab)^*$

4 - Begins & ends with 00 or 11

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

5 - 3rd character from right end is always a.

$R = (a+b)^* \underset{3^{rd}}{a} \underset{2^{nd}}{(a+b)} \underset{1^{st}}{(a+b)}$

6 - Exactly 2 b's.

$R = a^* b a^* b a^*$

7 - Every 0 is immediately followed by 11.

$(011+1)^*$

Ex. prove that

① $\epsilon + 1^*(011)^*(1^*(011)^*)^* = (1+011)^*$

$\epsilon + PP^* \Rightarrow P^* = (1+011)^*$

② prove or disprove $(r+s)^* = r^*+s^*$

LHS - $(r+s)^* = \{\epsilon, r, s, rr, ss, rs \cdots\}$

RHS - $r^*+s^* = \{\epsilon, r, rr, rrr, s, ss, sss \cdots\}$

R.H.S ≠ L.H.S

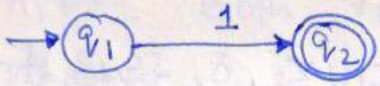
③ - $s(rs+s)^*r = rr^*s(rr^*s)^*$

L.H.S = $\{s, ssr, srs, \cdots\}$

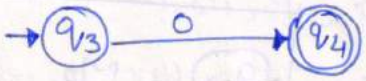
R.H.S = $\{rs, rrs, rrss \cdots\}$

R.H.S ≠ L.H.S

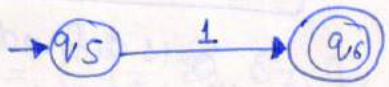
(r_3) →



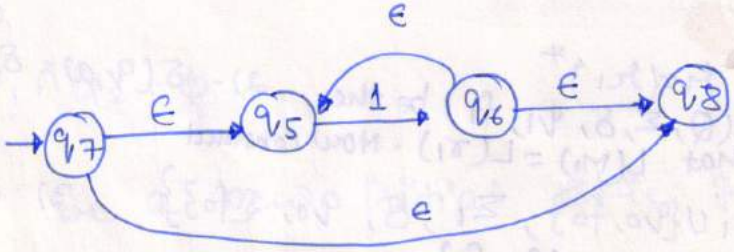
(r_1) →



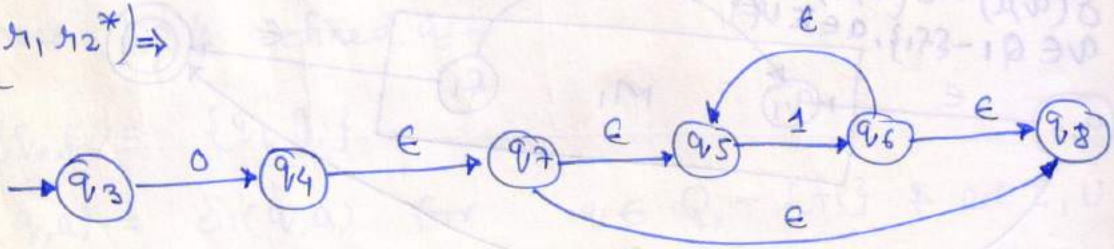
(r_2) →



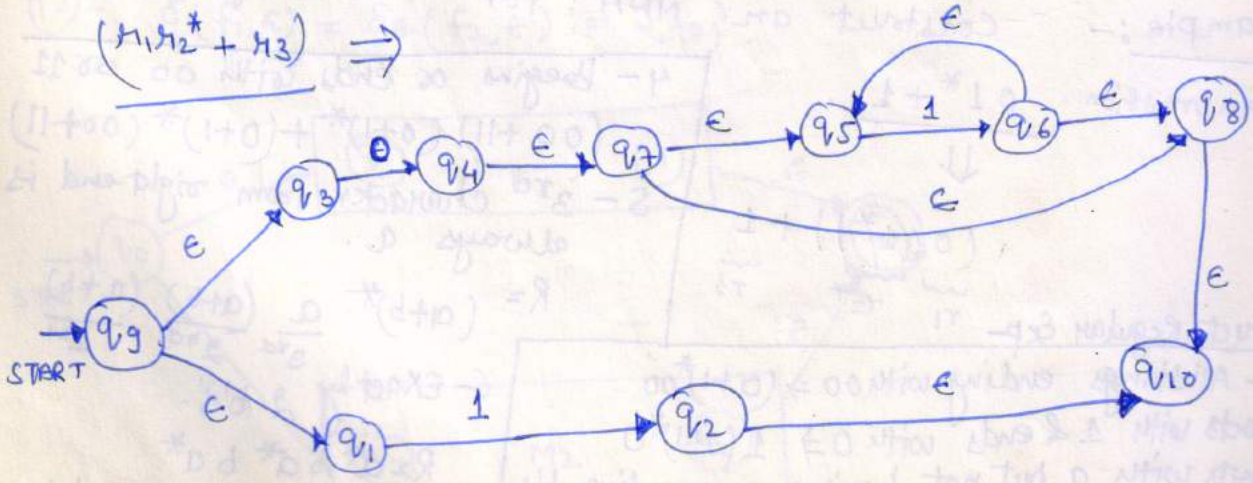
(r_2^*)



($r_1 r_2^*$) ⇒



($r_1 r_2^* + r_3$) ⇒



DFA to REGULAR EXPRESSION

(KLEEN'S THEOREM)

Theorem :-

If L is accepted by a DFA, then L is denoted by a regular expression.

proof :- Let L be the set accepted by the DFA

$$M = (\{q_1, q_2 \ldots q_n\}, \Sigma, \delta, q_1, F)$$

Let R_{ij}^{K} – is the set of all strings that take the finite automaton from state q_i to q_j without going through any state numbered higher than K.

"going through a state" we mean both entering & leaving.

⇒ Since there is no state numbered greater than n, R_{ij}^{n} denotes all strings that takes q_i to q_j.

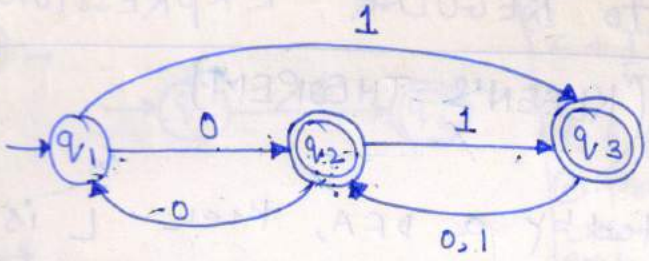
$$R_{ij}^{K} = R_{iK}^{K-1} \left(R_{KK}^{K-1}\right)^{*} R_{Kj}^{K-1} \cup R_{ij}^{K-1}$$

$$R_{ij}^{0} = \begin{cases} \{a \mid \delta(q_i, a) = q_j\} & \text{if } i \neq j \\ \{a \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\} & \text{if } i = j \end{cases}$$

So

$$L(M) = \bigcup_{q_j \text{ in } F} R_{1j}^{n}$$

Example :-



	K = 0	K = 1	K = 2
R_{11}^K	ϵ	ϵ	$(00)^*$
R_{12}^K	0	0	$0(00)^*$
R_{13}^K	1	1	0^*1
R_{21}^K	0	0	$0(00)^*$
R_{22}^K	ϵ	$00+\epsilon$	$(00)^*$
R_{23}^K	1	$1+01$	0^*1
R_{31}^K	ϕ	ϕ	$(0+1)(00)^*0$
R_{32}^K	0+1	0+1	$(0+1)(00)^*$
R_{33}^K	ϵ	ϵ	$\epsilon + (0+1)0^*1$

We need R_{12}^3 & R_{13}^3 because q_2 & q_3 are Final states.

$$R_{12}^3 = R_{13}^2 (R_{33}^2)^* R_{32}^2 \cup R_{12}^2$$

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

$$R_{13}^3 = 0^*1((0+1)0^*1)^*$$

MINIMISATION OF DFA

⇒ Any dfa defines a unique language, but the converse is not true. for a given language, there are many dfa's that accepts it.

Definition:-

Two states p & q are called indistinguishable if -

$\delta^*(p,\omega) \in F$ implies $\delta^*(q,\omega) \in F$

and

$\delta^*(p,\omega) \notin F$ implies $\delta^*(q,\omega) \notin F$, for all $\omega \in \Sigma^*$.

If on the other hand, there exists some string $\omega \in \Sigma^*$ such that

$\delta^*(q,\omega) \in F$ and $\delta^*(q,\omega) \notin F$, or vice-versa.

then the states q, and p are said to be distinguishable.

procedure:- mark.

1) - Remove all unreachable states.

2) - Consider all pairs of states (p,q). If $p \in F$ and $q \notin F$ or vice versa mark the pair (p,q) as distinguishable.

3) - Repeat the following step until no previously unmarked pairs are marked. for all pairs (p,q) and all $a \in \Sigma$, compute $\delta(p,a) = p_a$, $\delta(q,a) = q_a$. If the pair (p_a, q_a) is marked as distinguishable, mark (p,q) as distinguishable.

⇒ The procedure mark divides the states into equivalent classes.

Procedure – reduce :- Given a dfa $M = (Q, \Sigma, \delta, q_0, F)$ we construct a reduced dfa $\hat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q_0}, \hat{F_0})$.

1) – Use procedure mark to generate equivalent classes, say $\{q_i, q_j, \ldots q_k\}$.

2) – Create a state labeled $\{i, j, \ldots k\}$ for $\hat{M}$.

3) – for each transition rule of the

$$\delta(q_r, a) = q_p$$

if $q_r \in \{q_i, q_j, \ldots q_k\}$ and $q_p \in \{q_l, q_m \ldots q_n\}$

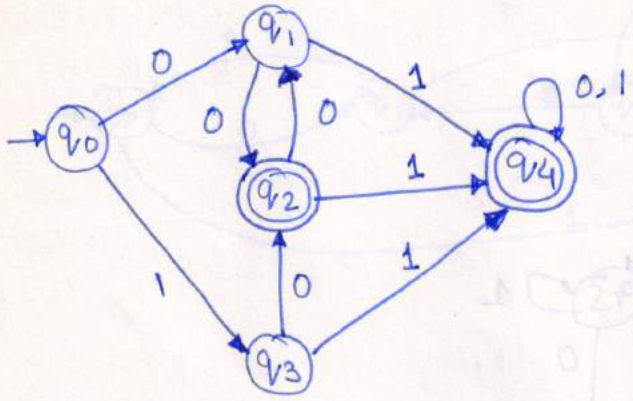
then add to $\hat{\delta}$ a rule –

$$\hat{\delta}(ij\ldots k, a) = lm\ldots n$$

4) – The initial state $\hat{q_0}$ is that of $\hat{M}$ whose label includes the 0.

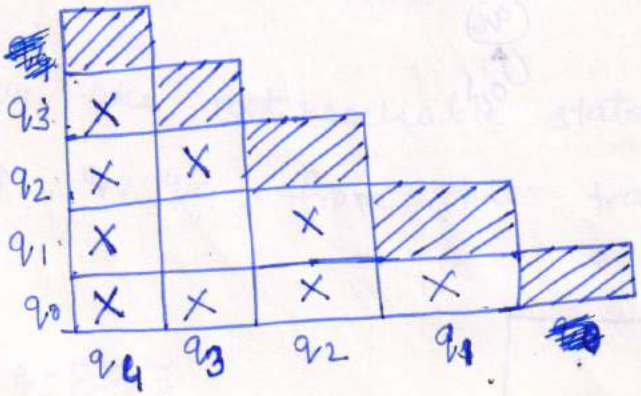
5 – $\hat{F}$ is the set of all the states whose label contains i such that $q_i \in F$.

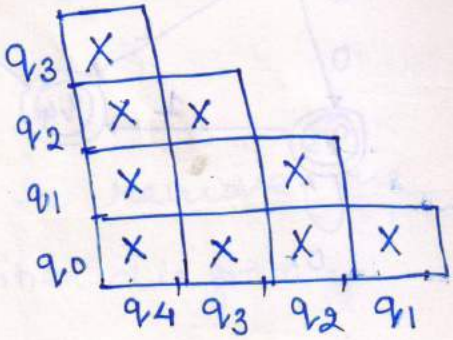
Example:-



$F = \{q_2, q_4\}$

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





~~$\delta(q_1, q_3)$~~

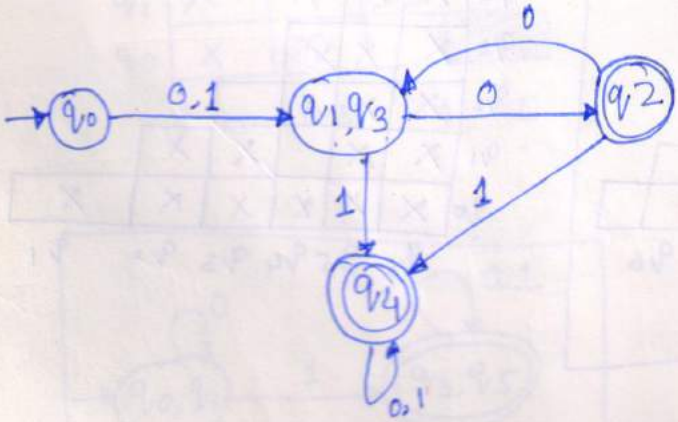
(q_1, q_3) { $\delta(q_1, 0) = q_2$ $\delta(q_1, 1) = q_4$
$\delta(q_3, 0) = q_2$ $\delta(q_3, 1) = q_4$

indistinguishable (q_1, q_3)

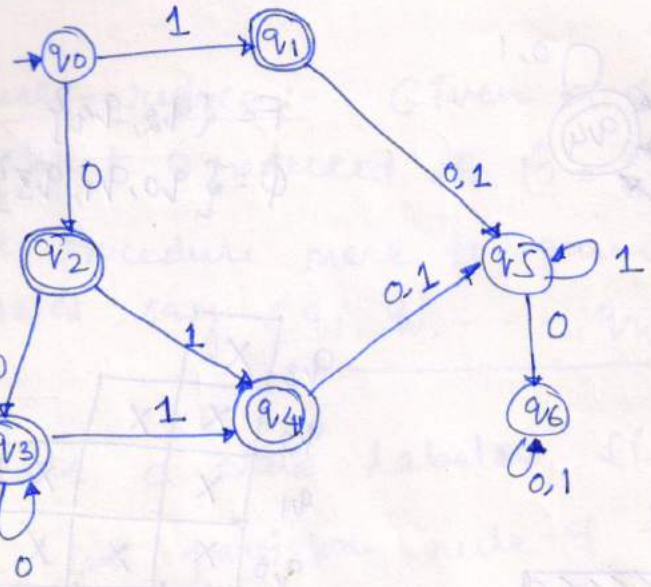
$\delta(q_2, 0) = q_1$
$\delta(q_4, 0) = q_4$ } distinguishable

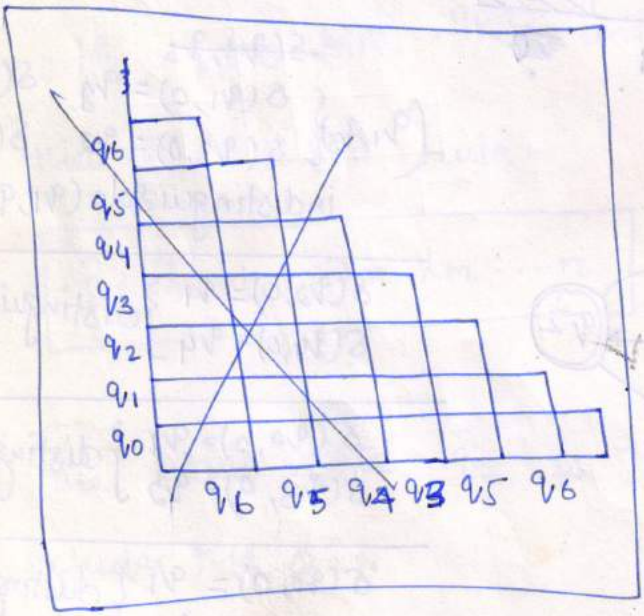
$\delta(q_0, 0) = q_1$
$\delta(q_3, 0) = q_2$ } distinguishable.

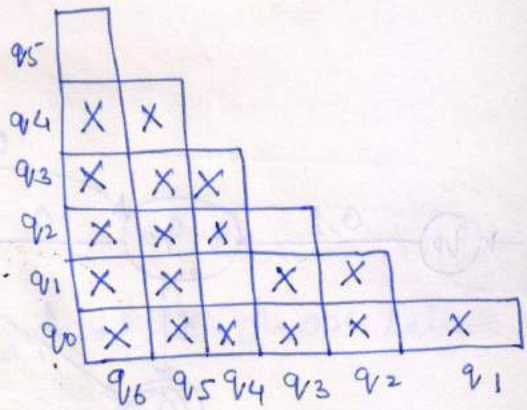
$\delta(q_0, 0) = q_1$
$\delta(q_1, 0) = q_2$ } distinguishable

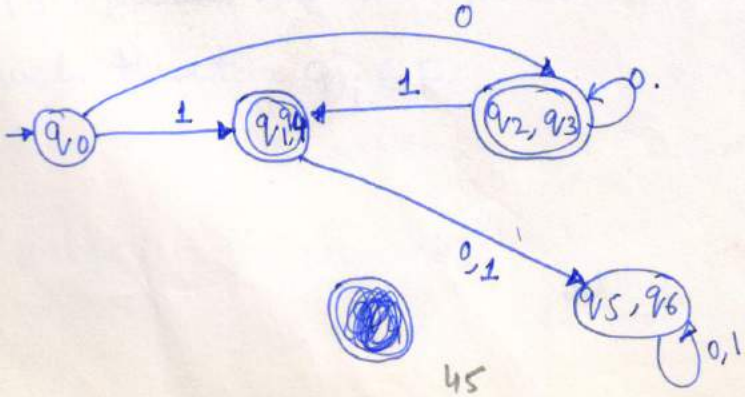


Example :- Minimize the following DFA —



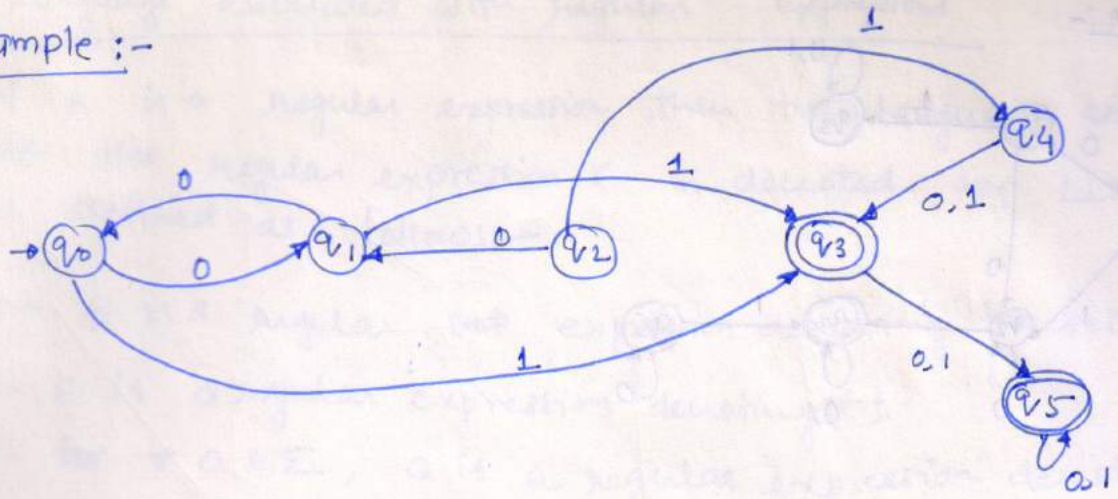




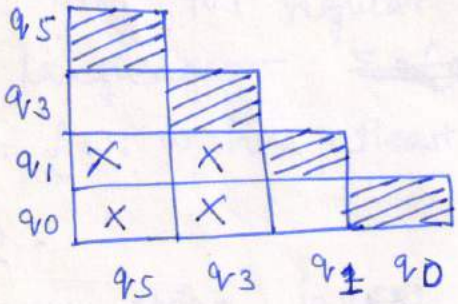


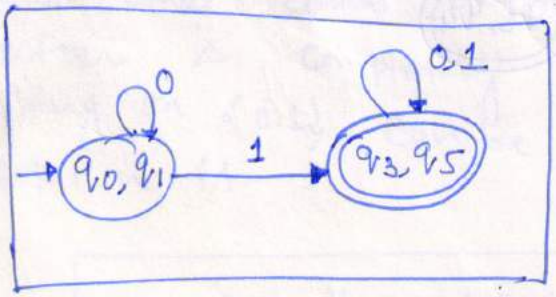
(minimized dfa)

Example :-



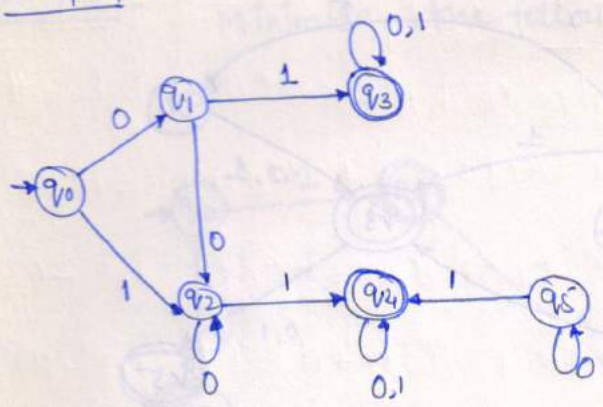
State q_2, q_4 are not reachable state. So remove the states q_2, q_4 from the transition diagram.



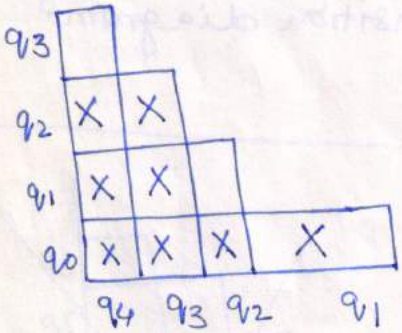


Minimized DFA

Example :-



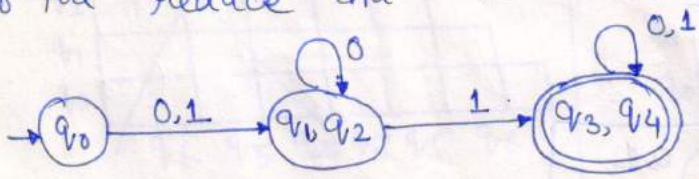
Solution :- State q_5 is unreachable (inaccessible). Remove state q_5 from the given dfa.



state (q_3, q_4) are indistinguishable.

state (q_1, q_2) " "

So the reduce dfa is-



Language associated with regular Expressions

If r is a regular expression then the language associated with the regular expression r is denoted by $L(r)$ is defined as follows:-

1) - ϕ is a regular set expression denoting the set $\{\phi\}$
2) - ϵ is a regular expression denoting $\{\epsilon\}$.
3) - for $\forall a \in \Sigma$, a is a regular expression denoting $\{a\}$.

if r_1 & r_2 are two r.e then —

4) - $L(r_1+r_2) = L(r_1) \cup L(r_2)$
5) - $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$
6) - $L(r_1^*) = (L(r_1))^*$

Exp:- Give the regular expression for the following language ~~$\Sigma = \{a, b\}$~~ $\Sigma = \{0, 1\}$

$L = \{w : w$ has atleast one pair of consecutive zeros $\}$

Solution:-

Every string in $L(r)$ must contain 00 somewhere but what comes before and what ~~comes~~ goes after is completely arbitrary. An arbitrary string on $\{0,1\}$ can be denoted by $(0+1)^*$. So the solution is —

$$r = (0+1)^* \; 00 \; (0+1)^*$$

Regular expression to NFA

Question-1 :- find an nfa for the following regular expressions—

1) $r = ab^* aa + bb a^* ab$

2) $r = (a+b)^* \; b\,(a+bb)^*$

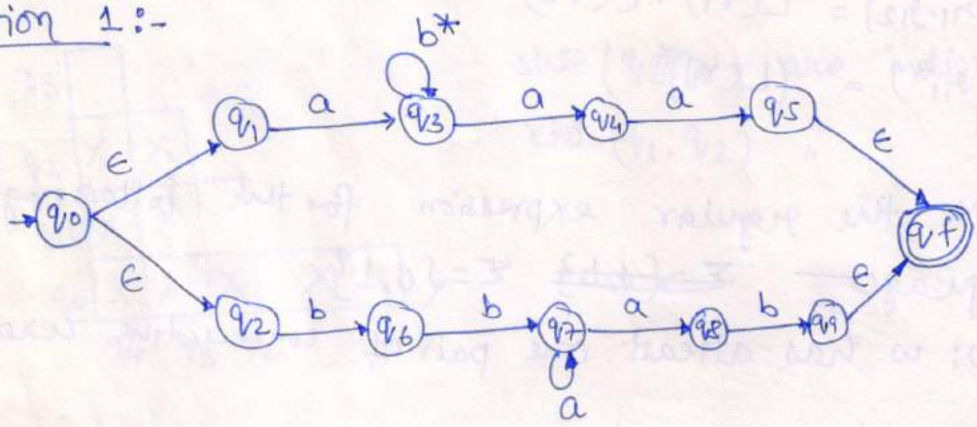
3) $r = aa^* + ab\, a^* b^*$

4) $r = ab\,(a+ab)^*\,(a+aa)$

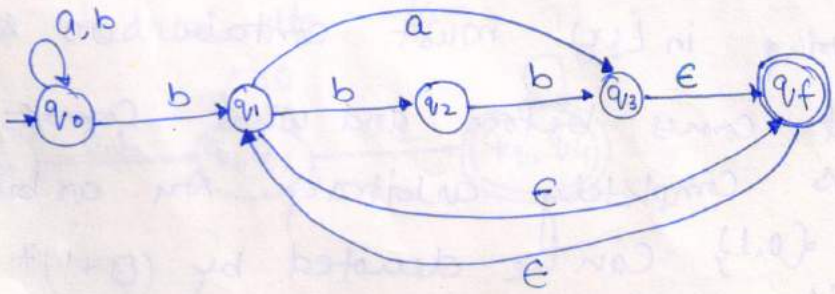
5) $r = (ab\,ab)^* + (aaa^* + b)^*$

6) $r = ((aa^*)^*\, b)^*$

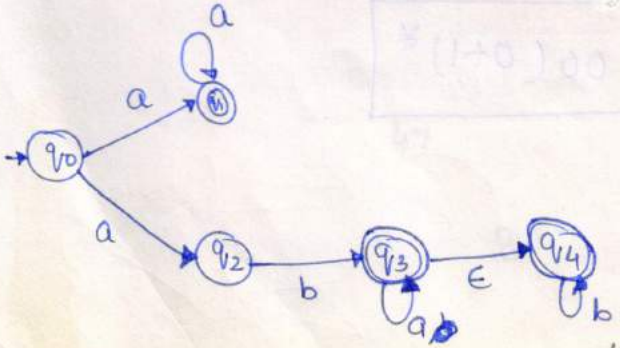
Solution 1:-



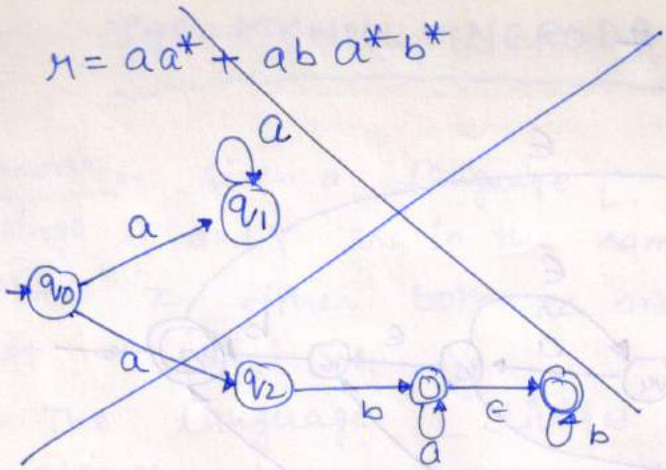
2:-



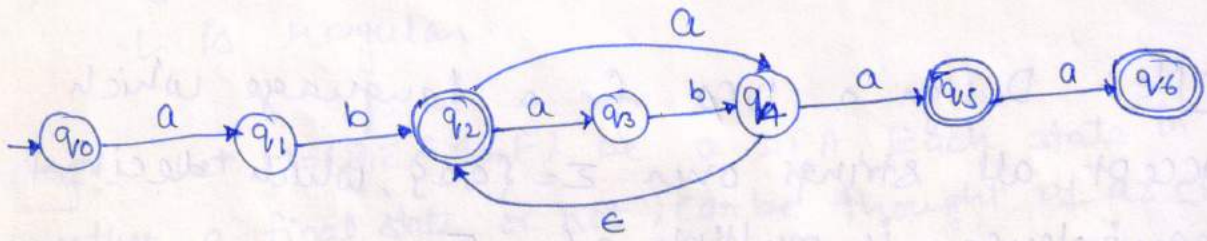
3:-



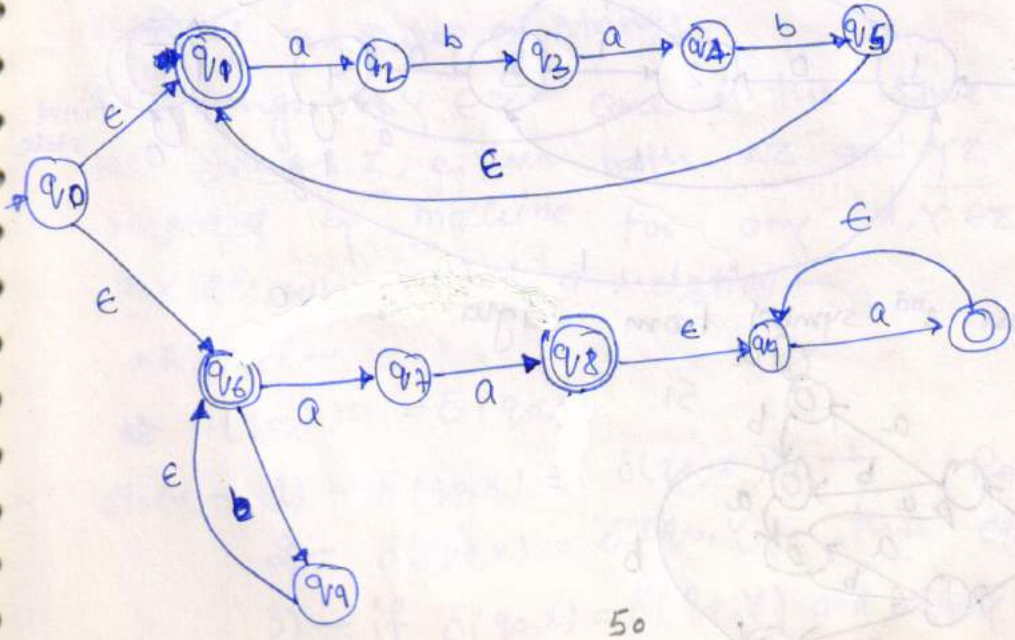
4:- $r = aa^* + ab\,a^*b^*$



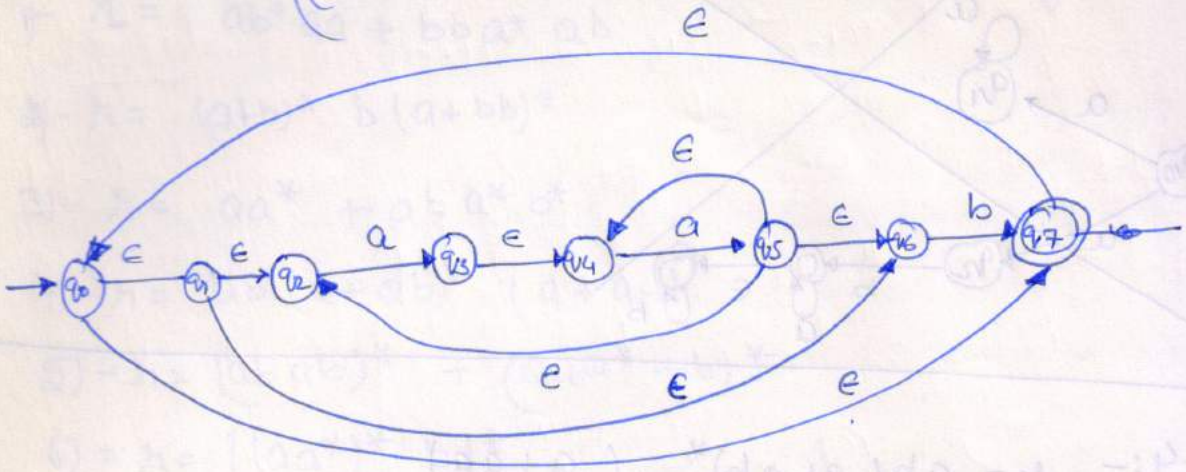
Solution-4:- $r = ab(a+ab)^*(a+aa)$



5:- $r = (ab\,ab)^* + (aaa^* + b)^*$

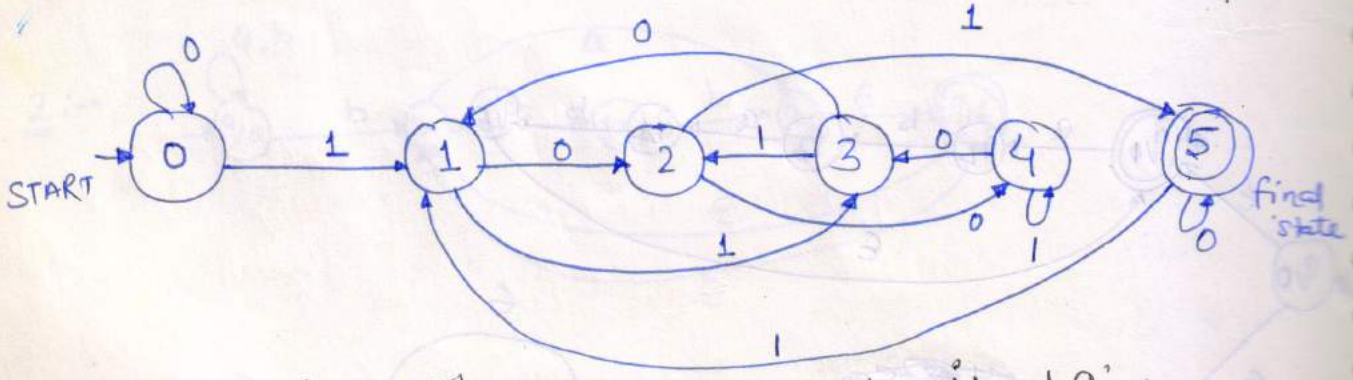


b - $r = ((aa^*)^* b)^*$

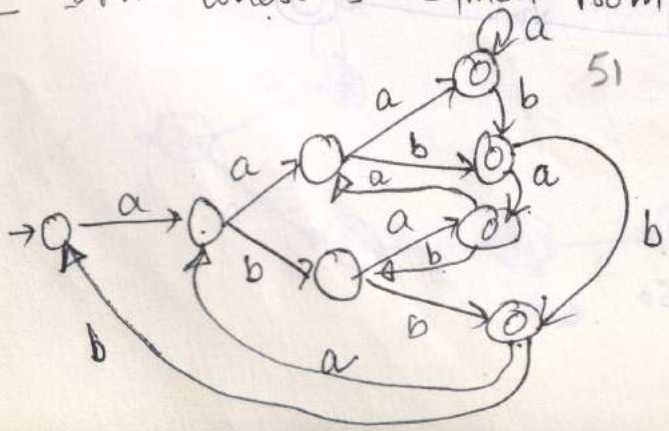


Example:- Design a DFA for a language which accept all strings over $\Sigma = \{0, 1\}$, which decimal equivalence is multiple of 5.

Solution:-



Ex. DFA whose 3rd symbol from Right is 'a'.



51

THE-MYHILL-NERODE THEOREM

Theorem:- Given a language L, we shall say that any two strings x and y are in the same class if for all possible strings z either both xz and yz are in L or both are not.

1)- The language L divides the set of all possible strings into separate (mutually exclusive) classes.

2)- If L is regular, the number of classes are finite.

3)- If the no. of classes L creates is finite, then L is regular.

Proof:- Let $M=(Q,\Sigma,\delta,q_0,F)$ be a DFA. Each state in M whether a final state or not, can be thought of as creating a group of a certain class of strings. Two strings can be said to both belong to the group of state q if they both trace a path from start state to q even if the path are very different. Thus we can say that every state defines a group of strings.

If any two strings $x \& y \in \Sigma^*$ are in the same group then for all strings z, either both xz and yz are accepted or rejected by machine. for any $x, y \in \Sigma^*$, let xRy ie $(x,y) \in R$, where R is a relation —

xRy iff —

$\delta(q_0,x) = \delta(q_0,y)$

since - 1)- $\delta(q_0,x) = \delta(q_0,x)$ — Reflexive

2)- $\delta(q_0,x) = \delta(q_0,y)$. then $\delta(q_0,y) = \delta(q_0,x)$ — Symmetric

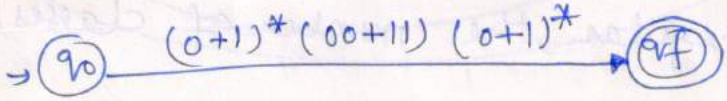
3)- if $\delta(q_0,x) = \delta(q_0,y)$ and $\delta(q_0,y) = \delta(q_0,z)$ then $\delta(q_0,x) = \delta(q_0,z)$. Equivalence relation.

Regular expression to finite automata:-

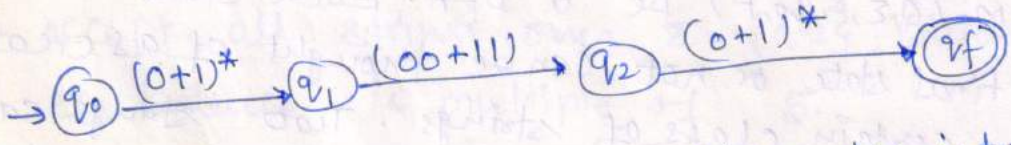
example - $r = (0+1)^* (00+11) (0+1)^*$

step-1:- create an intial state q_0 and a final state q_f.

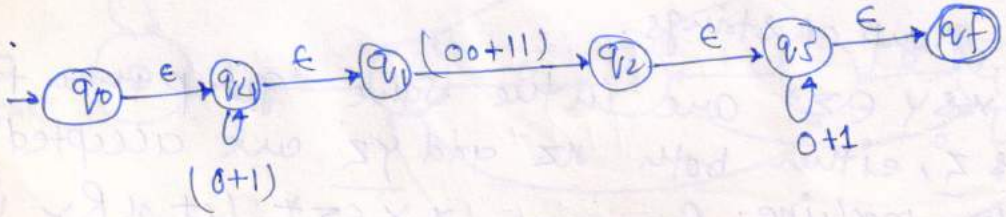
2)- Make a transition from q_0 to q_f labeled the given regular expression. ie-



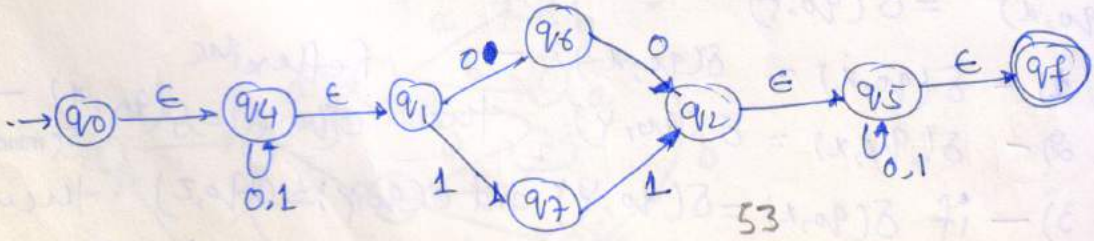
3)- Eliminate the concatenation in the r.e by introducing new vertices q_1 & q_2. ie-



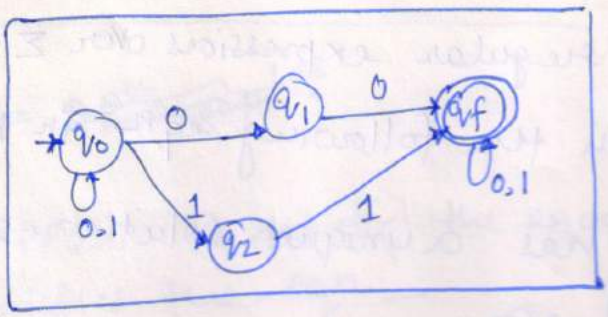
4)- Eliminate the * operations by introducing new states. ie-



5)- eliminate + operation & concatenation. ie-



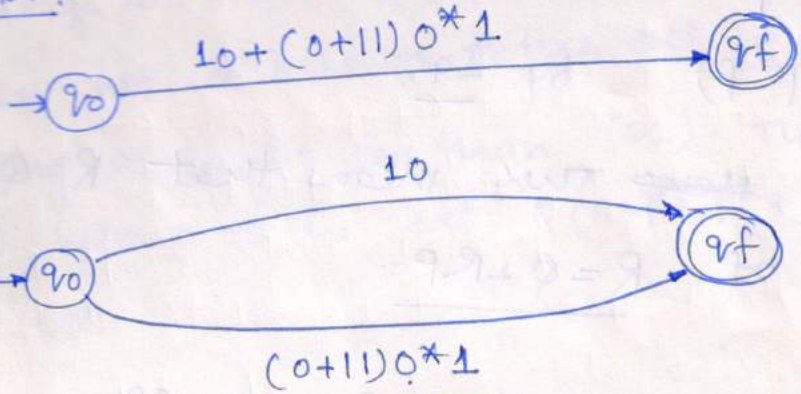
6)- Eliminate the ϵ-moves and we get-

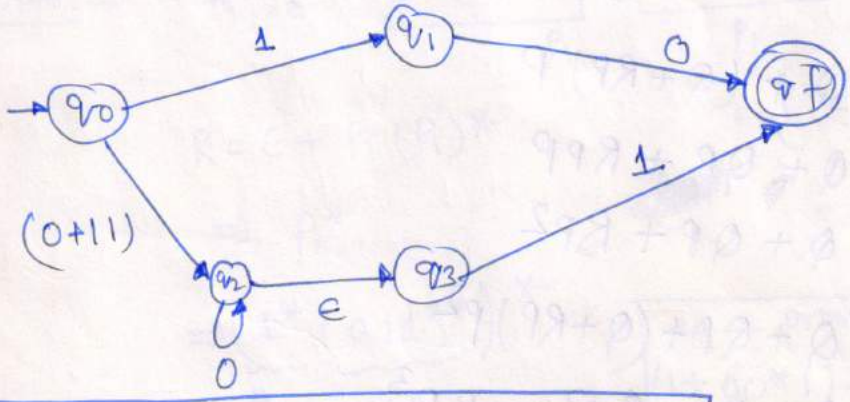


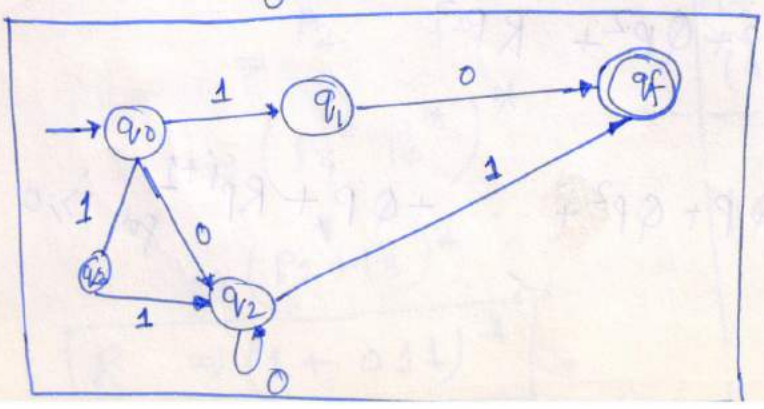
Example:- Construct NFA for the following regular expression-

$r = 10 + (0+11)\,0^*1$

Solution:-







ARDEN'S THEOREM :-

Let P and Q be two regular expressions over Σ. If P does not contain ϵ, then the following eqn in R,

$$R = Q + RP \quad \rightarrow (1)$$

has a unique solution, given by —

$$R = QP^*$$

Proof :- Put $R = QP^*$ in eq^n (1) if it is a solution—

$$R = Q + (QP^*)P$$

$$= Q + QP^*P$$

$$\Rightarrow Q(\epsilon + P^*P) \quad \text{by Iq.}$$

$R = QP^*$ ~~Hence~~ This means that $R = QP^*$ is a solution of $R = Q + RP$.

To prove uniqueness — Replace R by $Q + RP$

$$Q + RP = Q + (Q + RP)P$$

$$= Q + QP + RPP$$

$$= Q + QP + RP^2$$

$$= Q + QP + (Q + RP)P^2$$

$$\Rightarrow Q + QP + QP^2 + RP^3$$

$$= Q + QP + QP^2 + \dots + QP^i + RP^{i+1} \quad \text{for } i \geq 0$$

$R = Q(\epsilon + P + P^2 + \cdots + P^i) + RP^{i+1}$ for $i \geq 0$

$= QP^* + RP^{i+1}$

Suppose R satisfy the equation $R = Q + RP$, then it satisfies the eqn —

$R = Q(\epsilon + P + P^2 - - - + P^i) + RP^{i+1}$. for. $i \geq 0$

⇒ Let w be a string of length i in the set R. As we know that P does not contain ϵ, then w belongs to the set $Q(\epsilon + P + P^2 + \cdots P^i)$, because w does not belongs to the set RP^{i+1} (It has no string of length less than i+1). This means that w belongs to the set $Q(\epsilon + P + P^2 + \cdots + P^i)$, and hence to QP^*.

Example:— $R = \epsilon + \underbrace{1^*(011)^*}_{P_1}\left(\underbrace{1^*(011)^*}_{P_1}\right)^*$

$\{\epsilon + rr^* = r^*\}$

$R = \epsilon + P_1(P_1)^*$

$\Rightarrow P_1^*$

$\Rightarrow \left(\underbrace{1^*}_{P_2}\underbrace{(011)^*}_{P_3}\right)^*$

$\Rightarrow$

$\Rightarrow (P_2^* P_3^*)^*$

$\Rightarrow (P_2 + P_3)^*$

$R \Rightarrow (1 + 011)^*$

ex — prove that

$(1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^* (0 + 10^*1) = 0^*1(0 + 10^*1)^*$

$(1 + 00^*1) \Rightarrow (\epsilon + 00^*)1 = 0^*1$

Application of Arden's theorem :-

This is used to find the r.e recognised by a transition system (FA).

The following assumptions are made regarding the transition system

1) – It does not contains ϵ-moves.

2) – It has only one initial state: say V_1.

3) – Its vertices are $V_1, V_2 - - - V_n$.

4) – V_i the r.e represents the set of strings accepted by the system even though V_i is a final state.

5) – α_{ij} denotes the r.e representing the set of labels of edges from V_i to V_j. So- when there is no such edge, $\alpha_{ij} = \phi$. So-

$$V_1 = V_1\alpha_{11} + V_2\alpha_{21} + - - - - - + V_n\alpha_{n1} + \epsilon$$

$$V_2 = V_1\alpha_{12} + V_2\alpha_{22} + - - - - - + V_n\alpha_{n2}$$

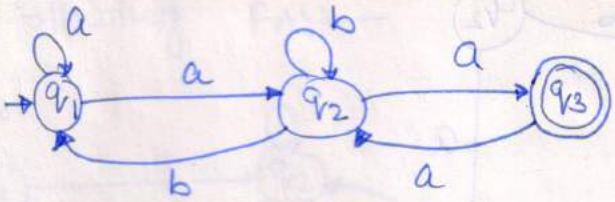
$$\vdots$$

$$V_n = V_1\alpha_{1n} + V_2\alpha_{2n} + - - - - + V_n\alpha_{nn}$$

By repeatedly applying substitutions and Arden's theorem we can express V_i in terms of α_{ij}'s.

⇒ Take the union of V_i's if more than one final state are there.

Example:-



$q_1 = q_1a + q_2b + \epsilon$ —— (I)

$q_2 = q_1a + q_2b + q_3a$ —— (II)

$q_3 = q_2a$ —— (III)

put $q_3 = q_2a$ in eqn (II) -

$q_2 = q_1a + q_2b + q_2aa$

$\underset{R}{q_2} = \underbrace{q_1a}_{Q} + \underbrace{q_2}_{R}\underbrace{(b + aa)}_{P}$

compare with $R = Q + RP$

$q_2 = q_1a(b+aa)^*$

substituting q_2 in eqn (I) -

$q_1 = q_1a + q_1a(b+aa)^*b + \epsilon$

$\underset{R}{q_1} = \underset{R}{q_1}\underbrace{(a + a(b+aa)^*b)}_{P} + \underset{\theta}{\epsilon}$

Hence

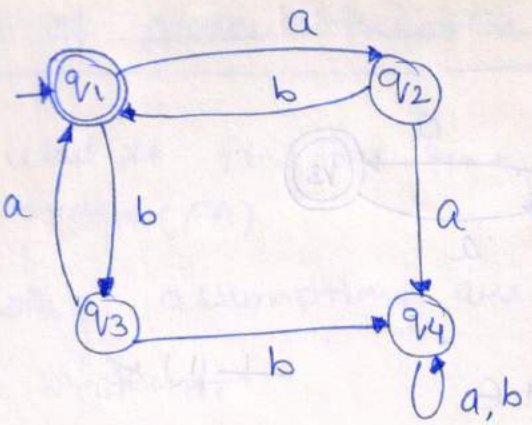
$q_1 \Rightarrow [a + a(b+aa)^*b]^*$

$q_2 = [a + a(b+aa)^*b]^* a(b+aa)^*$

Required Regular expression -

$q_3 = [a + a(b+aa)^*b]^* a(b+aa)^*a$

Example :-



Solution :-

$$q_1 = q_2 b + q_3 a + \epsilon$$

$$q_2 = q_1 a *$$

$$q_3 = q_1 b *$$

$$q_4 = q_2 a + q_3 b + q_4 (a+b)$$

Solve for q_1 because q_1 is the final state —

$$q_1 = q_2 b + q_3 a + \epsilon$$

$$= q_1 ab + q_1 ba + \epsilon$$

$$\underset{R}{q_1} \Rightarrow \underset{R}{q_1} (\underbrace{ab + ba}_{P}) + \underbrace{\epsilon}_{Q}$$

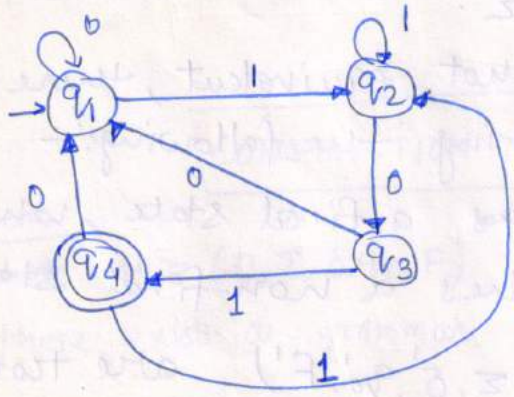
$$R = Q + RP$$

$$\Rightarrow QP^*$$

$$q_1 = \epsilon (ab + ba)^*$$

$$q_1 = (ab + ba)^*$$

Example:- find the regular expression equivalent to the following FA:-



Solution:-

$$q_1 = q_1 0 + q_3 0 + q_4 0 + \epsilon$$

$$q_2 = q_1 1 + q_2 1 + q_4 1$$

$$q_3 = q_2 0$$

$$q_4 = q_3 1$$

Solve for q_4 (final state) —

$$q_4 = q_3 1 \Rightarrow q_2 01 \quad \text{——— (1)}$$

$$q_2 = q_1 1 + q_2 1 + q_4 1 \Rightarrow q_1 1 + q_2 1 + q_2 011$$

$$\Rightarrow q_1 1 + q_2 (1 + 011)$$

$$q_2 = \underline{q_1 1 (1 + 011)^*}$$

$$q_1 = q_1 0 + q_2 00 + q_2 010 + \epsilon \Rightarrow q_1 0 + q_2 (00 + 010) + \epsilon$$

$$= q_1 0 + q_1 1 (1 + 011)^* (00 + 010) + \epsilon$$

$$\Rightarrow q_1 (0 + 1 (1 + 011)^* (00 + 010)) + \epsilon$$

$$q_1 = \underline{[0 + 1(1 + 011)^* (00 + 010)]^*}$$ put in eqn (1) —

$$q_4 \longrightarrow [0 + 1(1 + 011)^* (00 + 010)]^* 01$$

EQUIVALENCE OF TWO FINITE AUTOMATA

To Two F.A over Σ are equivalent if they accepts the same set of strings over Σ.

⇒ When the two f.A are not equivalent, there is some string $w \in \Sigma$ satisfying the following:-

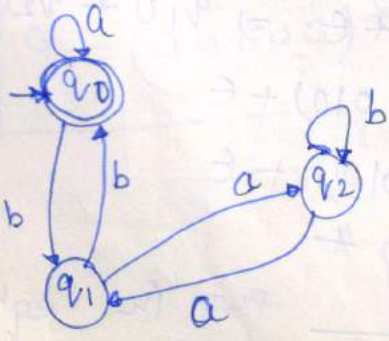
1)- One automaton reaches a final state, whereas the other automaton reaches a non-final state.

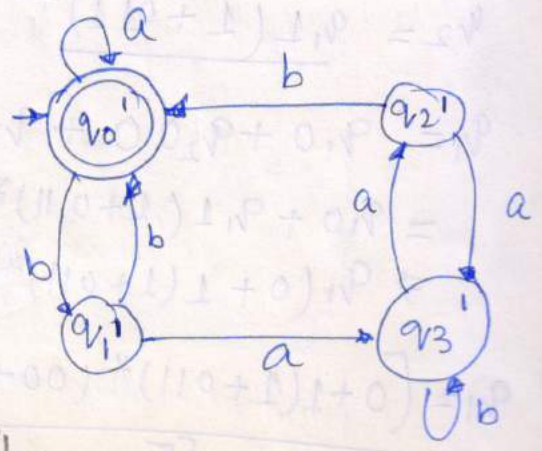
Let $M = (Q, \Sigma, \delta, q_0, F)$ & $M' = (Q', \Sigma, \delta', q_0', F')$ are two F.A over Σ.

(q, q')	$\underline{a}$ (q_a, q_a')	$\underline{b}$ (q_b, q_b')
(q_0, q_0')	(q_0, q_0')	(q_1, q_1')
(q_1, q_1')	(q_2, q_3')	(q_0, q_0')
(q_2, q_3')	(q_1, q_2')	(q_2, q_3')
(q_1, q_2')	(q_2, q_3')	(q_0, q_0')

⇒ If we reach a pair (q, q') such that q is a Final state of M, and q' is a non-final state of M' or vice versa, terminate the construction and conclude that M & M' are not equivalent.

Ex-





As we do not get a pair (q, q') where q is final and q' is non-final (or vice versa) at every row. Therefore M and M' are equivalent.

CONSTRUCTION OF REGULAR GRAMMAR FROM FA

Theorem:-

Let $M = (Q, \Sigma, \delta, q_0, F)$. If ω is in $L(M)$ then there exists a grammar $G = (V, T, P, S)$ such that $L(M) = L(G)$ or $L(G) = L(M)$

Proof:- Let $Q = \{q_0, q_1, \dots q_n\}$

then

$V = \{A_0, A_1, \dots A_n\}$

$S = A_0$

$T = \Sigma$

where P is defined by the following rules —

(1) - $A_i \to a A_j$ is included in P if $\delta(q_i, a) = q_j$. $q_j \notin F$.

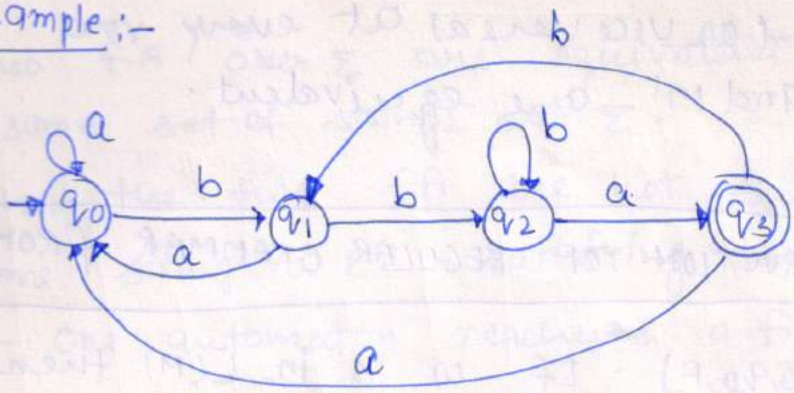
(2) - $A_i \to a A_j$ and $A_i \to a$ are included in P if $\delta(q_i, a) = q_j \in F$

So

$A_0 \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \dots \Rightarrow a_1 a_2 \dots a_k$

iff. $\hat{\delta}(q_0, a_1 a_2 \dots a_k) \in F$

example :-



$A_0 \rightarrow a A_0$
$A_0 \rightarrow b A_1$
$A_1 \rightarrow a A_0$
$A_1 \rightarrow b A_2$
$A_2 \rightarrow a A_3$
$A_2 \rightarrow a$
$A_2 \rightarrow b A_2$
$A_3 \rightarrow a A_0$
$A_3 \rightarrow b A_1$

P

$V = (A_0, A_1, A_2, A_3)$

Let $A_0 \Rightarrow a A_0 \Rightarrow a b A_1 \Rightarrow abb A_2 \Rightarrow abba$

$\rightarrow q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_2 \xrightarrow{a} (q_3)$

Hence $w = abba$ is generated by G and also accepted by M. So

$L(G) = L(M)$

CONSTRUCTION OF F.A FOR A GIVEN REGULAR GRAMMAR

Let $G = (\{A_0, A_1, \ldots A_n\}, \Sigma, P, A_0)$ is a grammar.
We construct a transition system M whose -

(i)- States correspond to variables.

(ii)- Initial state corresponds to A_0.

(iii)- Transition in M is correspond to productions in P.

We define M as $(\{q_0, q_1, \ldots q_n\}, \Sigma, \delta, q_0, \{q_f\})$
q_f

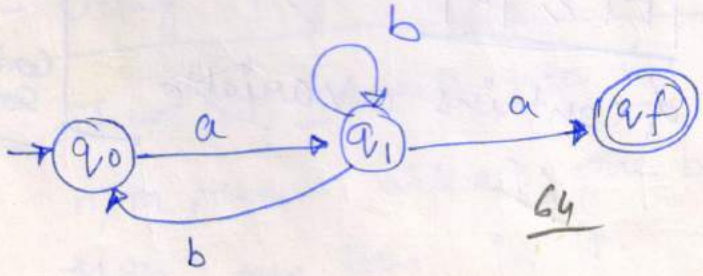
Where δ is defined as follows -

1)- Each production $A_i \to aA_j$ include a transition from q_i to q_j with labeled a.

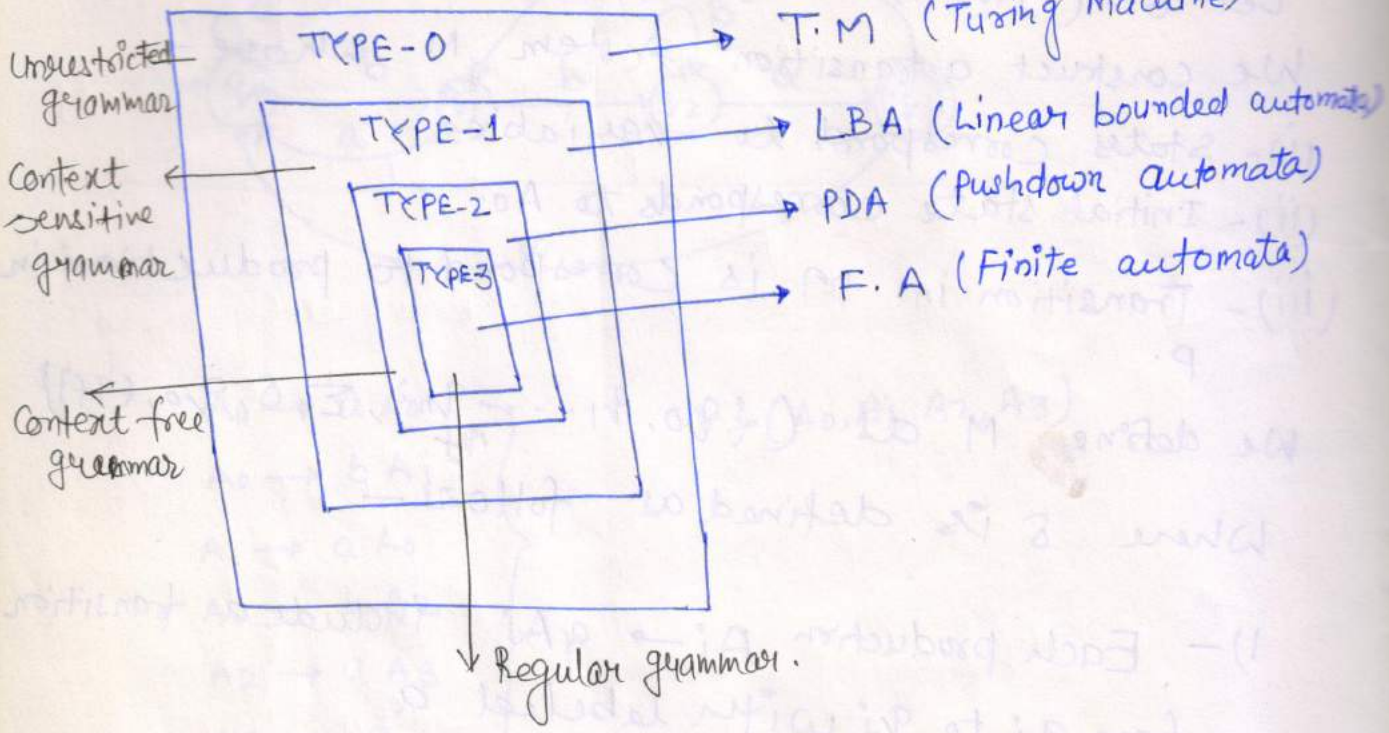
2)- Each production $A_K \to a$ includes a transition from q_K to q_f with label a.

Then $\boxed{L(M) = L(G)}$

example :- Let $G = (\{A_0, A_1\}, \{a, b\}, P, A_0)$ where P consist of $A_0 \to aA_1$, $A_1 \to bA_1$, $A_1 \to a$, $A_1 \to bA_0$



CHOMSKY HIERARCHY



Type-0 grammar :- $\alpha \longrightarrow \beta$

$\alpha, \beta \in (V \cup T)^*$

① α contains atleast one variable.

② α can not be NULL

Type-1 grammar :-

$\alpha \longrightarrow \beta$, $(\alpha, \beta) \in (V \cup T)^*$

① $|\beta| \geq |\alpha|$

② α contains a variable

Context Sensitive

Type-2 grammar :- $A \to \alpha$

$\alpha \in (V \cup T)^*$

$A \in V$

} Context free

Type-3 grammar :- $A \to aB$

$A \to a$

$A, B \in V$

$a \in T^*$

} Regular

Left linear grammar ($V \to VT^*$) or $V \to T^*$

Right linear grammar ($V \to T^*V$) or $V \to T^*$

PUMPING LEMMA FOR REGULAR LANGUAGES

Identifying Non-regular languages :-

Regular languages can be infinite, as most of our examples have demonstrated. A language is regular only if, in processing any string, the information that has to be remembered at any stage is strictly limited.

1) - Using pigeonhole principle :-

If we put n objects into m boxes (pigeonholes) and if $n > m$, then atleast one box must ~~be~~ have more than one item in it.

Example:- Is the language $L=\{a^n b^n : n \geq 0\}$ regular?

Let us suppose L is regular. Then some dfa M, $M=(Q, \{a,b\}, \delta, q_0, F)$ exists for it.

Let $\hat{\delta}(q_0, a^i) =$ for $i = 1,2,3 \ldots\ldots\ldots$

since there are infinite no. of i's, but only a finite no. of states in M, the pigeonhole principle tells us that there must be some state, say, q, such that

$$\hat{\delta}(q_0, a^n) = q$$

and $\hat{\delta}(q_0, a^m) = q$

with $n \neq m$. But since M accepts $a^n b^n$ we must have -

$$\hat{\delta}(q, b^n) = q_f \in F$$

$$\Rightarrow \hat{\delta}(q_0, a^m b^n) = \hat{\delta}(\hat{\delta}(q_0, a^m), b^n)$$

$$= \hat{\delta}(q, b^n)$$

$$= q_f$$

This contradicts the original assumption that M accepts $a^m b^n$ only if $n=m$, and leads us to conclude that L cannot be regular.

Application of my-Hill Nerode theorem

Example :- Let $L = \{w | w$ has even no. of 0's & 1's$\}$

In w there are four possible classes –

C_1 = Even No. of 0's & 1's.

C_2 = Even no. of 0's & odd no. of 1's

C_3 = Odd no. of 0's & even no. of 1's.

C_4 = odd no. of 0's & odd no. of 1's.

There are only 4 classes. So the language is regular.

Minimization of DFA

1) – Remove unreachable states.

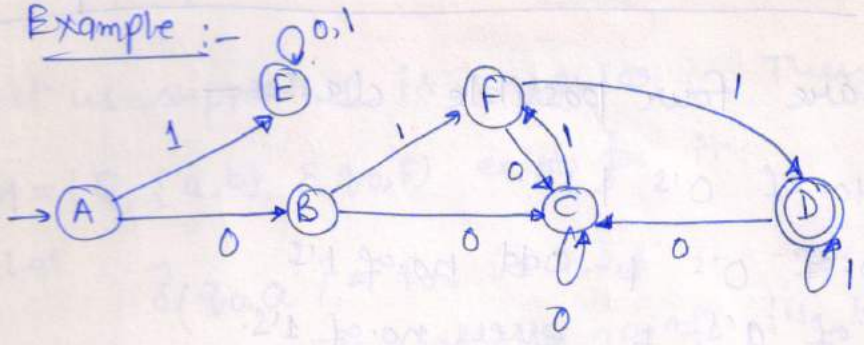
2) – Create two groups. One for final states and another for non-final states.

3) – partition the groups into no. of groups if they (p,q) are in different groups. ie. two states, are in same group if for any string z,

$\delta(p, z)$ & $\delta(q, z)$ belongs to the same group.

~~$\delta(p, z)$~~.

Example :-



1)- There are no unreachable states.

2)-

group-I (A, B, C, E, f)

group-II — (D)

$\delta(A,0) = B$
$\delta(B,0) = C$
$\delta(C,0) = C$
$\delta(E,0) = E$
$\delta(f,0) = C$

$\delta(A,1) = E$
$\delta(B,1) = F$
$\delta(C,1) = F$
$\delta(E,1) = E$
$\delta(F,1) = D$

Partition of group for input 0 is not possible, since all the members goes to the states which are in the group I. But for input 1, state f is distinguishable because it goes to group-II.

Hence.

A, B, C E	f	D
I	II	III

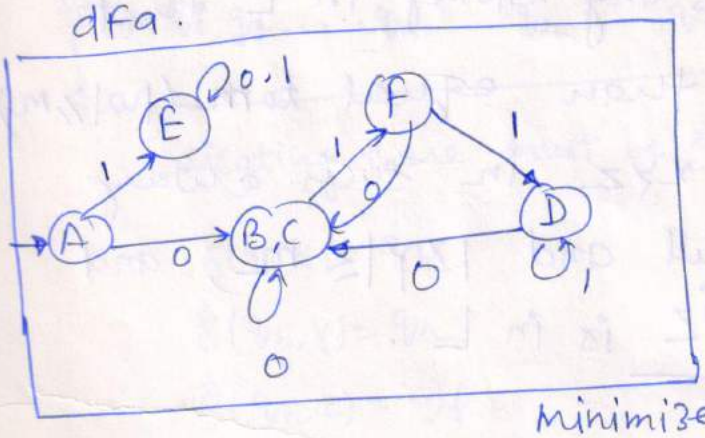
Again for group-I & input 1 - State B & C goes to group II. So

A,E	B,C	F	D
I	II	III	IV

for input 0, group-I . A goes to group II & E goes to group-I. So -

A	E	B,C	F	D
I	II	III	IV	V

So there are only five states in the minimized dfa.



Minimized dfa.

The pumping Lemma for regular languages

pumping lemma is a powerful tool for proving certain languages nonregular. It is called "pumping" because we pump more stuff into the middle of the word. It is called a "Lemma" because, although it is a theorem, its main importance is as a tool in proving other results of more direct interest; namely, it will help us to prove that certain languages are nonregular.

Theorem:- Let L be infinite regular language. Then there is a constant m (m is not greater than the number of states of smallest FA accepting L) such that if w is any string in L that have length greater than equal to m ($|w| \geq m$) we may write $w = xyz$ in such away that y is not null and $|xy| \leq m$, and for all $i \geq 0$, xy^iz is in L.

ie- 1)- $|w| \geq m$

2)- $|y| > 0$

3)- $|xy| \leq m$

then $xy^iz \in L$

Proof:- If L is regular, there exists a DFA M $=(Q, \Sigma, \delta, q_1, F)$ that recognizes it. Let $Q = \{q_0, q_1, q_2 \cdots q_m\}$

Now take a string $w \in L$ such that ~~[illegible]~~.

$$|w| \geq m = n$$

Since L is assumed to be infinite, this can always be done. consider the set of states the automaton goes through as it processes w, say, —

$$q_1\, q_i, q_j \ldots\ldots q_f$$

Since this sequence has exactly $|w|+1$ entries, at least one state must be repeated, and such a repetition must start no later than the n^{th} move. Thus the sequence must look like —

$$q_0 . q_i, q_j \ldots \cdot q_r \ldots\ldots q_r \ldots\ldots q_f$$

indicating there must be substrings x, y, z of w such that-

$$\hat{\delta}(q_0, x) = q_r$$
$$\hat{\delta}(q_r, y) = q_r$$
$$\hat{\delta}(q_r, z) = q_f$$

with $|xy| \leq n = m$ & $|y| \geq 1$. from this it immediately follows that

$$\hat{\delta}(q_0, xz) = q_f$$

as well as -

$$\hat{\delta}(q_0, xy^2z) = q_f$$
$$\hat{\delta}(q_0, xy^3z) = q_f$$

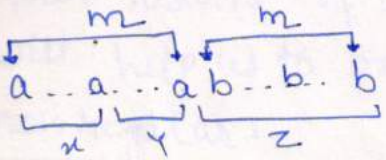
So xy^iz is also accepted by dfa and in L. Proved

Example:- 1-

Use the pumping to show that $L = \{a^n b^n : n \geq 0\}$ is not regular.

Solution:- 1- Let m be the constant given by pumping lemma. Let L is regular.

2)- Let $w = a^m b^m$, $|w| = 2m > m$



Y contains entirely of a's. Let $|Y| = K > 1$

Then $xy^i z$ is also in L.

Let $i = 0 \Rightarrow xz$ is also in L.

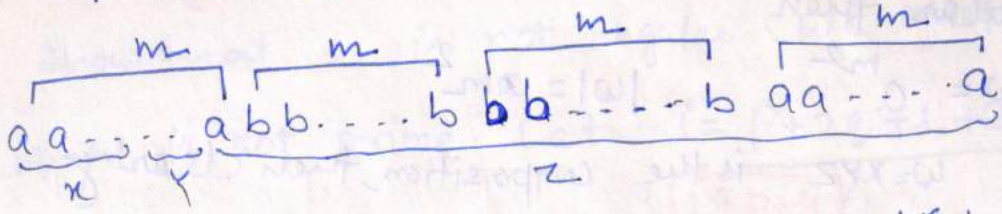
$\Rightarrow a^{m-K} b^m \in L$ but it is not true because L contains equal no. of a's & b's. So we can say that L is not regular.

Example:-2:- Show that $L = \{ww^R : w \in \Sigma^*\}$ is not regular, where $\Sigma = \{a, b\}$.

Solution:- Let us assume that L is regular and m is the constant given by the finite automata.

Let $\omega = a^m b^m b^m a^m$

$|\omega| = 4m > m$



~~Let~~ according to pumping lemma - $|xy| \leq m$ & $|y| > 1$.

due to this restriction we are bind to choose y entirely of a's. Let $|y| = K > 1$

So $xy^i z \in L$ according to pumping lemma.

Let $i = 0$ then

$xz = a^{m-K} b^m b^m a^m \notin L$.

So we can say that L is not regular.

Example :- 3 :- $L = \{\omega \in \Sigma^* : n_a(\omega) < n_b(\omega)\}$ is not regular.

Solution :- Let m be the constant given by pumping Lemma. Since we have complete freedom to choose ω. Let $\omega = a^m b^{m+1}$. Now because $|xy|$ cannot be greater than m, so pick y with all a's.

Let $y = aa$.

For $i = 2$.

$\omega_2 = xy^2z = a^{m+2} b^{m+1} \Rightarrow n_a(\omega) > n_b(\omega)$

but this is not true. So the language L is not regular.

Example :- Show that $L = \{a^n : n \text{ is a perfect square}\}$ is not regular

Solution :- Let m be the constant and assume L is regular then

$$w = a^{m^2} \qquad |w| = m^2$$

if $w = xyz$ is the composition, then clearly

$y = a^k$ with $1 \le k \le m$

In this case $w_0 = a^{m^2 - k}$

But $(m^2 - k) > (m-1)^2$

$m^2 - k = (m-1)^2$

$m^2 - k = m^2 + 1 - 2m$

$k = (2m - 1)$

but $1 \le k \le m$

so that w_0 cannot be in L. Therefore the language is not regular.

Example: Show that $L = \{a^n : n \text{ is prime}\}$ is not regular.

Solution :- Let L is regular and m be the constant given by pumping lemma.

we have $L = \{a^2, a^3, a^5, a^7 \ldots\}$

Let $x = a^p$, $y = a^q$, $z = a^r$ such that $(p+q) \le m$, $q \ge 1$

$w = xyz = a^p a^q a^r \Rightarrow a^{p+q+r}$

By pumping lemma xy^iz must be in L. ie

$a^{(p+iq+r)} \in L \Rightarrow$ $p+iq+r$ must be prime.

To show that L is not regular we must prove that $p+iq+r$ is not prime. Let $i = p+2q+r+2$ then—

$= p+(p+2q+r+2)q+r$

$= p+pq+2q^2+qr+2q+r$

$\Rightarrow p(1+q)+2q(q+1)+r(q+1)$

$\Rightarrow (q+1)(p+2q+r)$

$i = p+r \Rightarrow p+(p+r)q+r$

$\Rightarrow p+pq+qr+r$

$\Rightarrow p(1+q)+r(1+q)$

$\Rightarrow (1+q)(p+r)$.

as we know that $q \geq 1$ so $q+1 \geq 2$. ie—

$(p+iq+r)$ is a product of two natural numbers. So $(p+iq+r)$ is not prime. So L is not regular.

example:- $L = \{0^{n!} : n \geq 0\}$. show that L is not regular.

solution:- In L the length of strings are factorial of any number. Let us assume that L is regular and m is the constant given by the pumping lemma. To chose string w such that $|w| \geq m$, we choose

$w = 0^{m!}$ (if $m < 3$ then choose $w = 0^{3!}$)

Let $w = xyz$ in such a way $|x| \geq 1$, $|xy| \leq m$

Let $|y| = k \leq m$

Thus if $|xyz| = m!$

Then $|xz| = m! - k$ or $|xy^0z| = m! - k$

Since for $m > 2$ and $k \leq m$ we have

$(m! - k) > (m-1)!$

Thus $(m! - k)$ is not factorial of any number and $xy^2 \notin L$. Therefore L is not regular.

EXERCISE

prove that following languages are not regular -

1) — $L = \{a^n b^n c^n d^n : n \geq 0\}$

2) — $L = \{ww^R : w \in \{a,b\}^*\}$

3) — $L = \{w = w^R : w \in \{a,b\}^*\}$

4. $L = \{a^n b^m a^n : n, m \geq 0\}$

5 - $L = \{a^{m+n} b^n c^m : m, n \geq 0\}$

CLOSURE PROPERTIES of REGULAR LANGUAGES

(1) **Theorem:-** If L_1 and L_2 are regular languages then $L_1 \cup L_2$, $L_1 L_2$, L_1^* are also regular languages ie the set of regular languages are closed under union, concatenation & Kleene closure.

Proof:- If L_1 and L_2 are regular languages, there are regular expressions r_1 and r_2 that defines L_1 & L_2 respectively. As we know that if r_1 & r_2 are two regular expressions then -

1)- $r_1 + r_2$ is a regular expression denoting the language $L_1 + L_2$ ie $L_1 \cup L_2$.

2) $r_1 . r_2$ is a regular expression denoting the language $L_1 \cdot L_2$.

3)- $(r_1)^*$ is a regular expression denoting the set $(L_1)^*$.

So, $L_1 \cup L_2$, $L_1 \cdot L_2$, $(L_1)^*$ are closed under union, concatenation & kleene closure.

(2) CLOSED UNDER COMPLEMENT :-

Theorem:- If L is a regular language then $\bar{L}$ is also regular.

Proof:- Let $M = (Q, \Sigma, \delta, q_0, F)$ be the fA accepting L.

Then the dfa $\bar{M} = (Q, \Sigma, \delta, q_0, Q-F)$ accepts $\bar{L}$.

Let $w \in \Sigma^*$ if $\hat{\delta}(q_0, w) \in F$ then $w \in L$

if $\hat{\delta}(q_0, w) \in Q-F$ then $w \in \bar{L}$.

(3) CLOSED UNDER INTERSECTION :-

Theorem:- If L_1 & L_2 are two regular languages then $L_1 \cap L_2$ is also regular.

Proof:- Let $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ &

$M_2 = (P, \Sigma, \delta_2, p_0, F_2)$

Let $\hat{M}_1 = (\hat{Q}, \Sigma, \hat{\delta}, (p_0, q_0), \hat{F})$ whose set $\hat{Q} = Q \times P$ consists of pairs (q_i, p_j) and whose transition function $\hat{\delta}$ is such that –

$$\hat{\delta}((q_i, p_j), a) = (q_K, p_l)$$

whenever $\delta_1(q_i, a) = q_K$

$\delta_2(p_j, a) = p_l$

$\hat{F}$ is defined as the set of all (q_i, p_j) such that $q_i \in F_1$, and $p_j \in F_2$. Then it is simple to show that $w \in L_1 \cap L_2$ iff it is accepted by $\hat{M}$. So $L_1 \cap L_2$ is regular.

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

(4)- CLOSED UNDER DIFFERENCE :-

Theorem:- If L_1 and L_2 are regular languages then $(L_1 - L_2)$ is also regular.

Proof :- $L_1 - L_2 = L_1 \cap \overline{L_2}$

as we have previously proved that regular languages are closed under complement & intersection.

$\Rightarrow (L_1 - L_2)$ is regular.

(5)- CLOSED UNDER REVERSAL :-

Theorem:- If L is a regular language then L^R is also regular.

Proof :- If L is regular then there is a FA

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

Construct an equivalent FA with a single ~~final~~ Initial state

$q_f \in Q$ [initial state]

~~$F = \{q_f\}$~~ $q_n \in F$ [final state]

Do

Let $M^R = (Q, \Sigma, \delta_2, \{q_f\}, \{q_0\})$

where δ_2 is defined as follows:—

if $\delta_1(q_i, a) = q_j$

then

$\delta_2(q_j, a) = q_i$ i.e - reverse the direction of the transition.

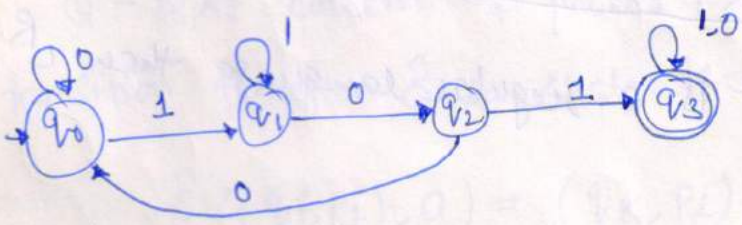
$\Rightarrow$ Initial state of the M is final state of M^R.

So if $w \in L$ then $w^R \in L^R$. So L^R is regular.

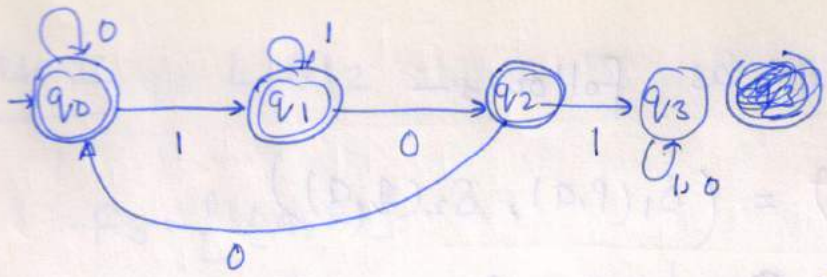
Example :- Construct an FA that does not accept the set of strings containing 101 as substring.

Solution:-

1)- Construct a FA that accept all strings containing 101 as substring.



2)- Since from the closure properties of regular languages if L is regular then $\bar{L}$ is also regular by changing the final state into non-final and non-final states to final states.



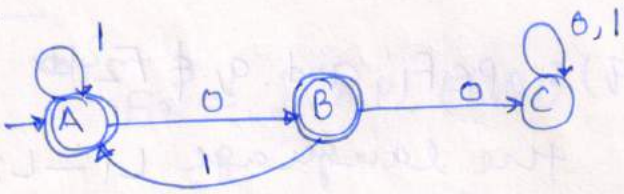
This dfa accepts all strings from Σ.

Example:- Let $L_1 = \{x \mid 00$ is not a substring of $x\}$

$L_2 = \{x \mid x$ ends with $01\}$

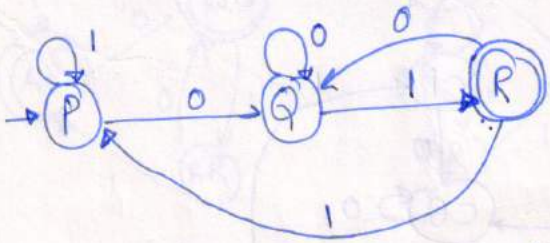
Find the DFA for the language $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$.

Solution:-



$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$

(DFA for L_1)



$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

Then $M_3 = (Q, \Sigma, \delta_3, q_0, F_3)$ - where $Q = Q_1 \times Q_2$

$q_0 = (q_1, q_2)$

Initial state (A,P).

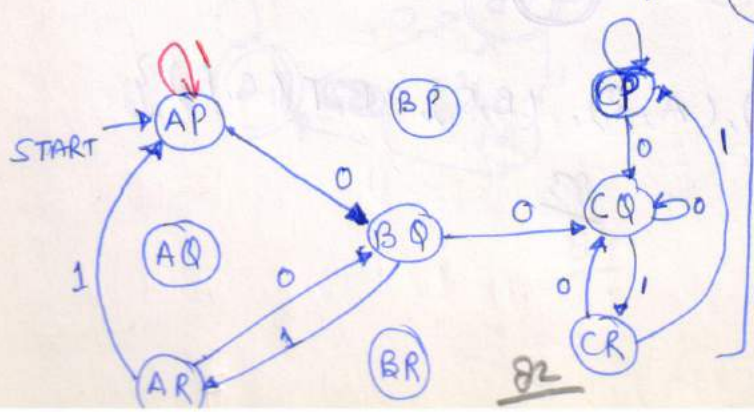
In order to draw the transition diagram we start with initial state (A,P).

$\delta_1(A, 0) = B$

$\delta_2(P, 0) = Q$

So $\delta_3((A,P), 0) = (B, Q)$.

Case-I - $L_1 \cup L_2$



δ_2

δ_3 is defined as follows:—

$$\delta_3((p,q),a) = (\delta_1(p,a), \delta_2(q,a))$$

for any $p \in Q_1$, $q \in Q_2$, $a \in \Sigma$.

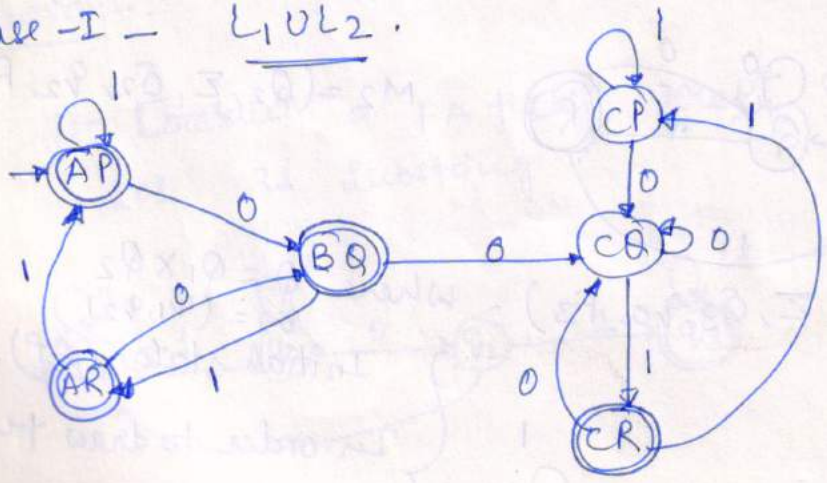
Then

1- IF $F_3 = \{(p,q) : p \in F_1$ or $q \in F_2\}$, M_3 accepts the language $L_1 \cup L_2$

2- If $F_3 = \{(p,q) : p \in F_1$ and $q \in F_2\}$ M_3 accepts the language $L_1 \cap L_2$

3- If $F_3 = \{(p,q) : p \in F_1$ and $q \notin F_2\}$ M_3 accepts the language $L_1 - L_2$.

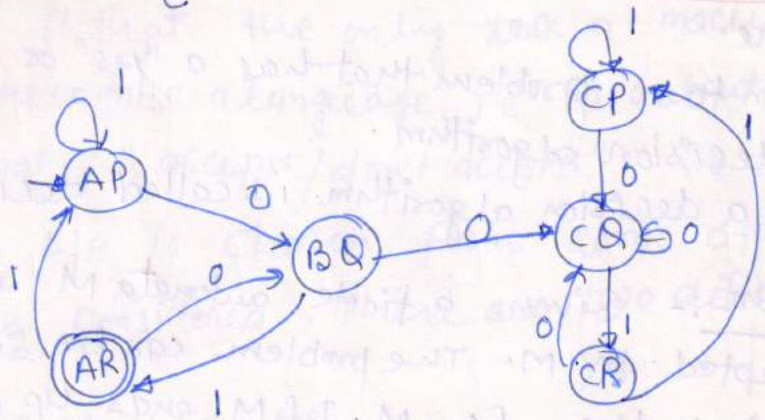
Case-I — $L_1 \cup L_2$.



$F_3 = \{(A,P), (A,R), (B,Q), (C,R)\}$

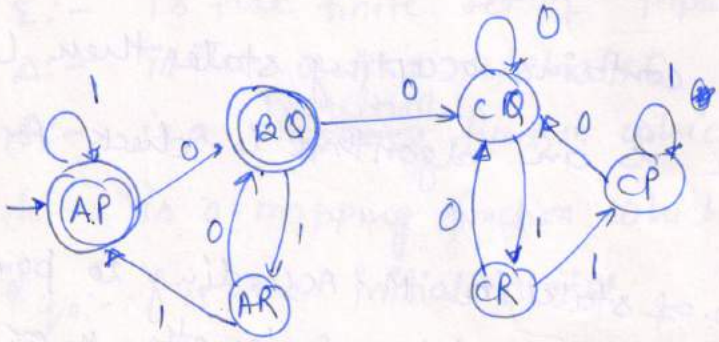
Case-II $L_1 \cap L_2$

$F_3 = \{(A,R)\}$.

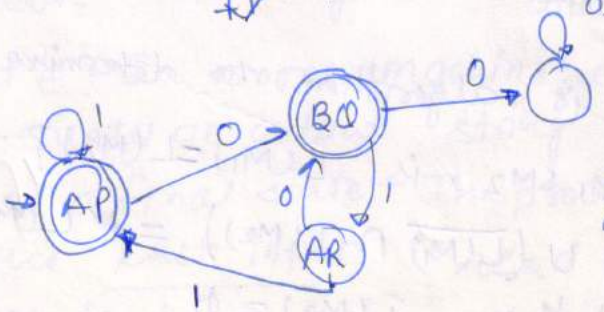


Case-III. $L_1 - L_2$

$F_3 = \{(A,P), (B,Q)\}$



⇓ OR



& ~~MFRO~~ (Minimized dfa).

Decision properties of regular languages

⇒ A problem is solvable if there is an algorithm that provides the answer in a finite no. of steps, no matter what the particular inputs are.

⇒ An effective solution to a problem that has a "Yes" or "No" answer is called a decision algorithm.

⇒ A problem that has a decision algorithm is called decidable.

1)- Membership problem:- Given a finite automata M and a string w, is w accepted by M. The problem can be solved by giving the string w to the FA, M. If M ends up in accepting state the answer is "Yes" otherwise the answer is "No".

2)- Emptiness:- There is a decision algorithm to find for a given finite automata M, is $L(M) = \phi$?

1)- If M contains final states then goto step 2.

2)- If Final states are not reachable then $L(M) = \phi$

3)- Else $L(M) \neq \phi$

4)- If M does not contains accepting states then $L(M) = \phi$

3)- Finiteness:- There is an algorithm to check for a given M, L(M) finite?.

1)- Let n be the no. of states in M. According to pumping lemma if M accepts even one string of length n or greater, then L(M) is infinite

4)- Equivalence:- There is algorithm to determine for given two Finite automata M_1 & M_2, is $L(M_1) = L(M_2)$?.

1). let $(L(M_1) \cap \overline{L(M_2)}) \cup (\overline{L(M_1)} \cap L(M_2)) = L(M_3)$.

2)- If $L(M_1) = L(M_2)$ then $L(M_3) = \phi$

3)- If $L(M_3) = \phi$ then $L(M_1) = L(M_2)$.

4)- Else $L(M_1) \neq L(M_2)$.

Finite automata With output

One limitation of the finite automata as we have discussed so far is that the only task a machine has done so far is to recognize a language i.e its output is limited to a binary signal : "accepts / don't accepts". So the model in which the o/p is chosen from some other alphabet have been considered. There are two different approaches-

1- Moore Machine :-

The moore machine was created by E.F moore (1956).
The moore machine is a collection of six tuples

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

Q :- is finite set of states

Σ :- is the finite set of input symbols

Δ :- is the output alphabet

δ :- is a ~~mapping~~ Transition function which maps $Q \times \Sigma$ to Q.

λ :- is a mapping function which maps from $\lambda : Q$ to Δ

q_0 :- is the initial state.

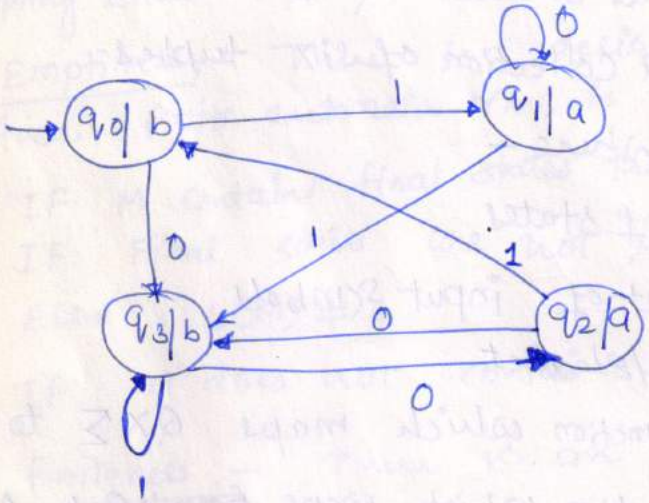
⇒ Any moore machine gives output $\lambda(q_0)$ in response to input ϵ. In moore machine, every possible input string creates an output string and there is no such thing as a final state. The processing is terminated when the last input is read and the last output character is produced.

06

Example:- Let us consider a moore machine

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

Current state	Next state input 0	Next state input 1	output
→ q_0	q_3	q_1	b
q_1	q_1	q_3	a
q_2	q_3	q_0	a
q_3	q_2	q_3	b



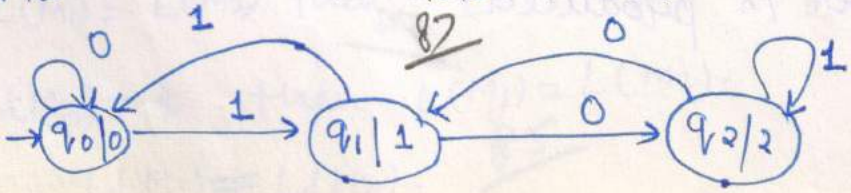
Let input string $w = 10101$

output:- baabab

⇒ As we always start with the start state ~~the first~~ q_0, which automatically produce the output character b.

ex:- (1)- ~~To~~ Moore machine to determine ~~red~~ residue mod 3.

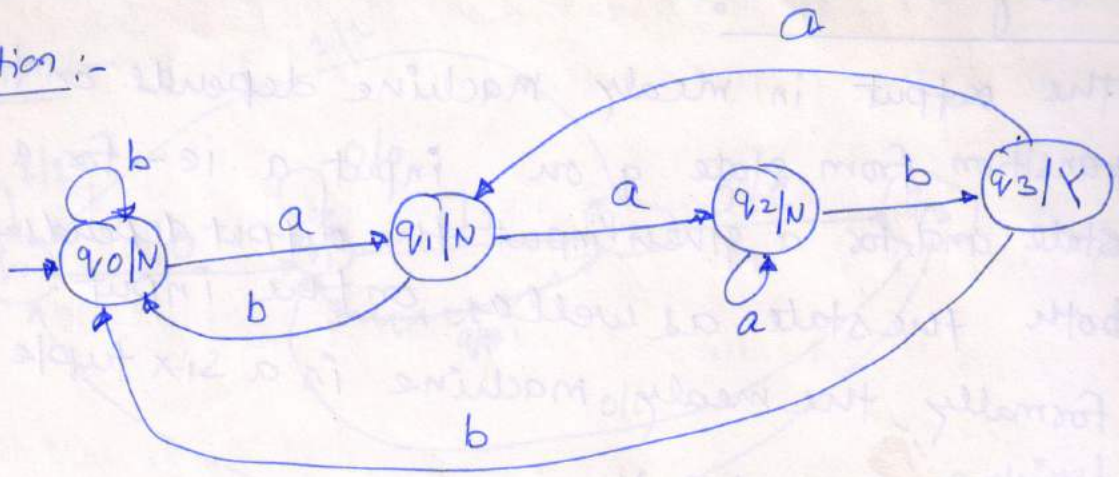
~~(2)~~-



82

Example :- Design a moore machine which counts the occurance of substring aab.

Solution :-

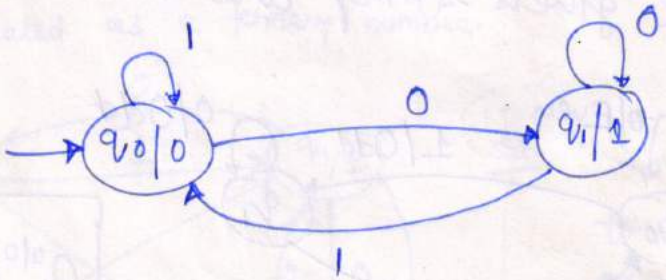


Let w = abaab

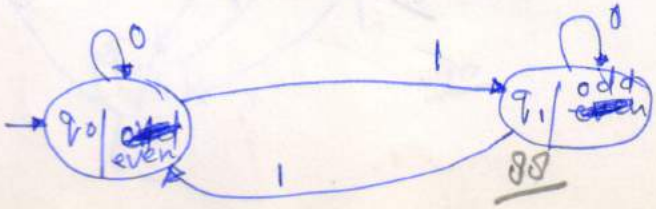
then output :- NNNNNY

Example :- Design a moore machine to get 1's complement of a given number.

Solution :-



Example :- Construct a moore machine which check whether no. of 1's in a given string are odd or even.



~~Example~~

Mealy Machine :-

The output in mealy machine depends on the transition from state q on input a. ie - for a given state and for a given input the output depends on both the state as well as on the input.

formally, the mealy machine is a six tuple.

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

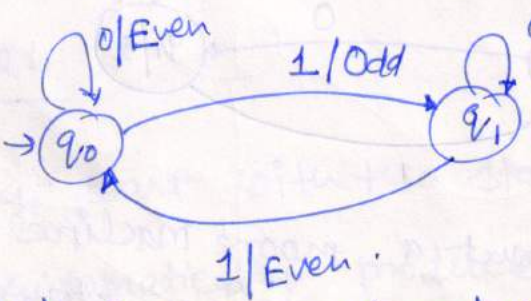
where $Q, \Sigma, \delta, \Delta, q_0$ are as in moore machine.

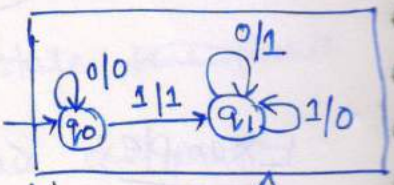
λ is output mapping function is defined as follows:-

$$\lambda : Q \times \Sigma \rightarrow \Delta$$

example :- Construct a Mealy machine which check wether no. of 1's in a given string are odd or even.

Solution :-



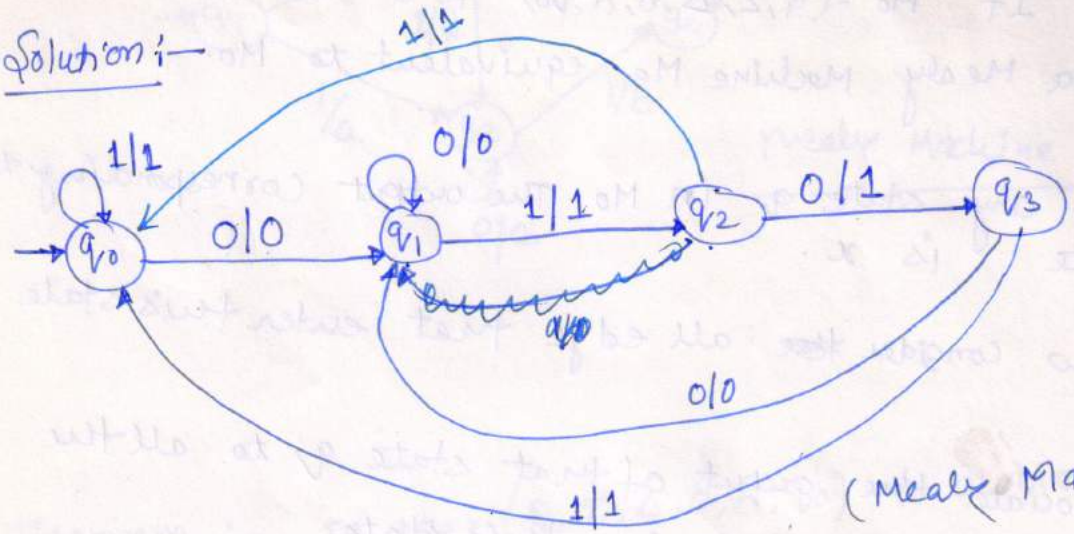


Q- ① Mealy Machine to find 2's complement of a binary No.

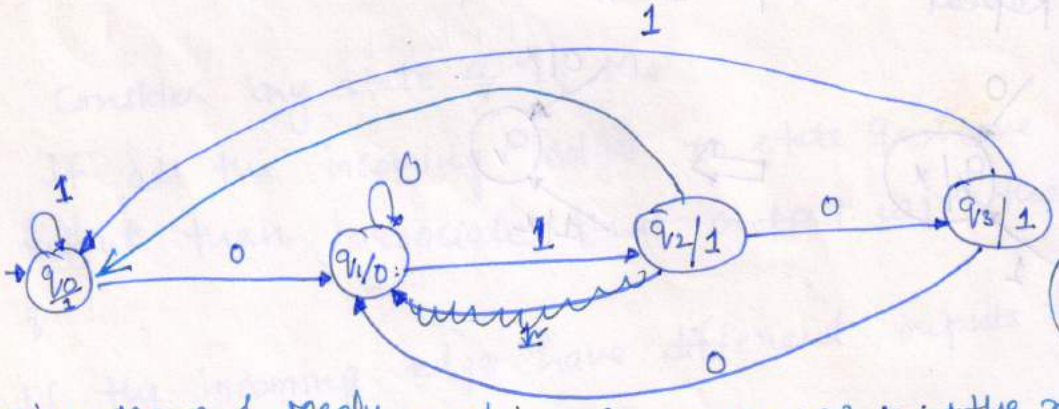
②

Example:- Design a Mealy and Moore machine which convert each occurance of substring 010 to 011.

Solution:-



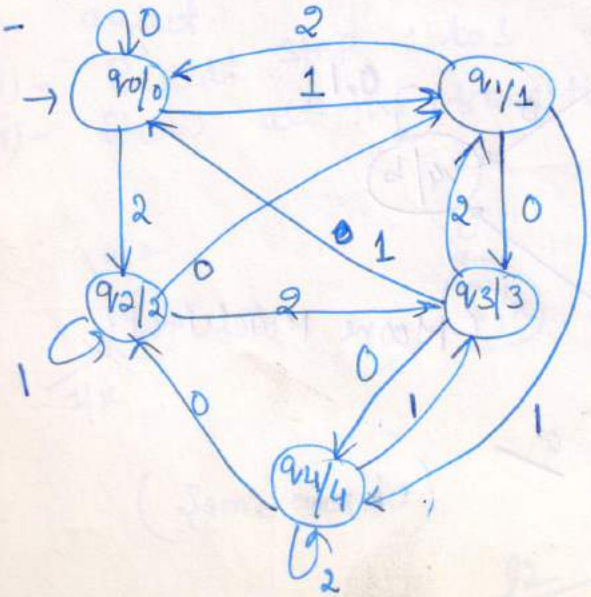
(Mealy Machine)



(Moore Machine)

ex- give Moore & Mealy machine for $\Sigma = \{0,1,2\}$ print the residue mods of input treated as a ternary number.

sol -



ex - Construct a Moore machine to determine residue mods for binary number.

90

Equivalence of Mealy & Moore Machine :-

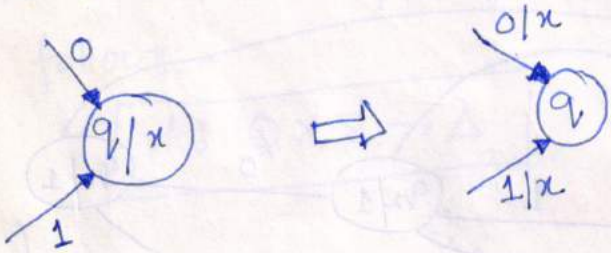
Theorem :- If $M_0 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is a Moore Machine, then there is a Mealy Machine M_e equivalent to M_0.

1)- Consider any state q in M_0. The output corresponding to this state is x.

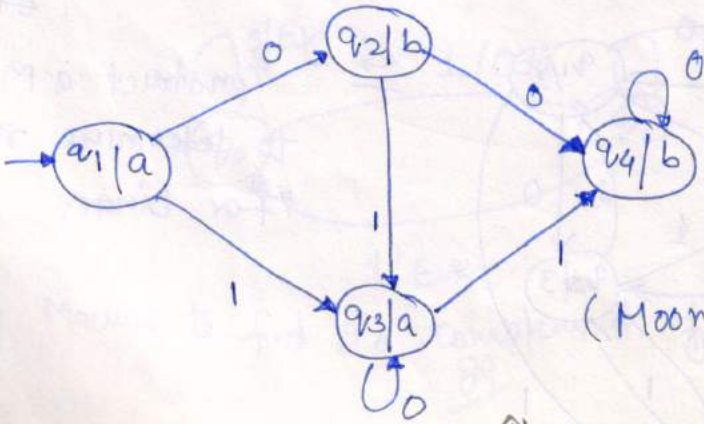
2) – Now consider all edge that enter this state q.

3) – Associate the ouput of that state q to all the edges that are entering this state.

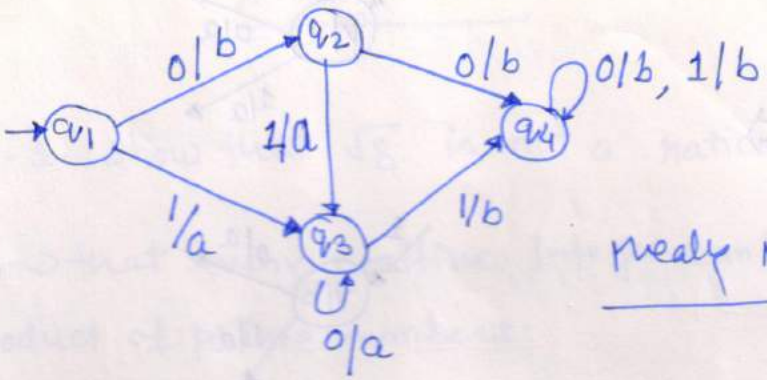
4)- Repeat step 1 to 3 for all states.



Example :-



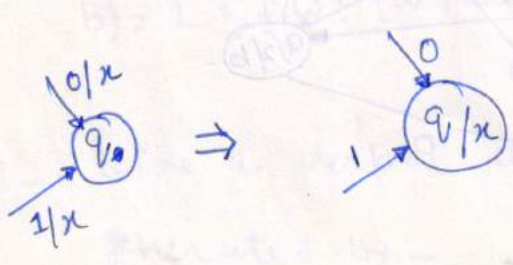
(Moore Machine).



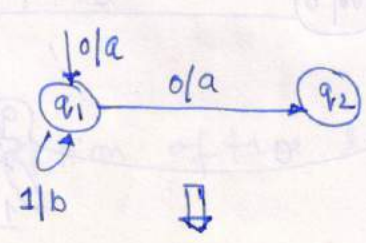
Mealy Machine

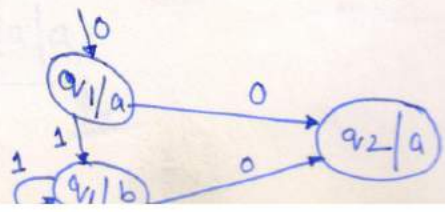
Theorem :- If $M_e = (Q, \Sigma, \Delta, \delta, d, q_0)$ is a Mealy Machine, then there is a Moore Machine M_0 equivalent to M_e.

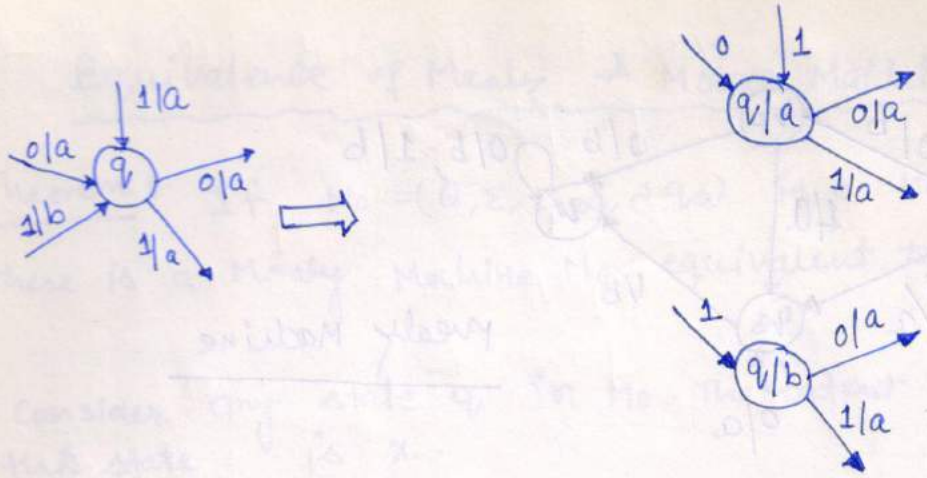
1)- Consider any state q in M_e.
2)- If all the incoming edges to state q have the same output then associate that output with the state q.
3)- If the incoming edges have different ouputs then make copies of that state (q) corresponding to each ouput.
4)- Repeat step 1 to 3 for all states.
5)- Show all the transitions as in Mealy Machine.



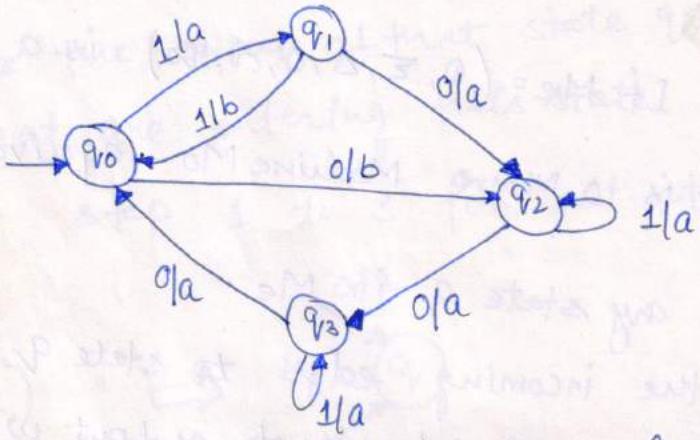
(Same output).



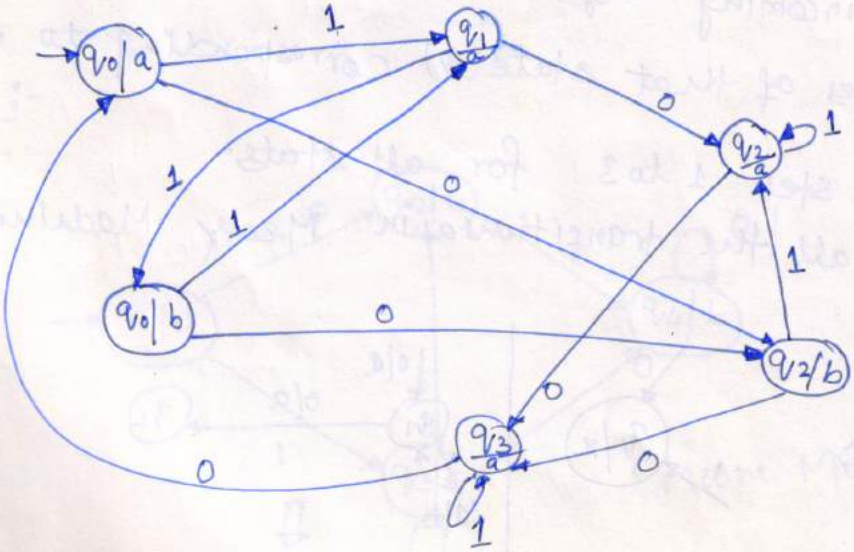




Example :-



State q_0 & q_2 have two different output. So copy state q_0 & q_2 for output a & b.



ASSIGNMENT - 1

Question :- 1. Show that $\sqrt{8}$ is not a rational number.

2) - Show that every positive integer can be expressed as the product of prime numbers.

3) - prove that $(L_1 L_2)^R = L_2^R L_1^R$

4) - Give a simple description of the language generated by the grammar with productions - $S \to aA \mid \epsilon$, $A \to bS$.

5) - Let $\Sigma = \{a, b\}$. For each of the following languages, find a grammar that generates it.

a) - $L_1 = \{a^n b^m : n \geq 0, m > n\}$

b) - $L_2 = \{a^n b^{2n} : n \geq 0\}$

c) - $L_3 = \{a^{n+2} b^n : n \geq 1\}$

d) - $L_4 = \{a^n b^{n-3} : n \geq 3\}$

6) - Let $\Sigma = \{a\}$. Find the grammar for the following languages-

a) - $L = \{w : |w| \bmod 3 = 0\}$

b) - $L = \{w : |w| \bmod 3 \neq |w| \bmod 2\}$

7) - Give a verbal description of the language generated by -

$S \to aSb \mid bSa \mid a$.

8)- Are the two grammars with respective productions

$$S \to aSb \mid ab \mid \epsilon$$

and

$$S \to aAb \mid ab$$
$$A \to aAb \mid \epsilon$$

equivalent?

9)- Design an accepter for integers in C.

10)- Show with example that if we change final state as non-final & non-final states as final states then the resulting dfa accepts $\bar{L}$.

11)- Find dfa's for the following languages on $\Sigma = \{a, b\}$

a)- $L = \{ w : n_a(w) \bmod 3 > 1 \}$

b)- $L = \{ w : w$ contains no runs of length less than four$\}$.

c)- Every aa is followed immediately by one b.

d)- Every substring of four symbols has at most two a's.

e)- The leftmost symbol differes from the rightmost one.

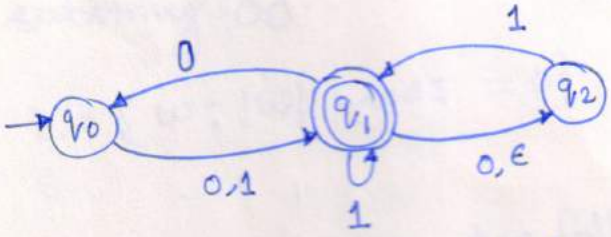
12)- Show that the language $L = \{ a^n : n \geq 4 \}$ is regular.

13)- Show that the language $L = \{a^n : n$ is either a multiple of three or a multiple of $5\}$ is regular.

14)- Show that if L is regular, so is $L \cup \{a\}$, for all $a \in \Sigma$.

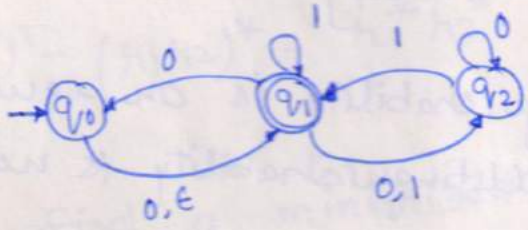
15)- Find an NFA with four states for
$L = \{a^n : n \geq 0\} \cup \{b^n a : n \geq 1\}$

16)-



Find a Minimized DFA for the above NFA with ϵ-moves.

17)- Convert the following NFA into an equivalent DFA-

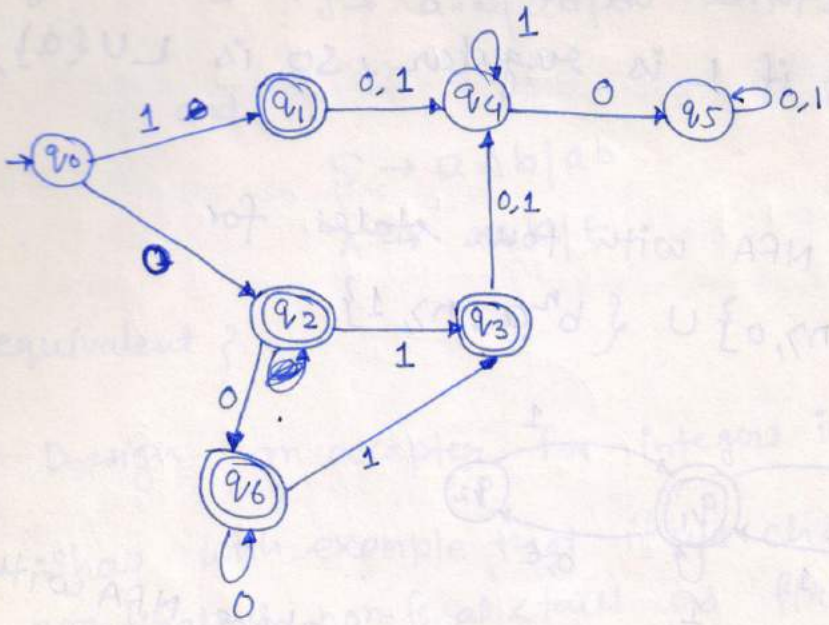


18)- prove that all finite languages are regular.

19)- Show that if L is regular, so is L^R.

20)- prove that for every NFA with an arbitrary no. of final states there is an equivalent NFA with only one state. Can we make a similar claim for dfa's.

21)- Find the minimal dfa for the following dfa -



22)- Find the Minimal dfa's for the following languages.

a)- $L = \{a^n : n \bmod 3 = 0\} \cup \{a^n : n \bmod 5 = 1\}$

b)- $L = \{a^n : n \geq 0, n \neq 3\}$

c)- $L = \{a^n : n \neq 2 \text{ and } n \neq 4\}$

23)- Show that indistinguishability is an equivalence relation but that distinguishability is not.

24)- Find a regular expression for the set $\{a^n b^m : n \geq 3$ m is even$\}$.

25)- Give regular expression for the following languages -

a)- $L_1 = \{a^n b^m : n \geq 4, m \leq 3\}$

b- $L_2 = \{a^n b^m : n < 4, m \leq 3\}$.

26)- Give a simple verbal description of the language $L((aa)^* b (aa)^* + a(aa)^* ba (aa)^*)$.

27)- Write regular expression for the following.

a)- all strings ending in 01.

b)- all strings containing an even number of 0's.

c)- all strings with atmost two occurence of the substring 00.

d)- $L = \{w : |w| \bmod 3 = 0\}$

28)- prove or disprove the following -

a)- $(r_1^*)^* = r_1^*$

b)- $r_1^* (r_1 + r_2)^* = (r_1 + r_2)^*$

c)- $(r_1 + r_2)^* = (r_1^* r_2^*)^*$

d)- $(r_1 r_2)^* = r_1^* r_2^*$

29)- Find a minimized dfa for the following regular expression-

a)- $r_1 = (aa^* (a+b))$

b)- $r_2 = (a+b)^* ab.$

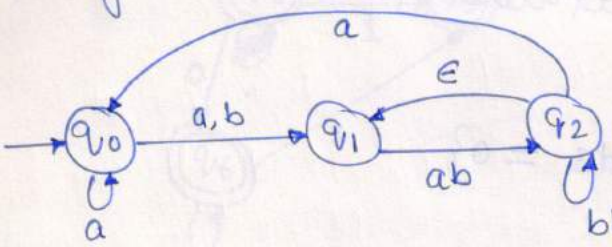
30)- Find a regular expression for $L = \{vwv : v, w \in \{a,b\}^*, |v| = 2\}$.

31) - Find dfa's that accepts the following languages -

a) - $L = L(ab^*a^*) \cup L((ab)^* ba)'$

b) - $L = L(ab^* a^*) \cap L(ab)^* ba)$

32) - Find an equivalent generalized transition graph with only two states -



33) - Construct a dfa that accepts the language generated by the grammar -

$S \rightarrow abA$

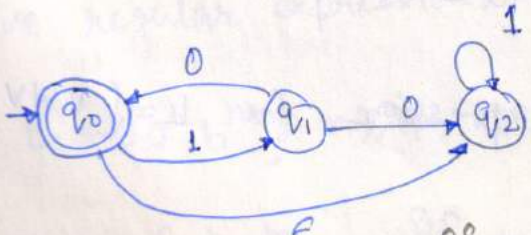
$A \rightarrow baB$

$B \rightarrow aA \mid bb$

34) - Find the grammar that generates the language $L(aa^* (ab + a)^*)$

35) - Show that any regular grammar G for which $L(G) \neq \phi$ must have atleast one production of the form $A \rightarrow x$, where $A \in V$ and $x \in T^*$.

36) - Find a left-linear grammar for the language accepted by the nfa below.



42)- Suppose we know that $L_1 \cup L_2$ is regular and that L_1 is finite. Can we conclude ~~that~~ this that L_2 is regular.

43)- If L is a regular language, prove that the language $\{uv : u \in L, v \in L^R\}$ is also regular.

44)- Let G_1 and G_2 be two regular grammars. Show ~~that~~ how one can derive regular grammars for the languages -

a)- $L(G_1) \cup L(G_2)$

b)- $L(G_1) \cdot L(G_2)$

c)- $L(G_1)^*$

45)- prove that the following languages are not regular-

a)- $L = \{a^n b^l c^k : k \geq n+l\}$

b)- $L = \{ww : w \in \{a,b\}^*\}$

c)- $L = \{w : n_a(w) \neq n_b(w)\}$

d)- $L = \{a^n b^l : n \leq l\}$

46)- prove that the language $L = \{a^n : n$ is a prime number$\}$ is not regular.

47)- Let L_1 & L_2 be regular languages. Is the language $L = \{w : w \in L_1, w^R \in L_2\}$ necessarily regular?

48)- Is the family of regular languages closed under infinite intersection?

49)- prove that $L = \{a^n : n = k^3$ for some $k \geq 0\}$ is not regular

50)- prove that $L = \{a^n : n = 2^k$ for some $k \geq 0\}$ is not regular.

UNIT→3

CONTEXT FREE GRAMMAR

formal language :- We refer the language in which strings are strictly formed by the rules are called "formal language".

formally, A grammar G is defined as a quadruple.

$G = (V, T, P, S)$ where

V = is a finite set of variables.
T = is a finite set of terminal symbols
P = is a " " " production or rules.
$S \in V$ is a start symbol.

⇒ A grammar G is context free if each production in grammar is of the form $A \to \alpha$

⇒ The language generated by a CFG is the set of strings of terminals that can be produced from the start symbol using the productions (rules). A language generated by a CFG is called a context free language.

$L = L(G)$

example :- context free grammar for arithmetic expressions—

$E \to (E)$
$E \to E+E$
$E \to E*E$
$E \to -(E)$
$E \to a$

Ambiguous context free grammar :-

A context free grammar G is ambiguous if there is atleast one string in L(G) having two or more distinct derivation trees (Both leftmost or both rightmost derivations)

example :- $S \to S+S \mid S-S \mid S*S \mid S/S \mid (S) \mid a$

let $\omega = a+(a*a)/a-a$

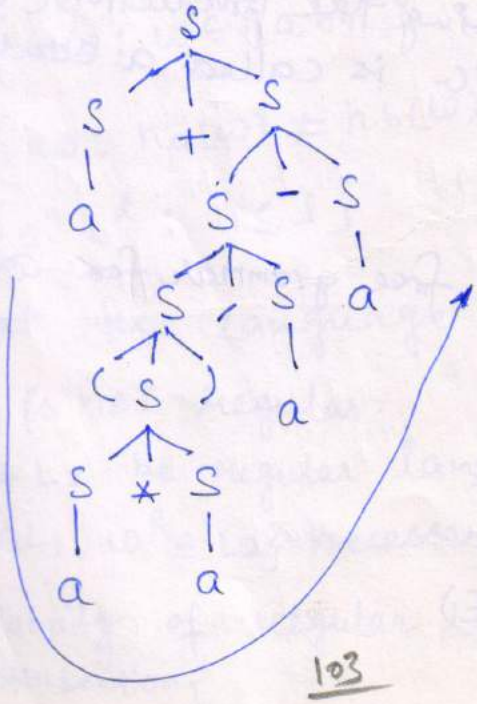
$S \Rightarrow S+\underline{S} \Rightarrow S+S-\underline{S} \Rightarrow S+\underline{S}-a$

$\Rightarrow S+S/\underline{S}-a \Rightarrow S+\underline{S}/a-a$

$\Rightarrow S+(S)/a-a \Rightarrow S+(S*S)/a-a$

$\Rightarrow S+(S*a)/a-a \Rightarrow S+(a*a)/a-a$

$\Rightarrow a+(a*a)/a-a$ (Right most derivation)

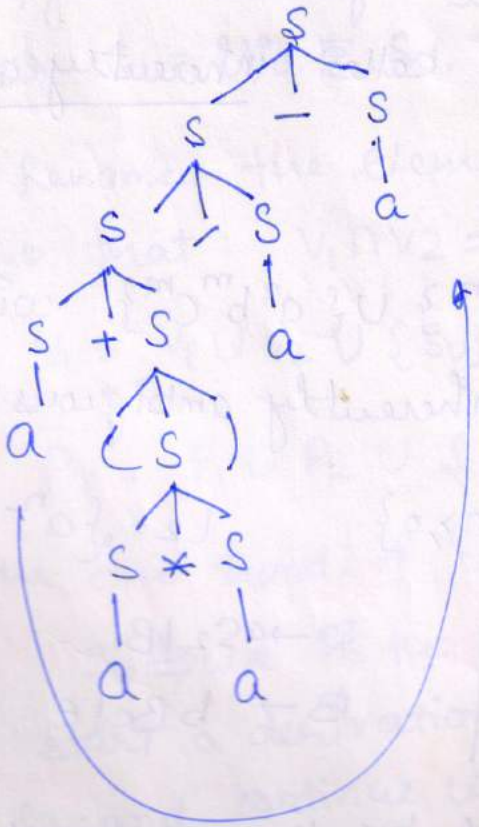


$\omega = a+(a*a)/a-a$

$S \Rightarrow S - \underline{S} \Rightarrow S - a \Rightarrow S|\underline{S} - a \Rightarrow \underline{S}|a - a$

$\Rightarrow S * \underline{S}|a - a \Rightarrow S * (\underline{S})|a - a \Rightarrow S + (S * \underline{S})|a - a$

$\Rightarrow S + (\underline{S} * a)|a - a \Rightarrow S + (a * a)|a - a \Rightarrow a + (a * a)|a - a$



$\omega = a + (a * a)|a - a$

The grammar G is ambiguous because there are two different rightmost derivation tree for the string $\omega = a + (a * a)|a - a$.

Inherent ambiguity :-

If L is a context free language for which there exists an unambiguous grammar, then L is said to be unambiguous.

⇒ If every grammar that generates L is ambiguous then the language is called inherently ambiguous.

example :- The language

$L = \{a^n b^n c^m\} \cup \{a^n b^m c^m\}$ with $n \neq m$ nonnegative, is an inherently ambiguous CFL.

$L_1 = \{a^n b^n c^m : m, n \geq 0\}$ $\quad$ $L_2 = \{a^n b^m c^m : n, m \geq 0\}$

$S_1 \rightarrow S_1 c \mid A$
$A \rightarrow aAb \mid \epsilon$

$S_2 \rightarrow aS_2 \mid B$
$B \rightarrow bBc \mid \epsilon$

Then L is generated by the combination of these two grammars with the additional production —

$$S \rightarrow S_1 \mid S_2$$

The grammar is ambiguous because $a^n b^n c^n$ has two distinct derivation, one starting with $S \Rightarrow S_1$ and other with $S \Rightarrow S_2$. So L is inherently ambiguous.

Q:- $L_1 = \{a^n b^n c^m d^m : n, m \geq 1\}$ $\quad$ $L_2 = \{a^n b^m c^m d^n : m, n \geq 1\}$

$S \rightarrow AB$
$S \rightarrow C$

$A \rightarrow aAb \mid ab$
$B \rightarrow cBd \mid cd$

$C \rightarrow aCd \mid aDd$
$D \rightarrow bDc \mid bc$

Theorem:- If L_1 and L_2 are context free languages then the languages $L_1 \cup L_2$, $L_1 \cdot L_2$, L_1^* are also context free languages.

Proof:- Let $G_1 = (V_1, \Sigma_1, S_1, P_1)$ and $G_2 = (V_2, \Sigma_2, S_2, P_2)$ generating L_1 and L_2, respectively. Then a grammar $G_U = (V_U, \Sigma, S_U, P_U)$ generates $L_1 \cup L_2$.

Step-1. Rename the elements of V_2 if necessary, so that $V_1 \cap V_2 = \phi$

2)- $V_U = V_1 \cup V_2 \cup \{S_U\}$

3)- $P_U = P_1 \cup P_2 \cup \{S_U \to S_1 \mid S_2\}$

On the one hand, if w is in either L_1 or L_2, then $S_U \overset{*}{\Rightarrow} w$ in the grammar G_U, because we can start a derivation with either $S_U \to S_1$ or $S_U \to S_2$ and continue with the derivation of w in G_1 or G_2. Therefore,

$$L_1 \cup L_2 \subseteq L(G_U)$$

$\Rightarrow$ A grammar $G_C = (V_C, \Sigma, S_C, P_C)$ generating $L_1 \cdot L_2$

step-1)- Rename the variables of V_2, so that

$V_1 \cap V_2 = \phi$

$V_c = V_1 \cup V_2 \cup \{S_c\}$

&

$P_c = P_1 \cup P_2 \cup \{S_c \to S_1 \cdot S_2\}$

If $x \in L_1 L_2$, then $x = x_1 x_2$ where $x_1 \in L_1$ & $x_2 \in L_2$.

$\Rightarrow$ A grammar $G^* = (V, \Sigma, S^*, P)$ generating L_1^*. Let

$V = V_1 \cup \{S^*\}$ where $S^* \notin V_1$.

$P = P_1 \cup \{S^* \to S S^* \mid \epsilon\}$

example :- 1)- Let $G = (\{S, A_1, A_2\}, \{a, b\}, P, S)$ where P consists of $S \to a A_1 A_2 a$, $A_1 \to ba A_1 A_2 b$, $A_2 \to A_1 ab$, $aA_1 \to baa$, $bA_2 b \to abab$. Test whether $\omega =$ baabbabaaabbaba is in $L(G)$.

2)- find parse Tree for the string abbcde, considering the productions. $S \to aAcBe$, $A \to Ab$, $A \to b$, $B \to d$.

3)- show that the CFG G with the following productions $S \to a \mid Sa \mid bSS \mid SSb \mid SbS$ is an ambiguous grammar.

4)- Construct grammar for the following languages.

1)- $L_1 = \{a^m b^m : m \geq 1\}$

2) $L_2 = \{b^n a^n : n \geq 1\}$

3)- $L_1 \cup L_2$

4)- $L_3 = \{a^l b^m c^n \mid l+m = n, l, m \geq 1\}$

$S \to aSc$
$S \to X$
$X \to bXc$
$X \to bc$

$a^l b^m c^{l+m}$
$a^l b^m c^m c^l$

A CFG equivalent to a regular expression :-

Let r is a regular expression, then there exists a context free grammar such that

$$L(G) = L(r)$$

example :- $r = (011+1)^*(01)^*$

The productions -

$A \to 011$
$A \to 1$
} generates the language $\{011, 1\}$.

$B \to BA \mid \epsilon$
$A \to 011 \mid 1$
} with B as a start symbol to generate the language $\{011, 1\}^*$

The production $D \to 01$ generating the language $\{01\}$.

$C \to DC \mid \epsilon$
$D \to 01$
} generates $\{01\}^*$.

So,

$S \to BC$
$B \to BA \mid \epsilon$
$C \to DC \mid \epsilon$
$D \to 01$
$A \to 011 \mid 1$
} will generate $(011+1)^*(01)^*$.

Ambiguous to Unambiguous

⇒ Ambiguity is normally a property of the grammar rather than the language. If a CFG is ambiguous, it is often possible and usually desirable to find an equivalent unambiguous CFG.

Let $S \longrightarrow S+S \mid S*S \mid (S) \mid a$ is an ambiguous grammar.

'*' should have higher precedence than '+', and a+a+a should mean (a+a)+a, not a+(a+a).

$$S \rightarrow S+T \mid T$$
$$T \rightarrow T*F \mid F$$
$$F \rightarrow (S) \mid a$$

(1) - factor :- A factor is an expression that cannot be broken apart by any adjacent operator, either a* or a+. The only factors are:
(a) - Identifiers - It is not possible to separate the letters of an identifier by attaching an operator.
(b) - Any parenthesized expression :- no matter what appears inside the parantheses. It is the purpose of parantheses to prevent what is inside from becoming the operand of any operator outside the parantheses.

(2). Term :- A term is an expression that can not be broken by + operator. A term may be a product of two or more factors.

(3). expression :- An expression will hence forth refer to any possible expression, including those that can be broken by either an adjacent * or an adjacent +. Thus an expression for our example is a sum of one or more terms.

$$E \rightarrow T \mid E+T$$
$$T \rightarrow F \mid T*F$$
$$F \rightarrow id \mid (E)$$

SIMPLIFICATION OF CONTEXT FREE GRAMMARS

1) - USELESS SYMBOLS

Remove all productions and symbols that can never take part in any derivation. such symbols are Called useless symbols and production involving any useless symbol is called useless production.

⇒ A grammar obtained by deleting useless symbols and productions is called reduced grammar.

⇒ A symbol A is useful if there is atleast one string $w \in L(G)$ such that $S \overset{*}{\Rightarrow} \alpha A \beta \overset{*}{\Rightarrow} w$.

Lemma 1:- If $G = (V, T, P, S)$ is a CFG with $L(G) \neq \phi$, we can find an equivalent grammar $G' = (V', T, P', S)$ such that each variable in G' derive some terminal string ie for each $A \in V'$, there is some $w \in T^*$ for which ~~$A \overset{*}{\Rightarrow} w$~~. $\boxed{S \overset{*}{\Rightarrow} \alpha A \beta \overset{*}{\Rightarrow} w}$

Proof:- 1) - Construction of V'.

a) - $Vold = \phi$

b) - $Vnew = \{ A \mid A \to w$, where $w \in T^* \}$

c) - While $Vold \neq Vnew$ repeat step (d) & (e)

d) - $Vold = Vnew$

e) - $Vnew = Vold \cup \{ A : A \to \alpha$ for some $\alpha \in (T \cup Vold)^* \}$

f) - $V' = Vnew$

g) - $P' = \{ A \to \alpha : A \in V'$ and $\alpha \in (V' \cup T)^* \}$

example :-

$S \to aA \mid bA \mid eC \mid Dd$

$A \to aDD \mid bC$

$B \to aB \mid eE$

$C \to e \mid C$

$D \to d$

$T = \{a, b, e\}$

Sol. $Vold = \phi$

$Vnew = \{C, D\}$. since $C \to e$, $D \to d$ ($Vnew = Vold$)

$Vold = Vnew = \{C, D\}$

$Vnew = \{C, D\} \cup \{A, C, D\} = \{A, C, D\}$ [$Vnew = Vold$]

$Vnew = \{A, C, D\} \cup \{S\} = \{S, A, C, D\}$ [$Vnew = Vold$]

$Vnew = \{S, A, C, D\} \cup \{\phi\}$

$= \{S, A, C, D\} \Rightarrow [Vold = Vnew]$.

So $V' = \{S, A, C, D\}$

$P' \Rightarrow S \to aA \mid bA \mid eC \mid Dd$

$A \to aDD \mid bC$

$C \to e \mid C$

$D \to d$

ex- $S \to AB \mid CA$
$B \to BC \mid AB$
$A \to a$
$C \to aB \mid b$
} $V_1 = \{S, A, C\}$

ex- $S \to aAa$
$A \to Sb \mid bCC \mid DaA$
$C \to abb \mid DD$
$E \to aC$
$D \to aDA$
$V' = \{S, A, C\}$

ex- $S \to AB$
$A \to a$
$B \to b$
$B \to C$
$E \to c$
} $V_1 = \{S, A, B, E\}$

$P_1 = S \to AB$
$A \to a$
$B \to b$
$E \to c$

Ans

Lemma-2 :- If $G=(V,T,P,S)$ is a CFG. Then we can find equivalent grammar $G'=(V',T',P',S)$ such that every symbol in $V' \cup T'$ appears in some sentential Form (ie for every X in $V' \cup T'$ there exists α such that $S \overset{*}{\underset{G'}{\Rightarrow}} \alpha$ and X is a symbol in the string α).

Proof :-

1)- Initially $V' = \{S\}$, $T' = \phi$

2)- Repeat while V' or T' change.

a)- for each $A \in V'$ if

$A \to \alpha_1 | \alpha_2 | \cdots | \alpha_n$ then add all variables of $\alpha_1, \alpha_2 \cdots \alpha_n$ to the set V'

and all terminals of $\alpha_1, \alpha_2 \cdots \alpha_n$ to T'

3). Construct P' as a set of productions of P containing only symbols of $(V' \cup T')$. Clearly all the symbols in V' and T' must occur in some string derivable from S.

ex :-

$S \to aAB | bA$

$A \to aAa | bCa$

$B \to Ca | Ac$

$C \to Ab | cB$

$D \to aAB | dB$ ✗

Step-

1- $V' = \{S\}$, $T' = \phi$

2- $V' = \{S, A, B\}$ $T' = \{a, b\}$

3- $V' = \{S, A, B, C\}$ $T' = \{a, b, c\}$

4- $V' = \{S, A, B, C\}$ $T' = \{a, b, c\}$

ex- $S \to aAa \mid bBb \mid cCc$
$A \to bS \mid aBC \mid AbD$
$B \to bC$
$C \to aCc \mid ac$
$D \to dD \mid bD$
$E \to ab \mid a$

} eliminate useless symbols.

Elimination of ϵ-production

Nullable Variable:-

1)- Any variable A for which there is a production $A \to \epsilon$; is nullable.

2)- if $A \to B_1 B_2 \cdots B_n$ is a production and all B_i's are nullable then, A is nullable.

Theorem:- If $G = (V, T, P, S)$ be any CFG with ϵ-production then there is a grammar G' that has no ϵ-productions and $L(G') = L(G) - \{\epsilon\}$.

proof:- To construct G'

1)- Find all nullable variables.

2)- for every production $A \to X_1 X_2 \cdots X_n$ in P, add to P' every production that can be obtained by deleting one or more nullable X_i's in all possible combination. If all X_i's are nullable then do not delet all.

3)- Do not add ϵ-productions into P'.

Elimination of unit production :-

Any production of the form $A \to B$, where $A, B \in V$ is called a unit production.

Theorem :- Let G be any CFG without ϵ-production then there exists G' with no unit productions and $\boxed{L(G') = L(G)}$.

Proof :- Let $G' = (V, T, P, S)$ is a given CFG. Let us Construct $G' = (V, T, P', S)$ as follows:-

1) - Add all non unit production of P into P'.

2) - for every pair (A, B) where B is A-derivable ($A \to B$) and every non unit production.

$$B \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$$

add the production -

$$A \to \alpha_1 \mid \alpha_2 \cdots \alpha_n.$$

3) - Do not add unit productions in P'.

ex :-

$S \to A \mid bc$

$A \to B \mid ca$

$B \to S \mid ab$

$P' =$ $S \to bc$

$A \to ca$

$B \to ab$

A-derivable = {B, S}

B-derivable = {S, A}

S-derivable = {A, B}.

for Pair (A,B) → Non-unit productions of variable B is added to variable A.

ie- $B \to ab$ is added as $A \to ab$

for pair (A,S) → $S \to bc \Rightarrow A \to bc$

for pair (B,S) → $S \to bc \Rightarrow B \to bc$

for pair (B,A) → $A \to ca \Rightarrow B \to ca$

for (S,A) → $A \to ca \Rightarrow S \to ca$

for pair (S,B) - $B \to ab \Rightarrow S \to ab$

$S \to bc \mid ca \mid ab$
$A \to ca \mid bc \mid ab$
$B \to ab \mid ca \mid bc$

example- 1)- $S \to AB \mid CA$, $B \to BC \mid AB$, $A \to a$, $C \to aB \mid b$ } find a reduced grammar.

2)- $S \to aAa$, $A \to Sb \mid bCC \mid DaA$, $C \to abb \mid DD$, $E \to aC$, $D \to aDA$ } find a reduce grammar.

3)- $S \to aS \mid AB$, $A \to \epsilon$, $B \to \epsilon$, $D \to b$ } Remove ϵ-production & then unit productions.

4)- $S \to aA \mid aBB$, $A \to aaA \mid \epsilon$, $B \to bB \mid bbC$, $C \to B$

[illegible] unit production, useless production.

NORMAL FORMS for CFG

When the productions of G satisfy certain restrictions then the grammar G is said to be in a "Normal form".

1)- CHOMSKY NORMAL FORM :-(CNF)-

A grammar G is in CNF if every production are of the form $\boxed{A \to BC \text{ or } A \to a}$

Where A, B, C are variables and a is a terminal.

procedure:- Any CFG without ϵ-production is generated by a grammar which is in CNF.

1) - Eliminate all ϵ-productions if any.
2) - Eliminate all unit production if any.
3) - convert all production into the form-

$A \to B_1 B_2 \cdots B_K$ where $K \geq 2$ \$

$B_1, B_2 \cdots B_K$ are variables.

OR

of form $A \to a$ where a is terminal.

4) - If $A \to X_1 X_2 \cdots X_n$ where $n \geq 2$, then introduce new variables $Y_1, Y_2 \cdots Y_{n-2}$ and replace $A \to X_1 X_2 \cdots X_n$ by the set of productions

$$\boxed{\begin{aligned} A &\to X_1 Y_1 \\ Y_1 &\to X_2 Y_2 \\ Y_2 &\to X_3 Y_3 \\ &\vdots \\ Y_{n-2} &\to X_{n-1} X_n \end{aligned}}$$

example :- $S \to AACD$

$A \to aAb \mid \epsilon$

$C \to aC \mid a$

$D \to aDa \mid bDb \mid \epsilon$

Solution :- 1) - eliminate ϵ-productions

Nullable variables = {A, D}

So productions are -

$S \to AACD \mid ACD \mid CD \mid AAC \mid AC \mid C$

$A \to aAb \mid ab$

$C \to aC \mid a$

$D \to aDa \mid aa \mid bDb \mid bb$

2) - eliminate unit productions -

$S \to C$ is the only unit production.

So add production $C \to aC \mid a$ to S.

ie $S \to AACD \mid ACD \mid CD \mid AAC \mid AC \mid ac \mid a$

3) - $S \to AACD \mid ACD \mid CD \mid AAC \mid AC \mid X_aC \mid a$

$X_a \to a$

$A \to X_aAX_b \mid X_aX_b$

$X_b \to b$

$C \to X_aC \mid a$

$D \to X_aDX_a \mid X_aX_a \mid X_bDX_b \mid X_bX_b$

$S \to AACD$

$S \to A T_1$

$T_1 \to A T_2$

$T_2 \to CD$

$S \to ACD$

$S \to A T_2$

$S \to AAC \longrightarrow$ $S \to A T_3$, $T_3 \to AC$

$A \to X_a A X_b \rightsquigarrow$ $A \to X_a T_4$, $T_4 \to A X_b$

$D \to X_a D X_a \longrightarrow$ $D \to X_a T_5$, $T_5 \to D X_a$

$D \to X_b D X_b \rightsquigarrow$ $D \to X_b T_6$, $T_6 \to D X_b$

$S \to CD \mid AC \mid X_a C \mid a$ are in CNF.

Example:- Convert the following grammar into CNF.

$S \to AaA \mid cA \mid BaB$

$A \to aaBa \mid cDA \mid aa \mid Dc$

$B \to bB \mid bAB \mid bb \mid aS$

$C \to Ca \mid bC \mid D$

$D \to bD \mid \in$

Ex - Simplify the following grammar by eliminating useless symbols and then convert it into CNF -

$S \to a \mid aA \mid B \mid C$

$A \to aB \mid \in$

$B \to Aa$

$C \to eCD$, $D \to dd$

example :- Convert the following grammar into CNF-

$S \to AABC$
$A \to bAa \mid \epsilon$
$B \to aB \mid b$
$C \to aCa \mid bCb \mid \epsilon$

Solution :- 1) - eliminate ϵ-production

Nullable variables = { A, C }

$S \to AABC \mid ABC \mid BC \mid AB \mid B \mid AAB$
$A \to bAa \mid ba$
$B \to aB \mid b$
$C \to aCa \mid bCb \mid aa \mid bb$

2)- Remove Unit production $S \to B$.

$S \to AABC \mid ABC \mid BC \mid AB \mid AAB \mid aB \mid b$
$A \to bAa \mid ba$
$B \to aB \mid b$
$C \to aCa \mid bCb \mid aa \mid bb$

3)- $S \to AABC \mid ABC \mid BC \mid AB \mid AAB \mid X_aB \mid b$
$A \to X_bAX_a \mid X_bX_a$
$B \to X_aB \mid b$
$C \to X_aCX_a \mid X_bCX_b \mid X_aX_a \mid X_bX_b$
$X_a \to a$
$X_b \to b$

The production-

$S \to AABC$ is converted to -

$S \to ABC$ is -

$S \to AT_1$
$T_1 \to AT_2$
$T_2 \to BC$

$S \to AT_3$
$T_3 \to BC$

$S \to AAB$ is -

$S \to AT_4$
$T_4 \to AB$

$C \to X_aCX_a$ is -

$C \to X_aC_1$
$C_1 \to CX_a$

$A \to X_aAX_b$ is replaced by -

$A \to X_aT_5$
$T_5 \to AX_b$.

$C \to X_bCX_b$ is -

$C \to X_bT_6$
$T_6 \to CX_b$

and rest of the production are in CNF..

Ex- 2000-01

$S \to bA \mid aB$
$A \to bAA \mid aS \mid a$
$B \to aBB \mid bS \mid b$

} find an equivalent grammar in CNF.

GREIBACH NORMAL FORM :-(GNF)

A context free grammar is said to be in GNF if all productions have the form $\boxed{A \to a\alpha}$ where A is a variable, a is a terminal, α is the string of only variables. ie $\alpha \in V^*$

for example :- $S \to aAAB$, $A \to a \mid aB$, $B \to b$, $C \to cD$, $D \to b$ } is in GNF.

but the production $A \to BaA$, $B \to aBa$ } are not in GNF

1)- Substitution Rule :-

If P contains a production of form

$A \to \alpha_1 B \alpha_2$ and variable B has production.

$B \to aB \mid a$, then Remove production $A \to \alpha_1 B \alpha_2$ and add the following prod^n —

$$\boxed{A \to \alpha_1 a B \alpha_2 \mid \alpha_1 a \alpha_2}$$

Ex- Let $G = (\{S, A\}, \{a, b\}, P, S)$ be grammar with production — $S \to aA \mid abS \mid abA$, $A \to aS \mid b$

using substitution rule for Variable A —

$S \to aaS \mid ab \mid abS \mid abaS \mid abb$

$A \to aS \mid b$ is useless.

2)- Left recursion:- A grammar G is said to be left recursive if it has a variable A such that there is a derivation $A \overset{*}{\Rightarrow} A\alpha$ for some α ie. A is the leftmost symbol of the right hand side

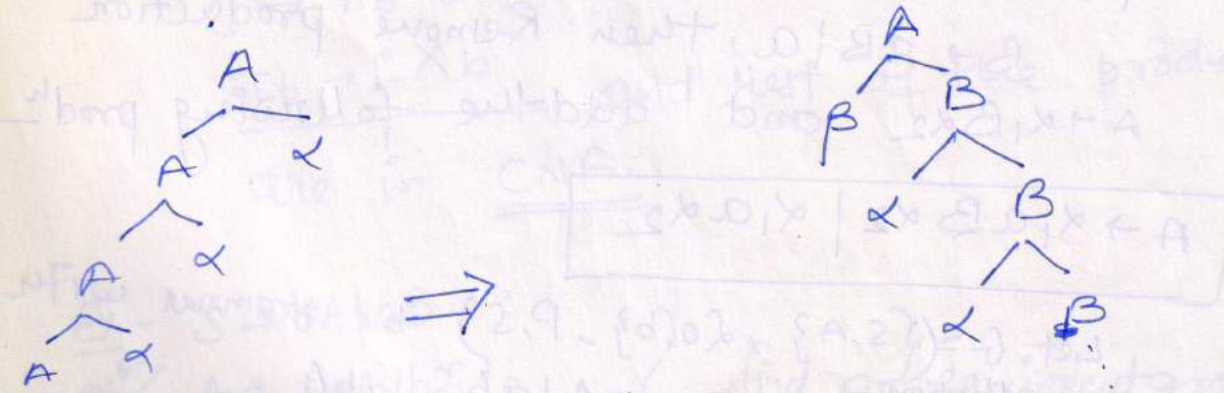
Theorem:- Let $G=(V,T,P,S)$ be a CFG with a left recursive pair of productions $A \to A\alpha \mid \beta$ where β does not begin with an A, then there is a CFG $G'=(V \cup \{B\}, T, P', S)$ where P' is obtained by deleting left recursive production and adding

$$A \to \beta B$$
$$B \to \alpha B \mid \epsilon$$

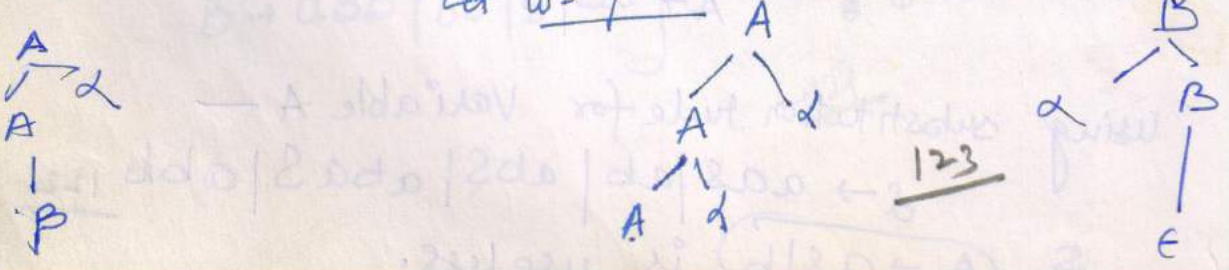
OR

$$A \to \beta B \mid \beta$$
$$B \to \alpha B \mid \alpha$$

* Left recursive grammar creates problem in top-down parsing.
A left recursive grammar can cause a recursive descent parser into infinite loop.



Let $w = \beta\alpha\alpha$



123

In general if

$A \to A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_m$

$A \to \beta_1 \mid \beta_2 \dots \mid \beta_n$ are productions

~~[illegible]~~ no β_i's begins with A, then

$$A \to \beta_1 B \mid \beta_2 B \dots \mid \beta_n B$$
$$B \to \alpha_1 B \mid \alpha_2 B \mid \dots \mid \alpha_m B \mid \epsilon$$

ex:- Eliminate left recursion from productions-

$A \to AC \mid AB \mid a \mid b \mid aa$

$B \to b$

$C \to c$

Sol:- $A \to \underbrace{AC}_{\alpha_1} \mid \underbrace{AB}_{\alpha_2} \mid \underbrace{a}_{\beta_1} \mid \underbrace{b}_{\beta_2} \mid \underbrace{aa}_{\beta_3}$

$\Downarrow$

$A \to aB' \mid bB' \mid aaB'$

$B' \to CB' \mid BB' \mid \epsilon$

$$\Rightarrow \boxed{\begin{aligned} A &\to aB' \mid bB' \mid aaB' \mid a \mid b \mid aa \\ B' &\to CB' \mid B \mid C \mid BB' \end{aligned}}$$

~~A → Aα~~
~~Aα β~~

Reduction to GNF

Theorem:- Every ~~CFG~~ context free language L without ~~ε-p~~ ε-Prod^n can be generated by a grammar for which every productions are in GNF i.e-

$A \to a\alpha$ where $\alpha \in V^*$

Step- 1):- Grammar must be in CNF, if not then convert it into CNF.

2) - Write G as $(\{A_1, A_2 \ldots A_n\}, T, P, A_1)$ by renaming the variables with A_1 as start symbol.

3) - Now try to modify the grammar in the form

$A_i \to a\alpha$ or $A_i \to A_j\alpha$ where $j > i$

by applying substitution rule.

4) - Apply substitution rule & left recursion method to convert the grammar into required form i.e - $A \to a\alpha$.

Example:-

$S \to AB$
$A \to BS \mid a$
$B \to SA \mid b$

} Convert the grammar into GNF.

Solution:- 1) - Grammar is already in CNF.

2) - Rename the variables

$A_1 \rightarrow A_2 A_3$ ✓

$A_2 \rightarrow A_3 A_1 \mid a$ ✓

$A_3 \rightarrow A_1 A_2 \mid b$

3) - The production $A_3 \rightarrow A_1 A_2$ is not in the form $A_i \rightarrow A_j \alpha$ where $j > i$. Apply substitution rule for A_1 -

$A_3 \rightarrow A_2 A_3 A_2$, again apply substitution rule for A_2 -

$A_3 \rightarrow \overbrace{A_3}^{A} \overbrace{A_1 A_3 A_2}^{\alpha} \mid \overbrace{a A_3 A_2}^{\beta_1} \mid \overbrace{b}^{\beta_2}$

eliminate left recursion -

$A_3 \rightarrow a A_3 A_2 B_3 \mid a A_3 A_2 \mid b B_3 \mid b$ ✓ GNF

$B_3 \rightarrow A_1 A_3 A_2 B_3 \mid A_1 A_3 A_2$

put A_3 into A_2.

$A_2 \rightarrow a A_3 A_2 B_3 A_1 \mid a A_3 A_2 A_1 \mid b B_3 A_1 \mid b A_1 \mid a$

Now put A_2 in A_1 -

$A_1 \rightarrow a A_3 A_2 B_3 A_1 A_3 \mid a A_3 A_2 A_1 A_3 \mid b B_3 A_1 A_3 \mid b A_1 A_3 \mid a A_3$

$A_2 \rightarrow a A_3 A_2 B_3 A_1 \mid a A_3 A_2 A_1 \mid b B_3 A_1 \mid b A_1 \mid a$

$A_3 \rightarrow a A_3 A_2 B_3 \mid a A_3 A_2 \mid b B_3 \mid b$

$B_3 \rightarrow a A_3 A_2 B_3 A_1 A_3 A_3 A_2 B_3 \mid a A_3 A_2 A_1 A_3 A_3 A_2 B_3 \mid$

$b B_3 A_1 A_3 A_3 A_2 B_3 \mid b A_1 A_3 A_3 A_2 B_3 \mid a A_3 A_3 A_2 B_3$

$B_3 \rightarrow a A_3 A_2 B_3 A_1 A_3 A_3 A_2 \mid a A_3 A_2 A_1 A_3 A_3 A_2 \mid b B_3 A_1 A_3 A_3 A_2$

CLOSURE PROPERTIES OF CONTEXT FREE LANGUAGES

Theorem:- If L_1 & L_2 are context free languages then $L_1 \cup L_2$, $L_1 L_2$, L_1^* are also context free languages i.e. context free languages are closed under union, concatenation, & kleene star.

proof:- for $L_1 \cup L_2$

$$G_U = (\{V_1 \cup V_2 \cup \{S_u\}\}, T, \{P_1 \cup P_2 \cup \{S_U \to S_1 | S_2\}\}, S_U)$$

for $L_1 \cdot L_2$

$$G_C = (V_1 \cup V_2 \cup \{S_C\}, T, P_1 \cup P_2 \cup \{S_C \to S_1 S_2\}, S_C)$$

for L_1^* -

$$G_S = (V_1 \cup \{S_S\}, T, P_1 \cup \{S_S \to S_1 S_S | \in\}, S_S\}$$

where

$G_1 = (V_1, T, P_1, S_1)$

$G_2 = (V_2, T, P_2, S_2)$

---

theorem:- If L_1 is a CFL then $\overline{L}$ (complement of L) is not a CFL. ie CFLs are not closed under complementation.

proof:-

By Demorgan's law we know that -

$$L_1 \cap L_2 = \overline{(\overline{L_1} \cup \overline{L_2})}$$

If CFL's are closed under complementation the $(\overline{L_1 \cup L_2})$ could be a CFL. But we will show that $L_1 \cap L_2$ is not necessarily a CFL. Therefore the CFL's are not closed under complementation.

Theorem:- If L_1 & L_2 are two context free languages then $L_1 \cap L_2$ is not a context free language.

proof:- Consider two languages-

$$L_1 = \{a^n b^n c^m : n \geq 0 \text{ and } m \geq 0\}$$

$$L_2 = \{a^n b^m c^m : m \geq 0 \text{ and } n \geq 0\}$$

Let $G_1 \Rightarrow$

$S' \rightarrow S_1 S_2$
$S_1 \rightarrow aS_1b \mid ab$
$S_2 \rightarrow cS_2 \mid c$

$G_2 \Rightarrow$

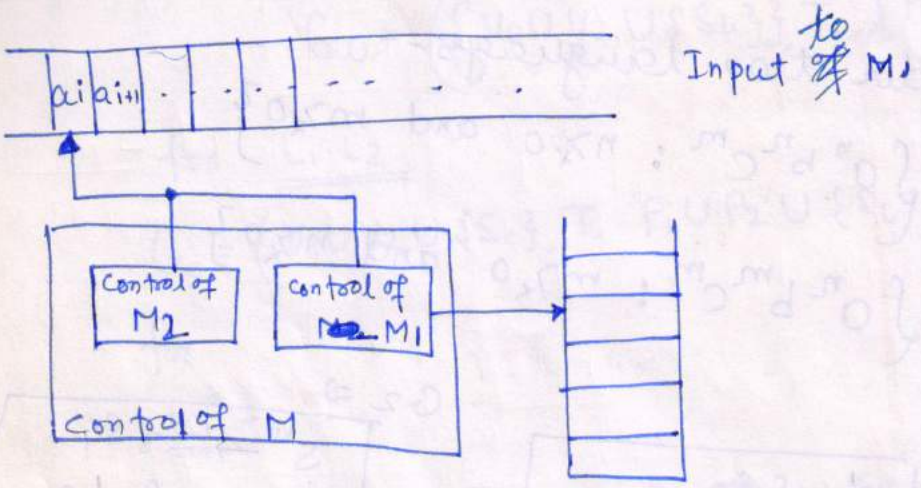
$S'' \rightarrow S_3 S_4$
$S_3 \rightarrow aS_3 \mid a$
$S_4 \rightarrow bS_4c \mid bc$

But $L_1 \cap L_2 = \{a^n b^n c^n : n \geq 0\}$ but with the help of pumping lemma we can proof that $\{a^n b^n c^n : n \geq 0\}$ is not a context free language. Thus $L_1 \cap L_2$ is not a context free language.

Theorem:- If L_1 is a CFL and L_2 is a regular language then $L_1 \cap L_2$ is a CFL.

Proof:- Let L_1 is accepted by PDA $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_1, Z_1, F_1)$ and L_2 is accepted by an F.A $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

We construct a PDA M for $L_1 \cap L_2$ by running M_1 & M_2 in parallel:



Here $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

and $Q = Q_1 \times Q_2$

$q_0 = (q_1, q_2)$

$F = F_1 \times F_2$

for $p \in Q_1$, $q \in Q_2$, $Z \in \Gamma$ and for any input $a \in \Sigma$

1)- $\delta((p,q), a, Z) = \{((p', q'), \alpha)\}$ such that

$\delta_1(p, a, Z) = \{(p', \alpha)\}$ & $\delta_2(q, a) = q'$

2)- $\delta((p,q), \epsilon, Z_0) = \{((p', q), \alpha)\}$ such that

$\delta_1(p, \epsilon, Z_0) = \{(p', \alpha)\}$ and $\delta_2(q, \epsilon) = q$

No change in state for ε-transition

ex-1) Show that the language $L = \{a^n b^n : n \geq 0, n \text{ is not a multiple of } 2\}$ is a context free language.

2) Show that $L = \{w : n_a(w) = n_b(w)$ and w does not contain abb as substring$\}$

Pumping Lemma for Context-free Languages

Theorem:- Let L be a context free language. Then there is a constant m depending only on L such that if z is in L and $|z| \geq m$, then we may write $z = uvwxy$ such that

1) - $|vx| \geq 1$

2) - $|vwx| \leq m$

and for all $i \geq 0$ uv^iwx^iy is in L.

Application of pumping Lemma:-

The pumping lemma can be used to prove a variety of languages not to be context free languages.

ex:- Consider the language $L = \{a^i b^i c^i : i \geq 1\}$

1) - Let us suppose L is context free.

2) - Let m be the constant given by pumping lemma.

3) - Consider $z = a^m b^m c^m$

$|z| = 3m$

4) - Write $z = a^m b^m c^m$ as $z = uvwxy$ so as to satisfy the conditions of the pumping lemma. vx does not contain instances of a's & c's because rightmost a is (m+1) position away from the leftmost c.

Case I. if v & x consists of a's only. Then uwy (the string $uv^i wx^i y$ with $i=0$) has m b's and m c's but fewer than m a's, since $|vx| \geq 1$. Thus uwy is not of the form $a^j b^j c^j$. So the language $L = \{a^n b^n c^n : n \geq 1\}$ is not context free.

PROOF of PUMPING LEMMA

Lemma - Let G be a context free grammar in CNF and T be a derivation Tree in G. If the length of longest path in T is less than or equal to k, then the yield of T is of length less than or equal to 2^{k-1}.

Proof :- property of Binary tree.

Theorem - Let L be a CFL. Then we can find a natural no. n such that
(i) - every $z \in L$ with $|z| \geq n$ can be written as $uvwxy$ and $|vx| \geq 1$, $|vwx| \leq n$ and $uv^k wx^k y \in L$ for all $k \geq 0$.

proof of pumping lemma -

1- Construct a grammer $G = (V, T, P, S)$ in CNF generating $L - \{\epsilon\}$.

2- Let $|V| = m$ and $n = 2^m$.

Ex: $L = \{ w : w \in \{0,1\}^* \text{ and } |w| \text{ is a perfect square} \}$ is not CFL.
↳ $z = 0^{m^2}$
$|vx| = n \leq m$

Ex - $L = \{ww : w \in \{0,1\}^*\}$ is not CFL.
$z = 0^m 1^m 0^m 1^m$

Ex - prove that $L = \{a^p : p \text{ is prime}\}$ is not CFL.

Ans - Let L is CFL and n be the natural no. given by pumping lemma. Let p be a prime no. greater than n. Then $z = a^p \in L$.
Let $z = uvwxy$.
By pumping lemma - $uv^0wx^0y = uwy \in L$. So $|uwy|$ is a prime no, say q. Let $|vx| = r$. Then $|uv^q wx^q y|$ $= q + qr$. As $q + qr$ is not a prime, $uv^q wx^q y \notin L$

Unit 4

PUSHDOWN AUTOMATA (PDA)

formally a PDA M is a seven tuple $(Q, \Sigma, \delta, \Gamma, q_0, z_0, F)$ where -

Q - is a finite set of states

Σ - input alphabet

Γ - stack alphabet

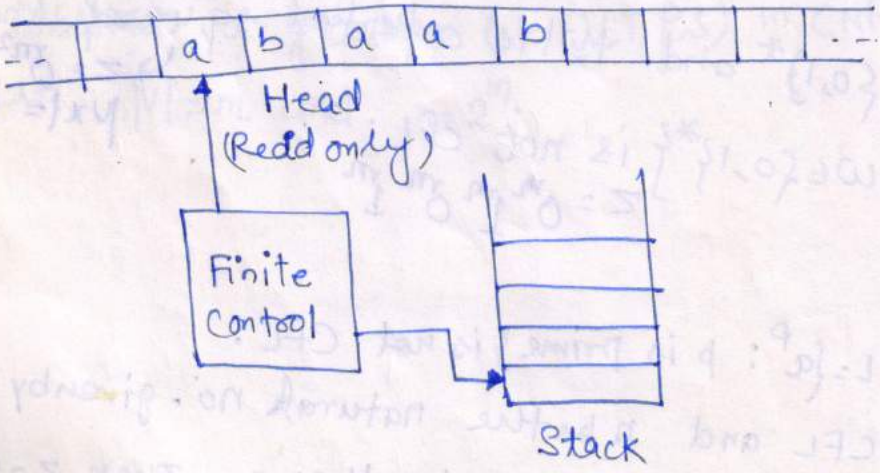
$q_0 \in Q$ is the "initial state"

$z_0 \in \Gamma$ is the "stack start symbol".

$F \subseteq Q$ is the set of final states

δ is a mapping function defined as follows -

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \text{ to } Q \times \Gamma^*$$



Instantaneous description :-

A configuration of PDA at a given instant, called instantaneous description (ID), is defined to be member of $Q \times \Sigma^* \times \Gamma^*$ i.e-

$$(P, ax, Z\alpha) \vdash_M (q, x, \beta\alpha)$$

- P : present state
- a : current input
- Z : Top of the stack (POP)
- q : Next state
- x : Input string left on the tape
- $\beta\alpha$: Stack configuration after reading input (a)

Acceptance by PDA :- 1) - Acceptance by final state :-

Let $M = (Q, \Sigma, \delta, \Gamma, q_0, Z_0, F)$, we say that a string w is accepted by the final state by PDA M if and only if $(q_0, w, Z_0) \vdash_M^* (P, \epsilon, \alpha)$ for some $P \in F$ and $\alpha \in \Gamma^*$. Thus language accepted by PDA M is

$$L(M) = \{ w : (q_0, w, Z_0) \vdash_M^* (P, \epsilon, \alpha) \text{ for some } P \text{ in } F \text{ \& } \alpha \in \Gamma^* \}$$

2)- Acceptance by empty stack :-

We define a string w is accepted by empty stack by PDA M iff $(q_0, w, z_0) \overset{*}{\vdash}_M (p, \epsilon, \epsilon)$ for some p in Q, and $N(M)$, the language accepted by empty stack is the set of all string accepted by empty stack by M. ie-

$$N(M) = \{ w : (q_0, w, z_0) \overset{*}{\vdash}_M (p, \epsilon, \epsilon), \text{ for some } p \in Q \}$$

⇒ Note that in case of $N(M)$ it is not necessary that PDA moves to any one of the final state.

Ex:- Design a PDA that accepts the language

$$L = \{ a^n b^n : n \geq 0 \}.$$

$\delta(q_0, \epsilon, z_0) = \{(q_1, z_0)\}$

$\delta(q_0, a, z_0) = \{(q_0, az_0)\}$

$\delta(q_0, a, a) = \{(q_0, aa)\}$

$\delta(q_0, b, a) = \{(q_1, \epsilon)\}$

$\delta(q_1, b, a) = \{(q_1, \epsilon)\}$

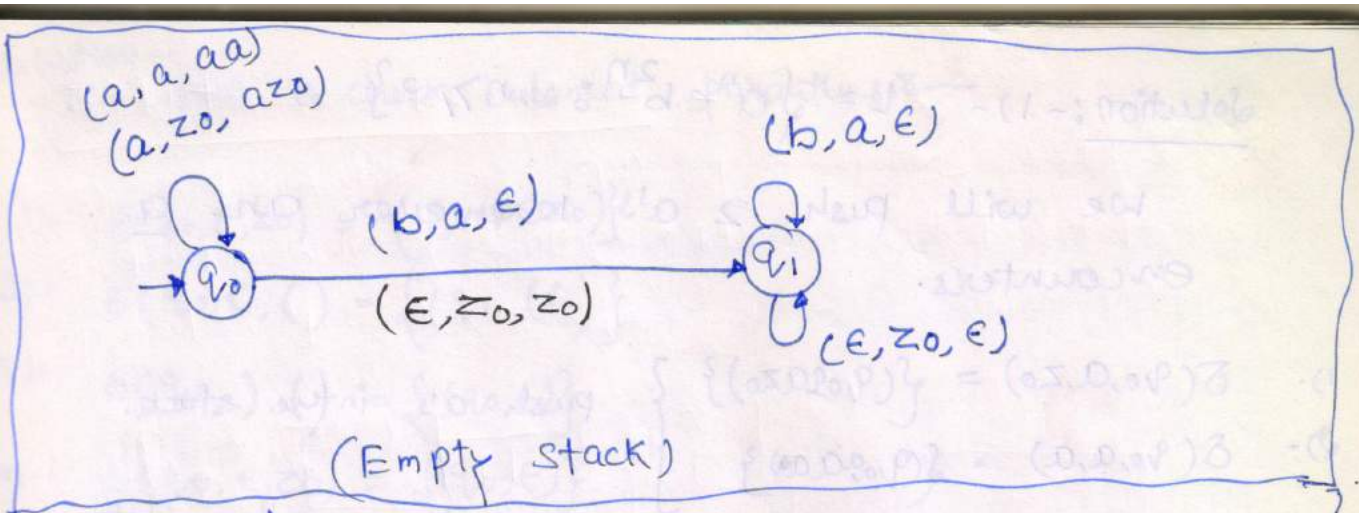
$\delta(q_1, \epsilon, z_0) = \{(q_1, \epsilon)\}$

↳ acceptance by Empty stack

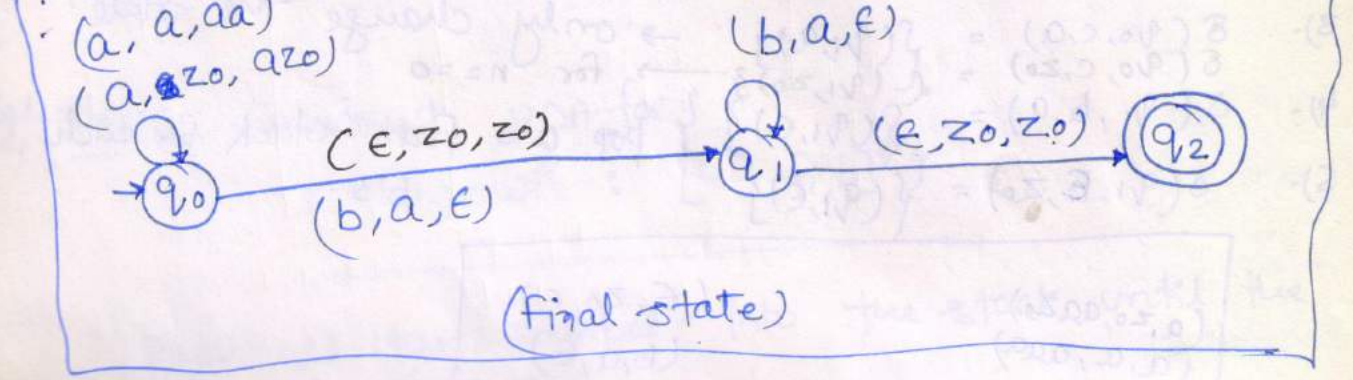
$\delta(q_1, \epsilon, z_0) = \{(q_2, z_0)\}$

where q_2 is final state

↳ Acceptance by final state.



(Empty stack)



(final state)

Exercise

ex:- 1) $L = \{ a^n c b^{2n} : n \geq 0 \}$

2) - $L = \{ w : w \in \{a,b\}^*$, and w contains equal No. of a's & b's $\}$.

3) - Construct a PDA to check balanced parenthesis.

4) - $L = \{ wc w^R : w \in \{a,b\}^* \}$.

5) - $L = \{ ww^R : w \in \{a,b\}^* \}$.

6) - $L = \{ w = w^R : w \in \{a,b\}^* \}$

7) - $L = \{ a^n b^m a^n : m,n \geq 1 \}$

8) - $L = \{ a^n b^m c^{m+n} : m,n \geq 1 \}$

9) - $L = \{ a^m b^n c^n d^m : m,n \geq 1 \}$.

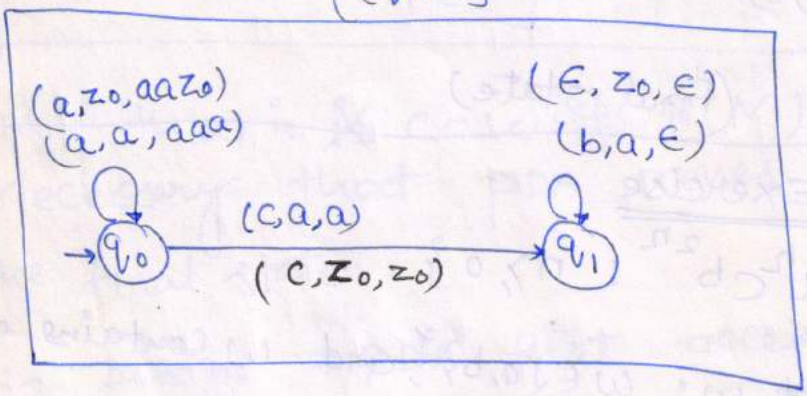
10) - $L = \{ a^m b^m c^n : m,n \geq 1 \}$

Solution:- 1)- $L = \{a^n c b^{2n} : n \geq 0\}$

We will push 2 a's whenever an a encounters.

1). $\delta(q_0, a, z_0) = \{(q_0, aaz_0)\}$
2). $\delta(q_0, a, a) = \{(q_0, aaa)\}$
} push a's in the stack.

3). $\delta(q_0, c, a) = \{(q_1, a)\}$ → only change the state.
$\delta(q_0, c, z_0) = \{(q_1, z_0)\}$ → for $n = 0$
4). $\delta(q_1, b, a) = \{(q_1, \epsilon)\}$
5). $\delta(q_1, \epsilon, z_0) = \{(q_1, \epsilon)\}$
} pop a's from stack for each b's.



Solution-2:- $L = \{w : w \in \{a,b\}^*$ and contains equal No. of as & bs$\}$

1)- $\delta(q_0, a, z_0) = \{(q_0, az_0)\}$
2)- $\delta(q_0, b, z_0) = \{(q_0, bz_0)\}$
3)- $\delta(q_0, a, a) = \{(q_0, aa)\}$
4)- $\delta(q_0, b, b) = \{(q_0, bb)\}$
} if top of the stack & current input symbol are same then push the input symbol into the stack.

5)- $\delta(q_0, b, a) = \{(q_0, \epsilon)\}$
6). $\delta(q_0, a, b) = \{(q_0, \epsilon)\}$
} pop on symbol from stack if Top of the stack & current input symbol differs.

7)- $\delta(q_0, \epsilon, z_0) = \{(q_0, \epsilon)\}$ } empty the stack.

Solution –
3)- PDA to check balanced paranthesis –

1)- $\delta(q_0, (, z_0) = \{(q_0, (z_0)\}$

2)- $\delta(q_0, (, () = \{(q_0, (()\}$

3)- $\delta(q_0,), () = \{(q_0, \epsilon)\}$

4)- $\delta(q_0, \epsilon, z_0) = \{(q_0, \epsilon)\}$

Solution – 4:– Construct PDA for

$L = \{wcw^R : w \{a,b\}^*\}$

push all the symbol into the stack until the symbol 'c' encounter.

1)- $\delta(q_0, a, z_0) = \{(q_0, az_0)\}$

2)- $\delta(q_0, b, z_0) = \{(q_0, bz_0)\}$

3)- $\delta(q_0, a, a) = \{(q_0, aa)\}$

4)- $\delta(q_0, b, a) = \{(q_0, ba)\}$

5)- $\delta(q_0, a, b) = \{(q_0, ab)\}$

6)- $\delta(q_0, b, b) = \{(q_0, bb)\}$

Whenever a 'c' encounter dont change the stack, but change the state.

7)- $\delta(q_0, c, a) = \{(q_1, a)\}$

8)- $\delta(q_0, c, b) = \{(q_1, b)\}$.
$\delta(q_0, c, z_0) = \{(q_1, z_0)\}$

9)- $\delta(q_1, a, a) = \{(q_1, \epsilon)\}$

10)- $\delta(q_1, b, b) = \{(q_1, \epsilon)\}$

} → w - abacaba

11)- $\delta(q_1, \epsilon, z_0) = \{(q_1, \epsilon)\}$

5)- $L = \{ ww^R : w \in \{a,b\}^* \}$

We use the same strategy as in previous example. The only difference is that how to find middle of string. Since in the previous example the symbol 'C' works as marker to show the middle. To guess the middle of the string we use following transitions in place of (7) & (8)-

7)- $\delta(q_0, \epsilon, a) = \{(q_1, a)\}$

8)- $\delta(q_0, \epsilon, b) = \{(q_1, b)\}$

and all the transitions in the previous example are same.

6) — Construct PDA for the language

$L = \{ w = w^R : w \in \{a,b\}^* \}$.

Clearly it is the palindrome of 'even' or 'odd' length. An even length palindrome is of form ww^R.

Let $w =$ ab | ba
w | w^R

↓ odd length palindrome

$\delta(q_0, a, z_0) = \{(q_0, az_0), (q_1, z_0)\}$
$\delta(q_0, b, z_0) = \{(q_0, bz_0), (q_1, z_0)\}$
$\delta(q_0, a, a) = \{(q_0, aa), (q_1, a)\}$
$\delta(q_0, b, a) = \{(q_0, ba), (q_1, a)\}$
$\delta(q_0, a, b) = \{(q_0, ab), (q_1, b)\}$
$\delta(q_0, b, b) = \{(q_0, bb), (q_1, b)\}$

$\delta(q_0, \epsilon, a) = \{(q_1, a)\}$
$\delta(q_0, \epsilon, b) = \{(q_1, b)\}$
$\delta(q_0, \epsilon, z_0) = \{(q_1, z_0)\}$
$\delta(q_1, a, a) = \{(q_1, \epsilon)\}$
$\delta(q_1, b, b) = \{(q_1, \epsilon)\}$
$\delta(q_1, \epsilon, z_0) = \{(q_1, \epsilon)\}$.

7)- $L = \{a^n b^m a^m ; m, n \geq 1\}$

Here the strings starts with a's with random nos. of b's then equal no. of a's in the first part of the string.

So push all a's into the stack and don't change the content of stack for b's and then pop on a from stack whenever a's encounters.

$\delta(q_0, a, z_0) = \{(q_0, az_0)\}$
$\delta(q_0, a, a) = \{(q_0, aa)\}$
} → pop all a's into stack.

$\delta(q_0, b, a) = \{(q_1, a)\}$
$\delta(q_1, b, a) = \{(q_1, a)\}$
} → don't change the stack for b's.

$\delta(q_1, a, a) = \{(q_2, \epsilon)\}$
$\delta(q_2, a, a) = \{(q_2, \epsilon)\}$
} pop a's

$\delta(q_2, \epsilon, z_0) = \{(q_2, \epsilon)\}$ → Empty stack.

8)- Construct PDA for language
$L = \{a^n b^m c^{m+n} : n, m \geq 0\}$

1- Push all a's & b's into the stack.

$\delta(q_0, a, z_0) = \{(q_0, az_0)\}$ } push all a's into stack
$\delta(q_0, a, a) = \{(q_0, aa)\}$

$\delta(q_0, b, z_0) = \{(q_1, bz_0)\}$ } push all b's into stack.
$\delta(q_1, b, b) = \{(q_1, bb)\}$

$\delta(q_1, c, b) = \{(q_2, \epsilon)\}$
$\delta(q_2, c, b) = \{(q_2, \epsilon)\}$
$\delta(q_2, c, a) = \{(q_2, \epsilon)\}$
$\delta(q_1, c, a) = \{(q_2, \epsilon)\}$
$\delta(q_2, \epsilon, z_0) = \{(q_2, \epsilon)\}$.

9)- Construct PDA for the language -
$L = \{a^m b^n c^n d^m : m, n \geq 1\}$

Push all a's & b's into the stack.

$\delta(q_0, a, z_0) = \{(q_0, az_0)\}$ / $\delta(q_0, b, z_0) = \{(q_1, bz_0)\}$
$\delta(q_0, a, a) = \{(q_0, aa)\}$
$\delta(q_0, b, a) = \{(q_1, ba)\}$ / $\delta(q_0, d, a) = \{(q_3, \epsilon)\}$
$\delta(q_1, b, b) = \{(q_1, bb)\}$

pop all b's ~~extract the~~ whenever a 'c' encounters

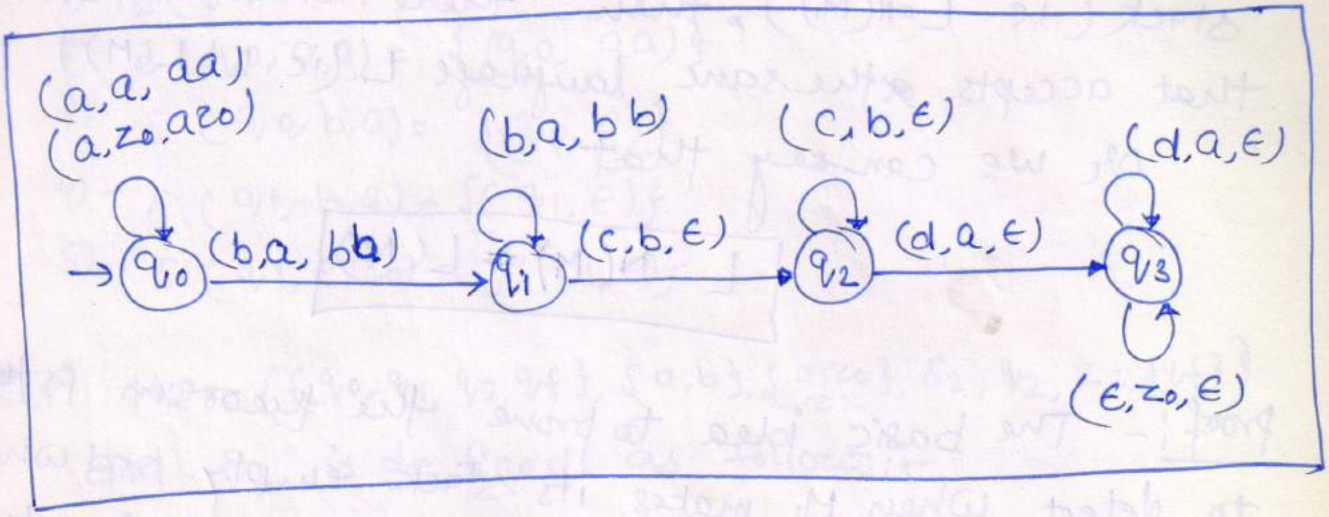
$\delta(q_1, c, b) = \{(q_2, \epsilon)\}$
$\delta(q_2, c, b) = \{(q_2, \epsilon)\}$

pop all a's whenever a 'd' encounters.

$\delta(q_2, d, a) = \{(q_3, \epsilon)\}$

$\delta(q_3, d, a) = \{(q_3, \epsilon)\}$

$\delta(q_3, \epsilon, z_0) = \{(q_3, \epsilon)\}$



10)- $L = \{a^m b^m c^n : n, m \geq 1\}$

push all a's into the stack.

$\delta(q_0, a, z_0) = (q_0, az_0)$

$\delta(q_0, a, a) = (q_0, aa)$

pop a's from stack whenever b encounters.

$\delta(q_0, b, a) = (q_1, \epsilon)$

$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, c, z_0) = (q_2, z_0)$ { if c encounters do not change the content of stack.

$\delta(q_2, c, z_0) = (q_2, z_0)$

$\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$ stack empty.

EQUIVALENE OF ACCEPTANCE BY FINAL STATE & EMPTY STACK :-

We shall now prove that the languages accepted by PDA's by final state are same as set of empty stack.

Theorem:- If L is accepted by PDA M_1 by empty stack (ie $L = N(M_1)$), then, there exist a PDA M_2 that accepts the same language L (ie $L = L(M)$)

Or we can say that

$$\boxed{L = N(M) = L(M)}$$

Proof:- The basic idea to prove the theorem is to detect when M_1 makes its stack empty and whenever M_1 makes its stack empty M_2 moves into the final state instead of making its stack empty.

Let $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, Z_1, \phi)$ (Here $L = N(M_1)$ by empty stack.

$M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, Z_2, \{q_f\})$

such that $Q_2 = Q_1 \cup \{q_2, q_f\}$

$\Gamma_2 = \Gamma_1 \cup \{Z_2\}$

and δ_2 as follows:-

1) - $\delta_2(q_2, \epsilon, Z_2) = \{(q_1, Z_1 Z_2)\}$

2) - for all $q \in Q_1$, $a \in \Sigma \cup \{\epsilon\}$, and $Z \in \Gamma_1$

$\delta_2(q, a, Z) = \delta_1(q, a, Z)$

3) - for all q in Q_1

$\delta_2(q, \epsilon, Z_2) = \{(q_f)\}$

Example :- Let $L = \{a^n b^n : n \geq 1\}$

Let $M_1 = (\{q_0, q_1\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \phi)$

such that $L = N(M)$ {by empty stack}.

& δ is defined as follows:-

1). $\delta(q_0, a, z_0) = \{(q_0, az_0)\}$

2)- $\delta(q_0, a, a) = \{(q_0, aa)\}$

3)- $\delta(q_0, b, a) = \{(q_1, \epsilon)\}$

4)- $\delta(q_1, b, a) = \{(q_1, \epsilon)\}$

5)- $\delta(q_1, \epsilon, z_0) = \{(q_1, \epsilon)\}$

Let $M_2 = (\{q_0, q_1, q_2, q_f\}, \{a, b\}, \{a, z_0, z_2\}, \delta_2, q_2, z_2, \{q_f\})$

and δ_2 is defined as follows:-

A). $\delta_2(q_2, \epsilon, z_2) = \{(q_0, z_0 z_2)\}$

$\delta_2(q_0, \epsilon, z_0) = \{(q_0, \epsilon)\}$

B)- 1)- $\delta_2(q_0, a, z_0) = \{(q_0, az_0)\}$

2)- $\delta_2(q_0, a, a) = \{(q_0, aa)\}$

3)- $\delta_2(q_0, b, a) = \{(q_1, \epsilon)\}$

4)- $\delta_2(q_1, b, a) = \{(q_1, \epsilon)\}$

5)- $\delta_2(q_1, \epsilon, z_0) = \{(q_1, \epsilon)\}$

C)- $\delta_2(q_0, \epsilon, z_2) = \{(q_f, \epsilon)\}$

$\delta_2(q_1, \epsilon, z_2) = \{(q_f, \epsilon)\}$

By final state.

Ex- Let $w = aabb$. process w on machine M_1

$(q_0, aabb, z_0) \vdash_{M_1} (q_0, abb, az_0) \vdash_{M_1} (q_0, bb, aaz_0) \vdash_{M_1} (q_1, b, az_0) \vdash_{M_1} (q_1, \epsilon, z_0) \vdash_{M_1} (q_1, \epsilon, \epsilon)$

accepted.

on Machine M_2 -

$(q_2, aabb, z_2) \vdash_{M_2} (q_0, aabb, z_0 z_2) \vdash_{M_2} (q_0, abb, az_0 z_2) \vdash_{M_2} (q_0, bb, aaz_0 z_2)$

$\vdash (q_1, b, az_0 z_2) \vdash_{M_2} (q_1, \epsilon, z_0 z_2) \vdash_{M_2} (q_1, \epsilon, z_2) \vdash_{M_2} (q_f, \epsilon\epsilon)$

accepted

Theorem 2:- If L is accepted by the PDA with final state then L is also accepted by the empty stack PDA.

Proof :- Let $L = L(M_1)$ then there exists M_2 such that $L = N(M_2)$.

Let $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, Z_1, F)$ and

$M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, Z_2, \phi)$ where -

$Q_2 = Q_1 \cup \{q_2, q_e\}$

$\Gamma_2 = \Gamma_1 \cup \{Z_2\}$

and δ_2 is defined as follows:-

1)- $\delta_2(q_2, \epsilon, Z_2) = \{(q_1, Z_1 Z_2)\}$

2)- for all q in Q_1, $a \in \Sigma \cup \{\epsilon\}$ and $Z \in \Gamma_1$

$\delta_2(q, a, Z) = \delta_1(q, a, Z)$

3)- for all $q \in F$ and $Z \in \Gamma_1$

$\delta_2(q, \epsilon, Z) = \{(q_e, \epsilon)\}$

4)- ~~for all $Z \in \Gamma_2$~~ $\delta_2(q_e, \epsilon, Z_2) = \{(q_e, \epsilon)\}$

Deterministic PDA

A PDA $M = (Q, \Sigma, \delta, \Gamma, q_0, Z_0, F)$ is deterministic if there is no configuration for which M has a choice of more than one moves. Formally, we say that a PDA M is deterministic if:-

for every $q \in Q$, $a \in \Sigma \cup \{\epsilon\}$ and $Z \in \Gamma$

1) - $\delta(q, a, Z)$ contains atmost one element

2) - If $\delta(q, \epsilon, Z) \neq \phi$, then

$\delta(q, a, Z) = \phi$ for all $a \in \Sigma$.

⇒ A language L is deterministic context free language (DCFL) if there exists a DPDA accepting L.

⇒ we may recall that we did not defined a "deterministic regular language" although we considered both DFA and NFA. The reason, ofcourse was that for any NFA there is an equivalent DFA recognising the same language. The corresponding result for PDA is not true.

example:- Show that $L=\{a^n c b^{2n} \mid n \geq 0\}$ is a deterministic context free language.

Sol:- 1). $\delta(q_0, a, z_0) = \{(q_0, aaz_0)\}$

2). $\delta(q_0, a, a) = \{(q_0, aaa)\}$

3). $\delta(q_0, c, a) = \{(q_1, a)\}$

4). $\delta(q_1, b, a) = \{(q_1, \epsilon)\}$

5). $\delta(q_1, \epsilon, z_0) = \{(q_1, \epsilon)\}$

for all $q \in Q, a \in \Sigma \cup \{\epsilon\}, z \in \Gamma$

$\delta(q, a, z)$ contains atmost one element.

∴ transition no. (5) contains ϵ moves -

and $\left.\begin{array}{l} \delta(q_1, a, z_0) = \phi \\ \delta(q_1, b, z_0) = \phi \end{array}\right\}$

Hence L is DCFL:-

Equivalence of PDA's and CFG's :-

Theorem:- Each context free language is accepted by some pushdown automaton.

Proof:- To prove theorem we will prove that if L is a context free language, then there exists a PDA M such that

$L = N(M)$

⇒ The PDA has only one state "q" and remains permanently in state "q".

Let $G = (V, T, P, S)$ is context free grammar.

(1) 1)- Convert the grammar in GNF.

2)- $M = (\{q\}, T, V, \delta, q, S, \phi)$

and δ is defined as follows:-

$\delta(q, a, A)$ contains (q, α) for each rule

$A \to a\alpha$

(2) If G is not in GNF.

1- $\delta(q, \epsilon, z_0) = \{(q, S)\}$

2- for all prodⁿ of form $A \to \alpha$

$\delta(q, \epsilon, A) = \{(q, \alpha)\}$

3- for $a \in T$

$\delta(q, a, a) = \{(q, \epsilon)\}$

ex:- Let $G = (V, T, P, S)$

$V = \{A_1, A_2, A_3\}$

$T = \{a, b\}$

$S = A_1$

$P = \{ A_1 \to aA_2 \mid aA_3$

$A_2 \to aA_2A_3 \mid aA_3A_3$

$A_3 \to b\}$

We can easily verify that $L = \{a^n b^n : n \geq 1\}$.

1- S-productions are given by $S \to [q_0, z_0, q]$ for every $q \in Q$.

2 - Each move erasing a stack symbol given by $(q', \epsilon) \in \delta(q, a, z)$ induces the production $[q, z, q'] \to a$

3 - Each move not erasing a stack symbol given by $\delta(q, a, z) = (q_1, z_1 z_2 \cdots z_m)$ induces many productions of the form -

$[q, z, q'] \to a[q_1, z_1, q_2][q_2, z_2, q_3] \cdots [q_m, z_m, q']$

where each of the states $q', q_2, q_3 \cdots q_m$ can be any state in Q.

Ex- $\delta(q_0, b, z_0) = (q_0, zz_0)$ $\quad \delta(q_0, a, z) = (q_1, z)$

$\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$ $\quad \delta(q_1, b, z) = (q_1, \epsilon)$

$\delta(q_0, b, z) = (q_0, zz)$ $\quad \delta(q_1, a, z_0) = (q_0, z_0)$

$V = \{S, [q_0, z_0, q_0], [q_0, z_0, q_1], [q_1, z_0, q_0], [q_1, z_0, q_1],$
$[q_0, z, q_0], [q_0, z, q_1], [q_1, z, q_0], [q_1, z, q_1]\}$

P $\quad S \to [q_0, z_0, q_0] \mid [q_0, z_0, q_1]$

$\delta(q_0, b, z_0) = (q_0, zz_0)$

$[q_0, z_0, q_0] \to b[q_0, z, q_0][q_0, z_0, q_0] \mid b[q_0, z, q_1][q_1, z_0, q_0]$

$[q_0, z_0, q_1] \to b[q_0, z, q_0][q_0, z_0, q_1] \mid b[q_0, z, q_1][q_1, z_0, q_1]$

$\delta(q_0, b, z) = (q_0, zz)$

$[q_0, z, q_0] \to b[q_0, z, q_0][q_0, z, q_0] \mid b[q_0, z, q_1][q_1, z, q_0]$

$[q_0, z, q_1] \to b[q_0, z, q_0][q_0, z, q_1] \mid b[q_0, z, q_1][q_1, z, q_1]$

$\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$

$[q_0, z_0, q_0] \to \epsilon$

Now the corresponding PDA is—

$M = (\{q\}, T, V, \delta, q, A_1, \phi)$

$\delta(q, a, A_1) = \{(q, A_2), (q, A_3)\}$

$\delta(q, a, A_2) = \{(q, A_2A_3), (q, A_3A_3)\}$

$\delta(q, b, A_3) = \{(q, \epsilon)\}$

Let $\omega = aabb$—

$(q, aabb, A_1) \vdash_M (q_1, abb, A_2) \vdash_M (q, bb, A_3A_3)$

$\vdash_M (q, b, A_3) \vdash_M (q, \epsilon, \epsilon)$

Theorem:- If a language is accepted by a PDA then it is a context free language.

Proof:- Let L is $N(M)$ for some PDA $M = (Q, \Sigma, \delta, \Gamma, q_0, Z_0, \phi)$. Let $G = (V, T, P, S)$ be a CFG where V is the set of objects of the form $[q, A, P]$, such that $q, P \in Q$, $A \in \Gamma$, plus the new symbol S. Then the set of production P having

S-production]

1) – for each $q \in Q$, the production $S \rightarrow [q_0, Z_0, q]$

2) – $[q, A, q_{m+1}] \rightarrow a[q_1, B_1, q_2][q_2, B_2, q_3] \cdots [q_m, B_m, q_{m+1}]$

for each $q, q_1, q_2 \cdots q_{m+1} \in Q$, $a \in \Sigma \cup \{\epsilon\}$ and

150 $A, B_1, B_2 \cdots B_m \in \Gamma$, such that

$\delta(q, a, A)$ contains $(q_1, B_1B_2B_3 \cdots B_m)$ (if $m = 0$ then [illegible] $[q, A, q_1] \rightarrow a$

ex:- Let $M = (\{q_0, q_1\}, \{a, b\}, \{X, Z_0\}, \delta, q_0, Z_0, \phi)$

Where δ is given by -

$\delta(q_0, a, Z_0) = \{(q_0, XZ_0)\}$ ✓

$\delta(q_0, a, X) = \{(q_0, XX)\}$

$\delta(q_0, b, X) = \{(q_1, \epsilon)\}$ ✓

$\delta(q_1, b, X) = \{(q_1, \epsilon)\}$

$\delta(q_1, \epsilon, X) = \{(q_1, \epsilon)\}$

$\delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$

Solution:- To construct a CFG $G = (V, T, P, S)$ that generates $N(M)$, Let

$V = (\{S, [q_0, X, q_0], [q_0, X, q_1], [q_1, X, q_0], [q_1, X, q_1],$
$[q_0, Z_0, q_0], [q_0, Z_0, q_1], [q_1, Z_0, q_0], [q_1, Z_0, q_1]\}, \{a, b\}, P, S)$

where P contains :-

1) - $S \to [q_0, Z_0, q_0]$
$S \to [q_0, Z_0, q_1]$

2) - For $\delta(q_0, a, Z_0) = \{(q_0, XZ_0)\}$

$[q_0, Z_0, q_0] \to a[q_0, X, q_0][q_0, Z_0, q_0] \mid a[q_0, X, q_1][q_1, Z_0, q_0]$

$[q_0, Z_0, q_1] \to a[q_0, X, q_0][q_0, Z_0, q_1] \mid a[q_0, X, q_1][q_1, Z_0, q_1]$

3. for $\delta(q_0, a, X) = \{(q_0, XX)\}$

$[q_0, X, q_0] \to a[q_0, X, q_0][q_0, X, q_0] \mid a[q_0, X, q_1][q_1, X, q_0]$

$[q_0, X, q_1] \to a[q_0, X, q_0][q_0, X, q_1] \mid a[q_0, X, q_1][q_1, X, q_1]$

4) $[q_0, X, q_1] \to b$ ✓
$[q_1, X, q_1] \to b$
$[q_1, X, q_1] \to \epsilon$
$[q_1, Z_0, q_1] \to \epsilon$

TWO STACK PUSHDOWN AUTOMATA

Let us define a two-stack PDA to be a eight tuple -

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

$M = (Q, \Sigma, \delta, \Gamma, q_0, Z_1, Z_2, F)$

where Z_1 - stack start symbol, for stack 1

Z_2 - " " " " stack 2

and $Q, \Sigma, \Gamma, q_0, F$ are defined previously.

δ is the transition relation is defined as follows:-

$$(Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma^* \times \Gamma^*) \longrightarrow (Q \times \Gamma^* \times \Gamma^*)$$

- Γ^* (pops the 1st stack)
- Γ^* (pops the 2nd stack)
- Γ^* (push the symbols on 1st stack)
- Γ^* (push the symbols on 2nd stack.)

example :- Construct a two-stack PDA for the language

$$L = \{a^n b^n c^n : n \geq 0\}$$

$\delta(q_0, \epsilon, Z_1, Z_2) = \{(q_2, \epsilon, \epsilon)\}$

$\delta(q_0, a, Z_1, Z_2) = \{(q_0, aZ_1, Z_2)\}$

$\delta(q_0, a, a, Z_2) = \{(q_0, aa, Z_2)\}$

→ push all a's into 1st stack

$\delta(q_0, b, a, Z_2) = \{(q_1, a, bZ_2)\}$

$\delta(q_1, b, a, b) = \{(q_1, a, bb)\}$

→ push all b's into 2nd stack.

$\delta(q_1, c, a, b) = \{(q_2, \epsilon, \epsilon)\}$

$\delta(q_2, c, a, b) = \{(q_2, \epsilon, \epsilon)\}$

→ pop a from 1st stack & b from 2nd stack when a 'c' on tape.

$\delta(q_2, \epsilon, Z_1, Z_2) = \{(q_2, \epsilon, \epsilon)\}$ → empty the stack.

Example :- $L = \{a^n b^n a^n b^n : n \geq 0\}$

$\delta(q_0, \epsilon, z_1, z_2) = \{(q_3, \epsilon, \epsilon)\}$

$\delta(q_0, a, z_1, z_2) = \{(q_0, az_1, az_2)\}$

$\delta(q_0, a, a, a) = \{(q_0, aa, aa)\}$ } push a's on both stacks.

$\delta(q_0, b, a, a) = \{(q_1, \epsilon, ba)\}$

$\delta(q_1, b, a, b) = \{(q_1, \epsilon, bb)\}$ } on b's pop from stack-1 and push b on stack-2.

$\delta(q_1, a, z_1, b) = \{(q_2, az_1, \epsilon)\}$

$\delta(q_2, a, a, b) = \{(q_2, aa, \epsilon)\}$ } Push a's on stack-1 and pop b from stack-2.

$\delta(q_2, b, a, a) = \{(q_3, \epsilon, \epsilon)\}$

$\delta(q_3, b, a, a) = \{(q_3, \epsilon, \epsilon)\}$

$\delta(q_3, \epsilon, z_1, z_2) = \{(q_3, \epsilon, \epsilon)\}$

L = Construct a two-stack PDA for the language

$L = \{a^n b^n c^n d^n : n \geq 1\}$

$\delta(q_0, a, z_1, z_2) = \{(q_0, az_1, az_2)\}$

$\delta(q_0, a, a, a) = \{(q_0, aa, aa)\}$

$\delta(q_0, b, a, a) = \{(q_1, \epsilon, ba)\}$

$\delta(q_1, b, a, b) = \{(q_1, \epsilon, bb)\}$

$\delta(q_1, c, z_1, b) = \{(q_2, cz_1, \epsilon)\}$

$\delta(q_2, c, c, b) = \{(q_2, cc, \epsilon)\}$

$\delta(q_2, d, c, a) = \{(q_3, \epsilon, \epsilon)\}$

$\delta(q_3, d, c, a) = \{(q_3, \epsilon, \epsilon)\}$

$\delta(q_3, \epsilon, z_1, z_2) = \{(q_3, \epsilon, \epsilon)\}$

DECISION ALGORITHMS FOR CFL'S

1)- Emptiness :-

Theorem :- If G is a Context free grammar, then there exists some algorithm to answer whether $L(G) = \phi$ or not.

proof :- To test whether the language generated by grammar G is empty or not first we elliminate the useless symbols & productions.

1). If the start symbol of G is found to be useless then $L(G) = \phi$

2)- otherwise $L(G) \neq \phi$.

2)- Finiteness :-

Theorem :- for a given CFG G, there exists an algorithm to determine whether L(G) is infinite or not.

proof :- 1)- Remove all ϵ-productions, unit productions and useless symbols.

2)- Draw a directed graph.

a)- If $A \to \alpha B \beta$ is a production then there is an edge from vertex A to B.

3)- If the directed graph has a cycle then L(G) is infinite.

$\Rightarrow$ If any variable is repeated in the derivation ie for any $A \in V$, $\boxed{A \overset{*}{\Rightarrow} \alpha A \beta}$ then infinite productions.

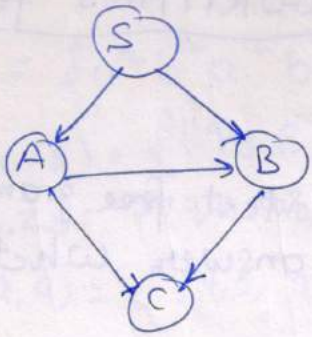
ex - Let $G = (V, T, P, S)$ is a CFG, with

$S \to AB \mid a$

$A \to BC \mid b$

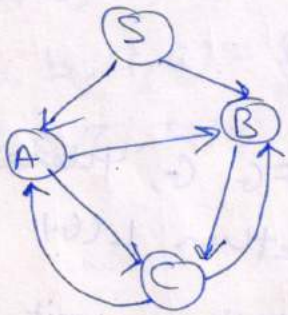
$B \to CC \mid c$

$C \to d$



Since, there is no cycle in the graph, the L(G) is finite.

ex-2 :- $S \to AB \mid a$
$A \to BC \mid B$
$B \to CC \mid c$
$C \to AB \mid d$



Since there is a cycle in the graph then the L(G) is infinite.

ex :- Consider the grammar G with productions -

$S \to ABa \mid bAD \mid b$
$A \to Ca \mid bDA$
$B \to bAA$
$C \to aDa \mid bA \mid aa$
$D \to DAbA$

test whether L(G) is infinite or finite.

3)- Membership ;-

For a given grammar G and any string w there exists an algorithm to determine whether $w \in L(G)$.

Proof :- CYK algorithm (Cocke - Younger - Kasami) -

1- Grammar must be in CNF.

2- Begin

for i=1 to n do $\quad n = |w|$

$V_{i1} = \{A : A \to a$ is a production and i^{th} symbol of w is a$\}$

For j=2 to n do

for i=1 to n-j+1 do

begin

$V_{ij} = \phi$

For K= 1 to j-1 do

$V_{ij} = V_{ij} \cup \{A : A \to BC$ is a prod^n & B is in V_{iK} & C is in $V_{i+K, j-K}\}$

end

if S (start symbol) is a member of V_{1n} then $w \in L(G)$

end.

1)- First convert the grammar in to CNF.

ex. $S \to AB|BC$

$A \to BA|a$

$B \to CC|b$

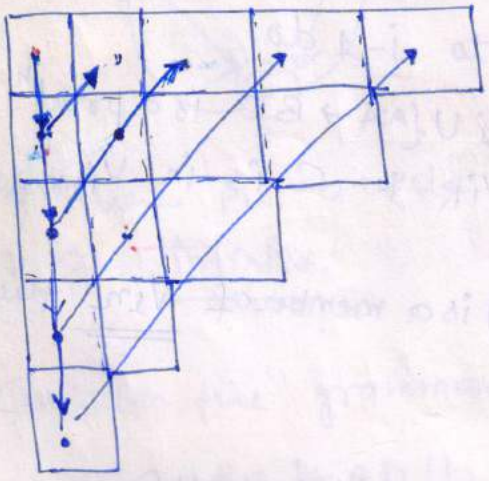
$C \to AB|a$

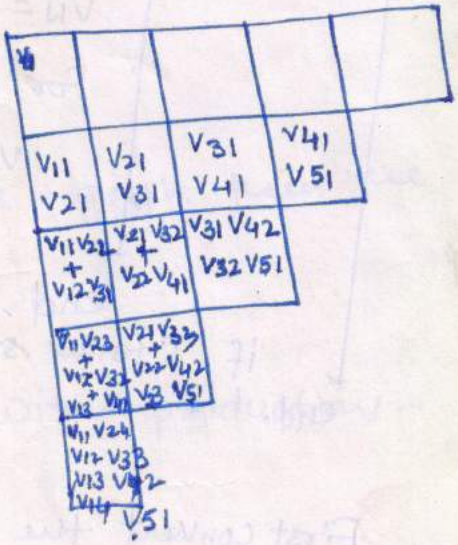
Let $w =$ baaba.

$i \rightarrow$	b (1)	a (2)	a (3)	b (4)	a (5)
1	B	A, C	A, C	B	A, C
2	S, A	B	S, C	S, A	
3	ϕ	B.	B		
4	ϕ	S, A, C			
5	A, S, C				

($j \downarrow$)

V_{15} contains start symbol so $w \in L(G)$.





ex → $S \rightarrow AS \mid a$
$A \rightarrow BC$
$B \rightarrow a$
$C \rightarrow AB$

$w = aaaa$ check whether $w \in L(G)$ or not.

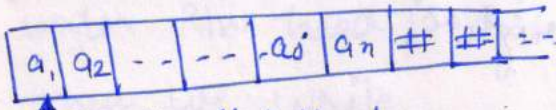
ex - $S \rightarrow AB$
$A \rightarrow BB \mid a$
$B \rightarrow AB \mid b$
} $w =$ aabbb

157

UNIT 5

TURING MACHINES

1936



* Among various formulations, Turing's formulation is accepted as a model of algorithm or computation.

The church-Turing thesis states that any algorithmic procedure that can be carried out by Turing machine/human computer can be carried out by Turing machine

Like an FA and PDA, T.M has a finite control of states, it has a linear tape, which has a left end and is potentially infinite to the right. The tape is divided into cells and each cell can hold one symbol from the alphabet. If the square (cell) has no symbol we say it contains the blank symbol.

The Machine has a head (Read/Write) that at any time read a symbol and write a symbol.

⇒ The head can move left or write from the current position after reading one symbol.

The turing machine can formally denoted as

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

Where Q is the finite set of states

Γ is the " " of tape symbols.

$\# \in \Gamma$ is the blanck.

Σ is the finite set of input symbols.

q_0 is the initial state $\in Q$.

F is the set of final states.

δ is the transition function and defined as follows-

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

we use the symbols L and R to denote movement of head to the left or right

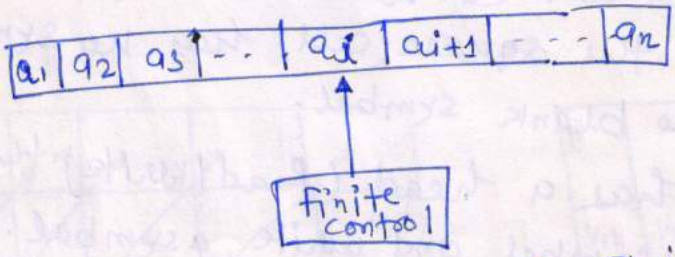
Instantaneous description:- (ID)

The ID of turing machine is defined in terms of entire input string and the current state. We denote an ID of the turing machine M by $\alpha q \beta$ where

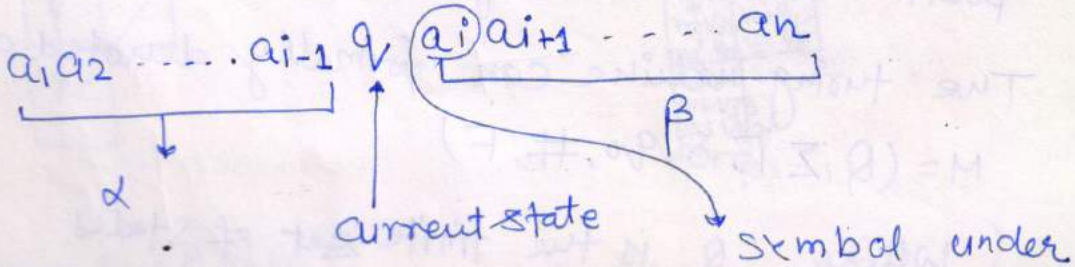
q is the present state of M.

α, β is the string in Γ^*, ie entire input string.

ex-



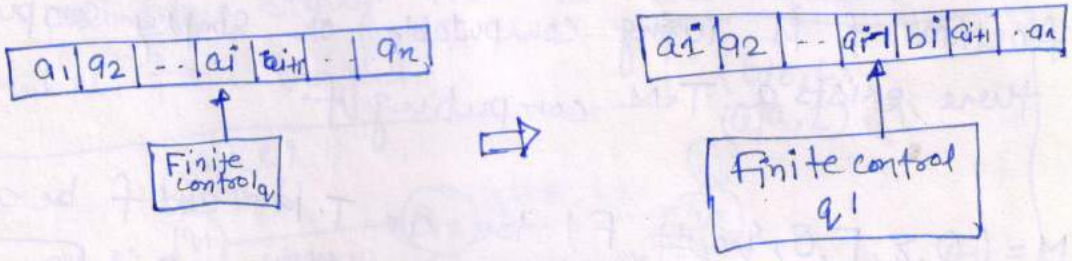
Let the current state of finite control is q, then ID of this machine is-



Moves in TM :-

Let us consider a. $a_1 a_2 \ldots a_{i-1} q a_i a_{i+1} \ldots a_n$ be an ID. Here the current state is q and the symbol under R/W head is a_i; suppose $\delta(q, a_i) = (q', b_i, L)$ then we write

$$a_1 a_2 \ldots q' a_{i-1} b_i a_{i+1} \ldots a_n$$

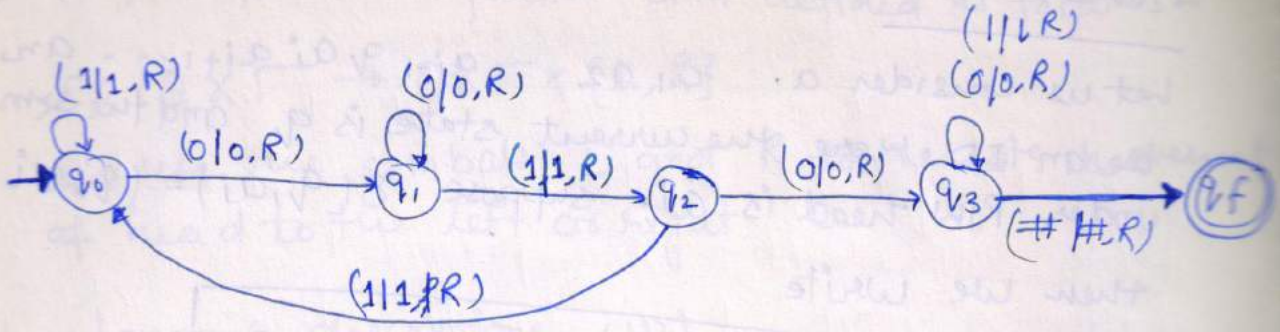


Acceptance by Turing Machine :-

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ is a TM. The language accepted by M, denoted by L(M), is the set of those strings in Σ^* that cause M to enter a final state when placed, justified at left on the tape of M, with M in state q_0 and the tape head of M at the leftmost cell. formally —

$$L(M) = \{ w : w \in \Sigma^* \text{ and } q_0 w \vdash_M^* \alpha q_f \beta \text{ for some } q_f \in F, \alpha, \beta \in \Gamma^* \}$$

ex- Construct a TM that accepts the language $L = \{ w : w \in \{0,1\}^* \text{ and } w \text{ contains } 010 \text{ as a substring} \}$.



TURING COMPUTABLE FUNCTION :-

A function f is Turing computable or simply computable if there exists a T.M computing f.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ be a T.M and f be a function with domain D. Let for any input string w $\boxed{f(w) = y}$ we say that the function is computed by T.M M if —

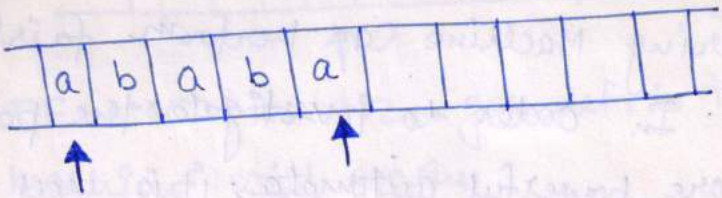
$q_0 w \overset{*}{\underset{M}{\vdash}} \alpha_1 q \alpha_2$ for all $w \in D$ and $y = \alpha_1 \alpha_2$.

⇒ If a function with domain D & K-tuples say $w_1, w_2 \cdots w_K$, M computes f if for every K-tuples $w_1, w_2 \cdots w_K$ on which f is defined —

$q_0 w_1 \# w_2 \# \cdots \# w_K \overset{*}{\underset{M}{\vdash}} \alpha_1 q \alpha_2$

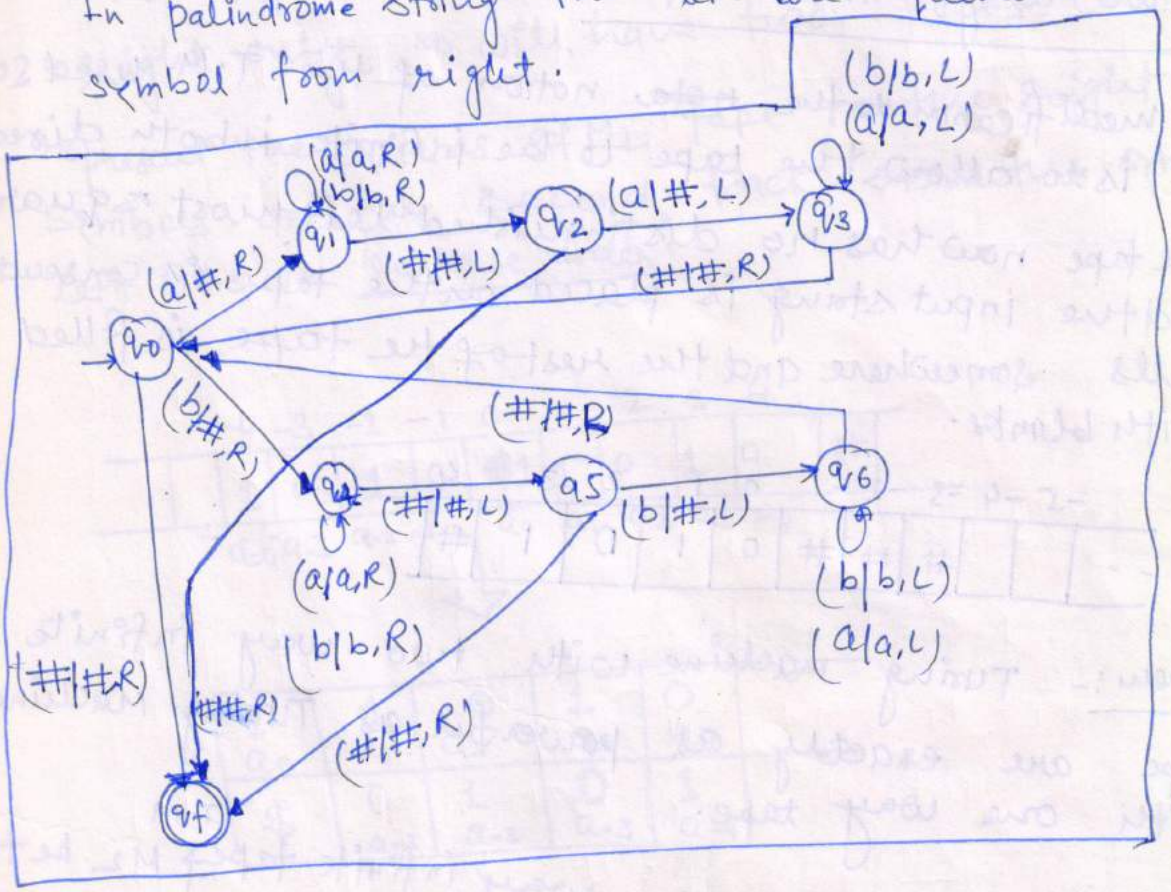
where $\boxed{f(w_1, w_2 \cdots w_K) = \alpha_1 \alpha_2}$

ex:- Construct a T.M that recognize the palindrome strings.

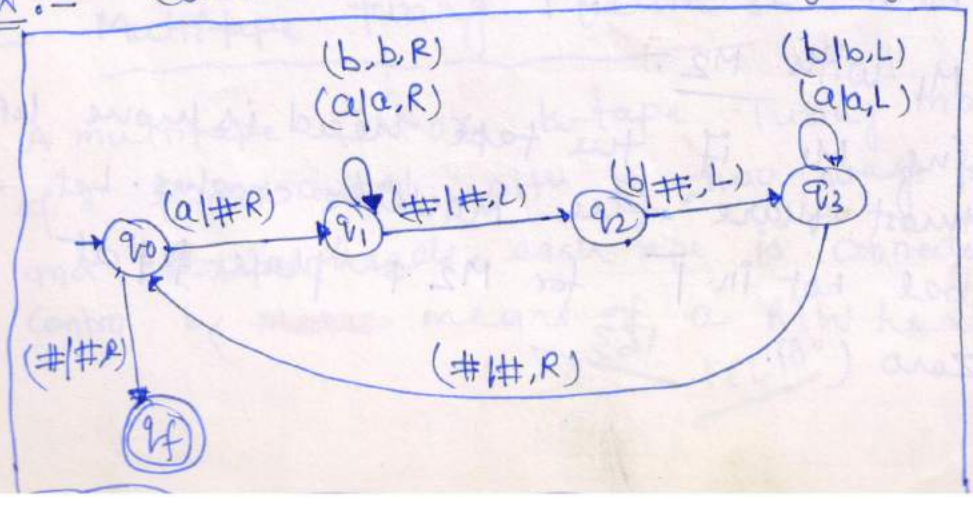


There are two types of palindrome- 1)- Even palindrome.
2)- Odd palindrome.

In palindrome string from left are equivalent to the symbol from right.



ex:- Construct a T.M for the language $L = \{a^n b^n : n \geq 0\}$



VARIANTS OF TURING MACHINE :-

It is clear that turing Machine can perform fairly powerful computations. In order to investigate the possibility of designing yet more powerful automata; it is well to consider the effect of extending the Turing machine model in various ways -

1) - Two way infinite tape :-

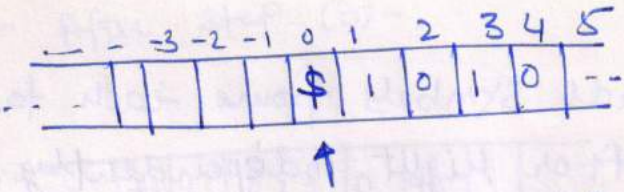
A modification to the ~~nota~~ notion of the T.M used so far is to allow the tape to be infinite in both directions. The tape now has no distinguished leftmost square and the input string is placed on the tape in consecutive cells somewhere and the rest of the tape is filled with blanks.

Theorem :- Turing machine with two way infinite tape are exactly as powerful as Turing Machine with one way tape.

Proof :- Let M_1 be the one way infinite tape turing machine. T.M M_2 be the two way infinite tape

⇒ Simulate M_1 with M_2.

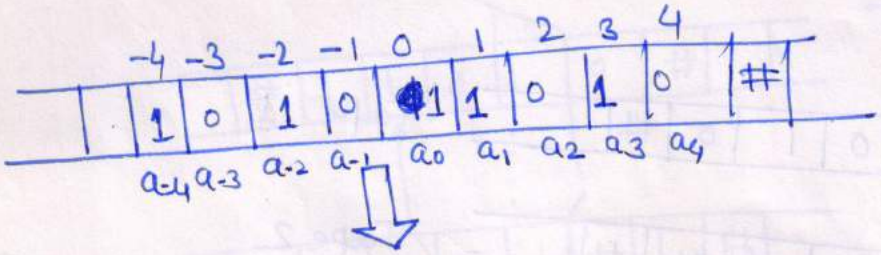
In machine M_1 if the tape head is move left from leftmost square the machine crashes. Let $ be a symbol not in Γ for M_2 & place $$ at position zero ("0").



IF Machine M_2 goes left to the "$" symbol then machine will crashes.

Simulate M_2 with M_1 :-

Let M_1 have a tape that is infinite to the right only, M_1 will have two tracks, one to represent the squares of M_2's tape to the right of the Symbols, and the second track stores the symbols left side in reverse order.



1 a_0	1 a_1	0 a_2	1 a_3	0 a_4
$	0 a_{-1}	1 a_{-2}	0 a_{-3}	1 a_{-4}

2)- Multitape Turing Machine :-

A multitape T.M or K-tape Turing machine consists of a finite control with K, two way infinite tapes and K-tapes heads, each tape is connected to the finite control by ~~means~~ means of a R|W head.

In one move —

1)- The finite control reads symbols from each tape.

2)- Move each head Left or right independently.

3)- Write symbols on the current cells.

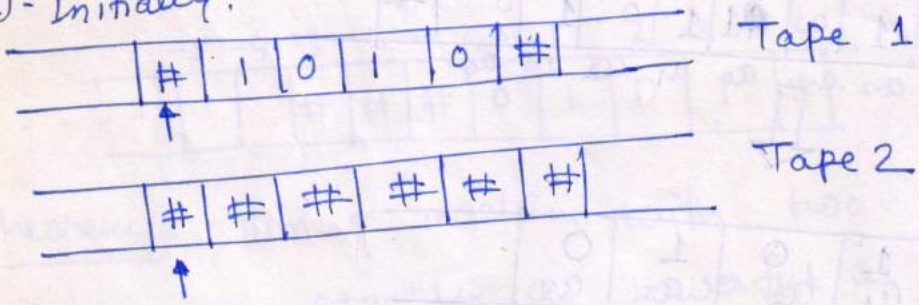
ex- Copying machine — (with two tapes).

1) — Copy each symbol of the first tape onto second, moving the heads of both tapes to the right, until blanck is found.

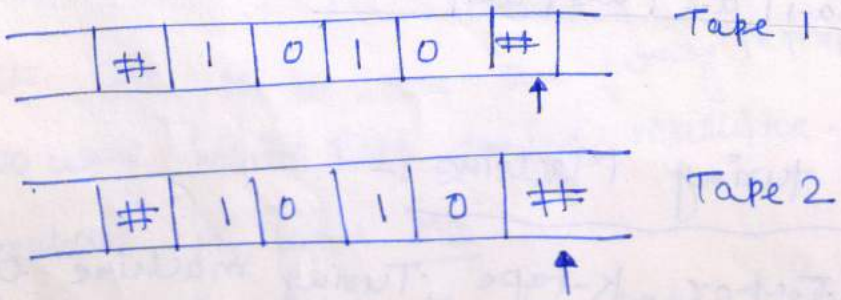
2)- Move the head of the First tape to the left until blanck is found.

3) — Again move the head of both tapes to the right, copying the symbols from first head to the second tape.

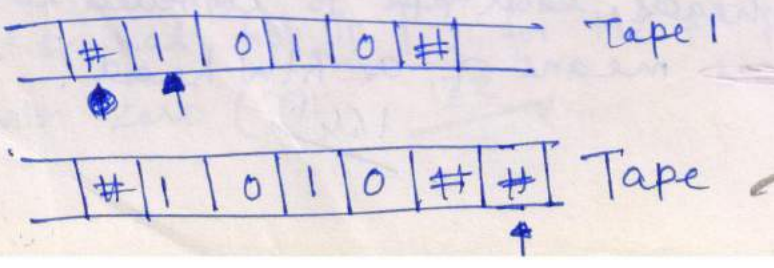
(a)- Initially.



(b)- After step (1) -

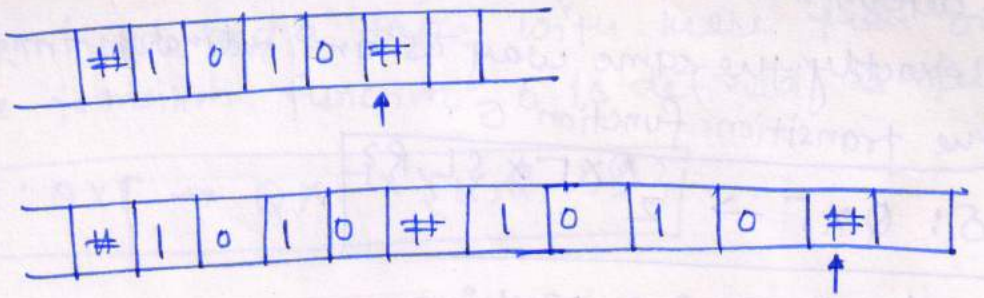


(c)- After step (2) -



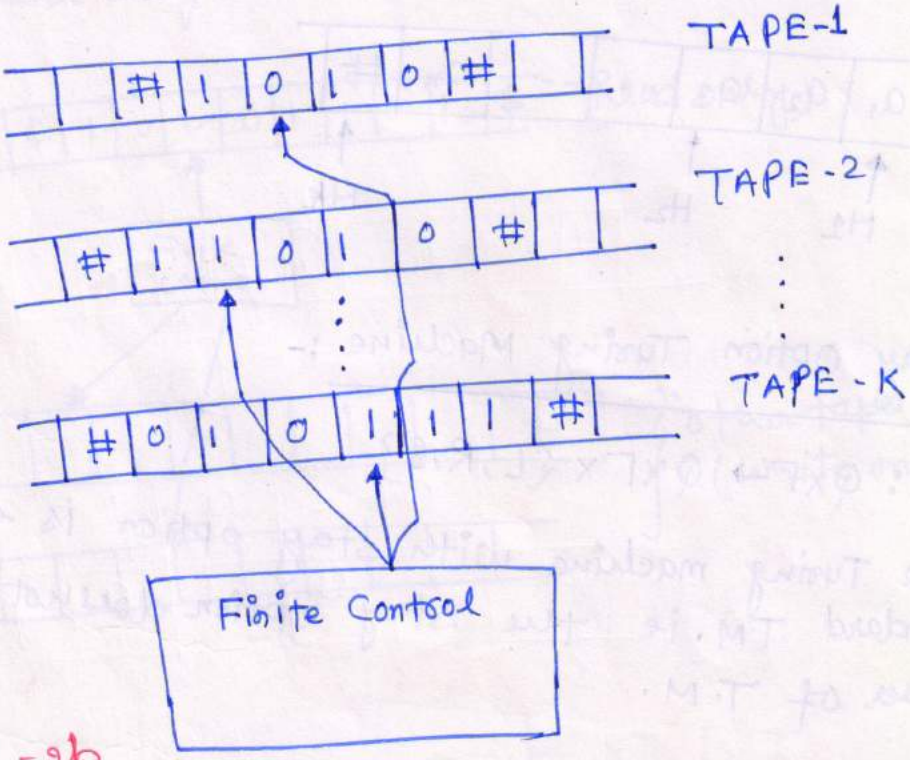
165

(d) – After step (3) –

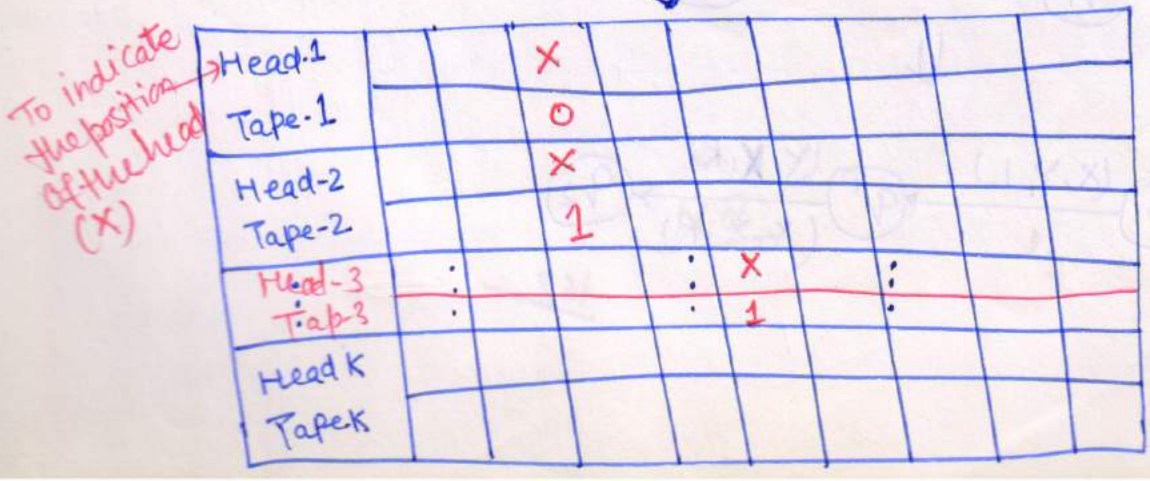


Theorem:– Turing machine with multiple tapes are exactly as powerful as basic Turing machine.

Proof:–



Total No of tracks = 2k



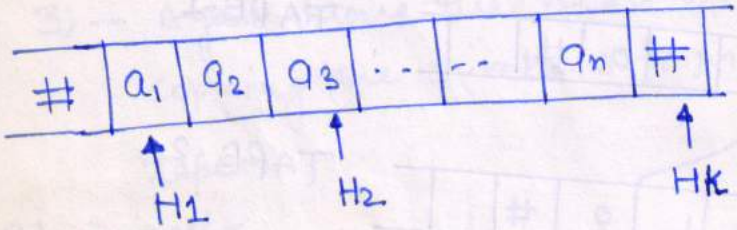
3)- Non-deterministic Turing Machine :-

A non-deterministic TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ is defined exactly the same way as an ordinary TM, except that the transition function δ.

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$$

4)- Multi head Turing Machine :-

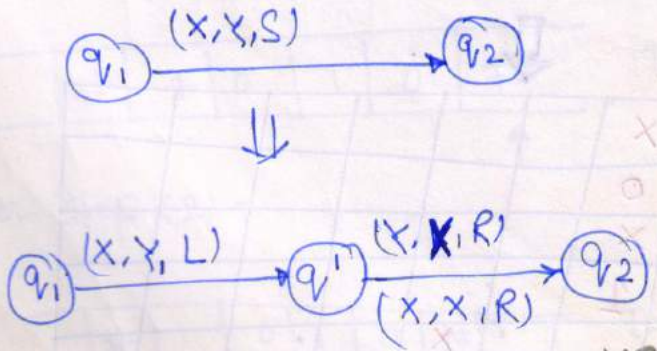
In multi head TM, we have K (K>0) numbers of heads on the same tape and a Move of the TM depends on the state and on all the symbols scanned by each head.



5)- Stay option Turing Machine :-

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$

⇒ The Turing machine with stay option is equivalent to standard TM. ie the stay option does not extend the power of T.M.



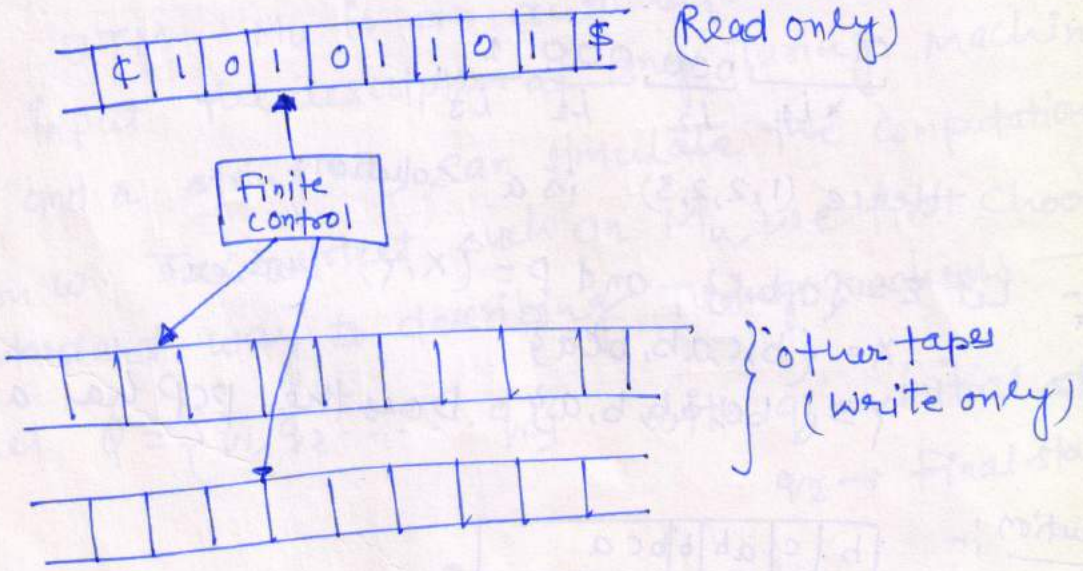
(6)- Multidimensional Turing Machine :-

The K-dimentional Turing machine has finite control, tape head and a tape with more than one dimension. The transition function δ is defined as follows-

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, U, D\}$$

7) - Offline - Turing machine :-

An off-line turing machine is a special case of multi-tape turing machine in which the input tape is made read only.



POST - CORRESPONDENCE PROBLEM :-

An instance of PCP consists of two lists, $X = (x_1, x_2 \cdots x_n)$ and $Y = (y_1, y_2 \cdots y_n)$ of strings over some alphabet Σ. This instance of PCP has a solution if there is any sequence of integers $i_1, i_2 \cdots i_m$ with $m \geq 1$, such that

$$x_{i_1} x_{i_2} \cdots x_{i_m} = y_{i_1} y_{i_2} \cdots y_{i_m}$$

The sequence $(i_1, i_2 \cdots i_m)$ is a solution to this instance of PCP.

ex:- $X = \{01000, 0, 01\}$, $Y = \{01, 000, 1\}$. Does the PCP has a solution.

Solution:-

01000 · 0 0 01

01 | 000 | 000 | 1

i_1 i_2 i_2 i_3

Hence (1,2,2,3) is a solution.

ex- Let $\Sigma = \{a, b, c\}$ and $P = (X, Y)$ where

$X = \{b, c, ab, bca\}$

$Y = \{bc, ab, b, a\}$. Does the pcp has a solution.

Solution:-

b	c	ab	b	bca
x_1	x_2	x_3	x_1	x_4

bc	ab	b	bc	a
y_1	y_2	y_3	y_1	y_4

Hence (1, 2, 3, 1, 4) is a solution.

ex- Let $X = \{ab, ba, b, abb, a\}$ & $Y = (aba, abb, ab, b, bab)$

Does the PCP has a solution.

UNIVERSAL TURING MACHINE :-

As Church thesis " Turing machines are as much powerful as much digital computer". But once δ is defined, the machine is restricted to carrying out one particular type of computation. Digital computer on the other hand are general purpose machine that can be programmed to do different jobs at different times. This objection can be overcome by designing a reprogrammable Turing machine, called a "Universal Turing machine".

A UTM M_u is an automaton that, given as input the description of any Turing machine M and a string w, can simulate the computation of M on w. To construct such an M_u, we first choose a standard way to describing Turing machines —

Let $Q = \{q_1, q_2 \cdots \cdots q_n\}$ with $q_1 \rightarrow$ initial state
$q_2 \rightarrow$ Final state

$$\Gamma = \{a_1, a_2 \cdots \cdots a_m\},$$

a_1 ↓ Blanck

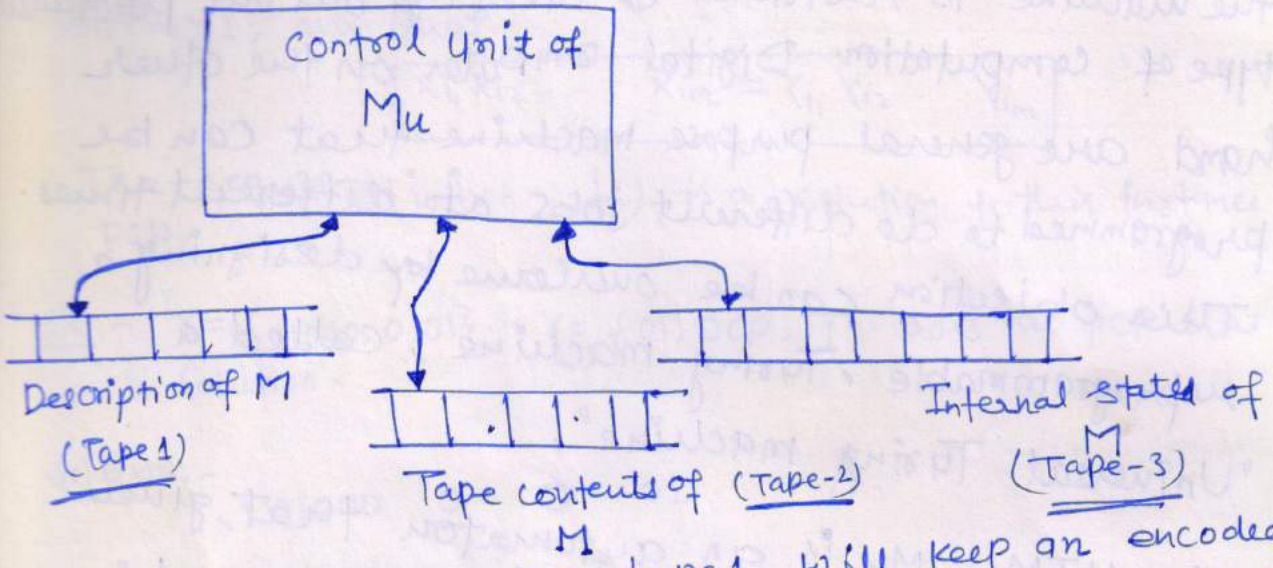
We then select an encoding in which q_1 is represented by 1, q_2 is represented by 11 and so on. Similarly a_1 is encoded as 1 and a_2 as 11 etc. The symbol 0 will be used as a separator between 1's.

Left → 1
Right → 0

for example - $\delta(q_1, a_2) = (q_2, a_3, L)$ might appear as -

....10110 110 1110 10 ....

A UTM M_u, then has an input alphabet that include $\{0,1\}$.



For any input M and w, tape 1 will keep an encoded definition of M. Tape-2 will contain the tape contents of M, and tape-3 the internal state of M. Mu looks first at the content of tape-2 & tape 3 to determine the configuration of M. It then consults tape-1 to see what M would do in this configuration. Finally ~~state~~ tape 2 and 3 will be modified to reflect the result of the move.

CHURCH THESIS

Suppose we were to make the conjecture that, in some sense, Turing machines are equal in power to a typical digital computer? How could we defend or refute such a hypothesis?

To defend it, we could take a sequence of increasingly more difficult problems and show that how they are solved by some Turing machine.

we might try to find some procedure for which we can write a computer program, but for which we can show that no Turing machine can exists. If this were possible, we would have a basis for rejecting the hypothesis. But no one has yet been able to produce a counter example, the fact that all such tries have been unsuccessful must be taken as circumstantial evidence that it cannot be done.

⇒ "Every indication is that "Turing machine are in principle as powerful as any computer".

⇒ "The hypothesis states that any computation that can be carried by mechanical means can be performed by some Turing machine".

A computation is mechanical if and only if it can be performed by some Turing machine.

RECURSIVE & RECURSIVELY ENUMERABLE LANGUAGES

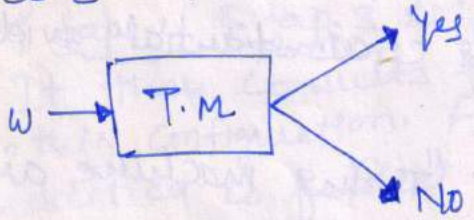
TURING DECIDABLE LANGUAGES :-

A T.M M decides L if M computes the characteristic function $F(w) = \begin{cases} 0 & \text{if } w \notin L \\ 1 & \text{if } w \in L \end{cases}$

ie. A language L is turing decible if there exists a Turing machine M that accepts L and that halts on every w in Σ^* producing output 1 if $w \in L$ and 0 otherwise. ie There exists a membership algorithm for L.

1. Recursive language :-

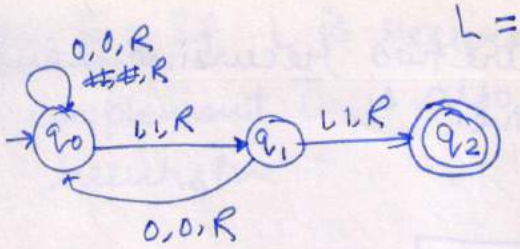
A language is recursive if there is a T.M that decides L. ie.



2). Recursive enumarable language :-

A language L is r.e if there exists a T.M that accepts L. ie a language L is r.e if there is a T.M that accepts words in L and either rejects or loop forever for every words in the complement of L.

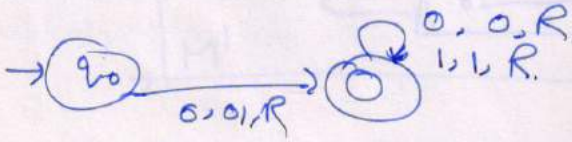
ex -1- $L = (0+1)^* 11 (0+1)^*$



if $w = 11$ accepts
$w = 00$ Reject (loop for ever)
$w = 101$ Reject.

Hence the language L is recursive enumerable.

2) -

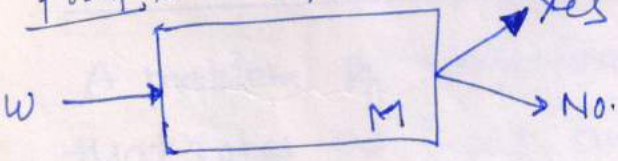


Accepts all words start with 0 & reject all that do not. Hence the language is recursive.

Properties of R.E & Recursive languages :-

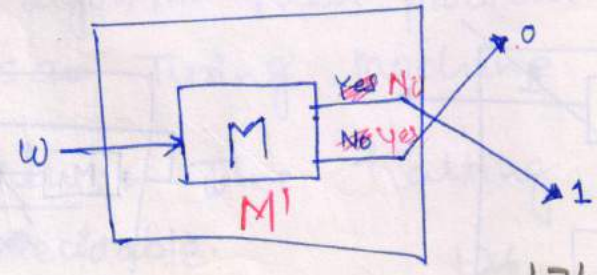
Theorem: If the language L is recursive, then $\bar{L}$ is also recursive.

Proof:- If L is recursive the there is a T.M- M



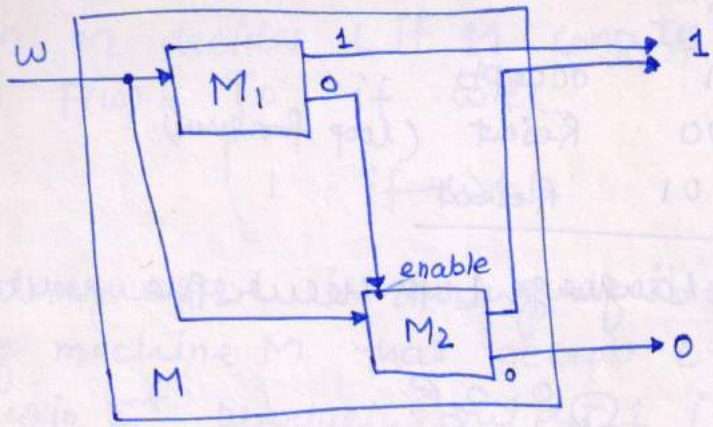
Let M' be the turing machine that accepts $\bar{L}$

M	output of M	M'	output of M'
w	1	x	0
x	0	w	1



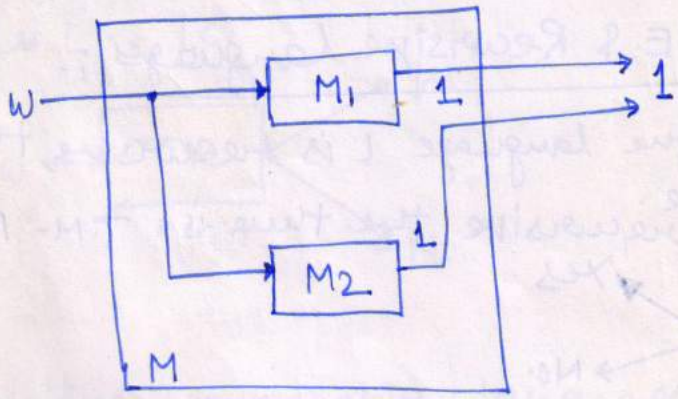
Theorem-2 :- If L_1 & L_2 are two recursive languages then $L_1 \cup L_2$ is recursive.

Proof :-



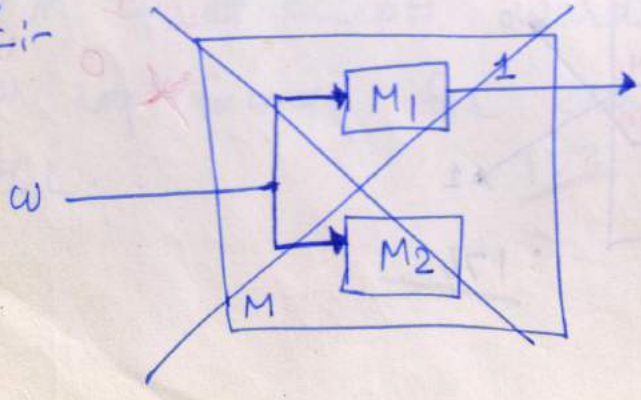
Theorem-3 :- If L_1 & L_2 are two recursive enumerable languages then $L_1 \cup L_2$ is also recursive enumerable.

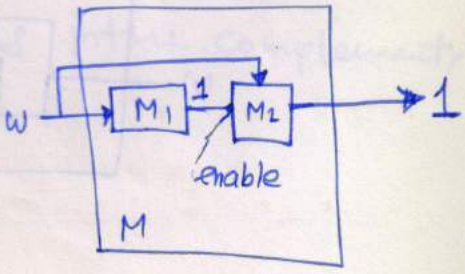
Proof :-



Theorem-4 :- If L_1 & L_2 are R.e then $L_1 \cap L_2$ is also recursive ~~enumerable~~ enumerable.

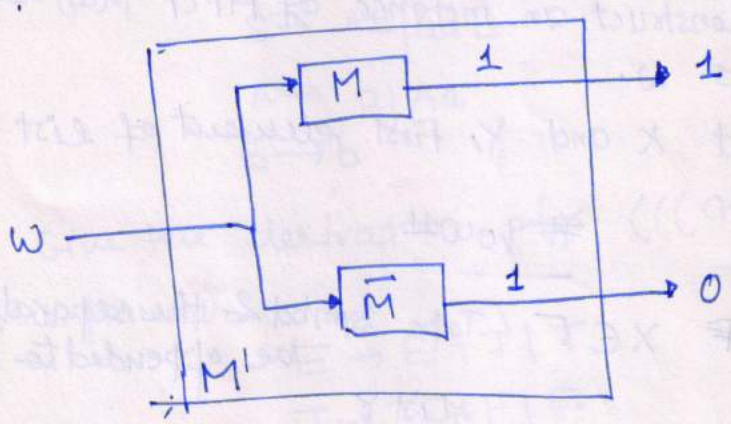
Proof :-





Theorem-5 : If L is recursive enumerable language whose complement $\bar{L}$ is also recursive enumerable then L is recursive.

Proof :



Halting problem of turing machine :-

Suppose we are given an input string w and a Turing machine M. Can we tell whether or not M halts on w?

Undecidable problem :-

A problem is undecidable if there is no algorithm that takes as input an instance of the problem and determines whether the answer to that instance is "yes" or "no". If there exists any such algorithm then problem is "decidable". ie there exists a Turing machine to decide it.

Theorem :- The halting problem of turing machine is undecidable.

Theorem:- PCP is undecidable.

proof:- It is sufficient to show that if MPCP were decidable then it would be decidable whether a TM accepts a given word. That is we reduce TM to MPCP, which is reduced to PCP. for each M and w we construct an instance of MPCP that has a solution if M accepts w.

1)- create two list X and Y, First element of list X & Y are

List X	List Y
#	$\#q_0w\#$

2)- for each ~~$x \in \Gamma$~~ $X \in \Gamma$ (Tape symbol & the separator # can be appended to both lists)

List X	List Y
X	X
#	#

3)- for each $q \in Q-F$, $p \in Q$. $X, Y, Z \in \Gamma$

List X	List Y	
qX	Yp	if $\delta(q,X) = (p,Y,R)$
ZqX	pZY	if $\delta(q,X) = (p,Y,L)$
q#	Yp#	if $\delta(q,\#) = (p,Y,R)$
Zq#	pZY#	if $\delta(q,\#) = (p,Y,L)$

4)- for each $q \in F$, and X and $Y \in \Gamma$

List X	List Y
XqY	q
Xq	q
qY	q

5) for each $q \in F$

List X	List Y
q##	#

Ex-

List A	List B
#	$\#q_1 01\#$
0	0
1	1
#	#
$q_1 0$	$1q_2$
$0q_1 1$	$q_2 00$
$1q_1 1$	$q_2 10$
$0q_1\#$	$q_2 01\#$
$1q_1\#$	$q_2 11\#$
$0q_2 0$	$q_3 00$
$1q_2 0$	$q_3 10$
$q_2 1$	$0q_1$
$q_2\#$	$0q_2\#$
$0q_3 0$	q_3
$0q_3 1$	q_3
$1q_3 0$	q_3
$1q_3 1$	q_3
$0q_3\#$	q_5
$1q_3\#$	q_3
$q_3 0$	q_3

Ex-

	0	1	#
→ q_1	$(q_2, 1, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$
q_2	$(q_3, 0, L)$	$(q_1, 0, R)$	$(q_2, 0, R)$
(q_3)	—	—	—

	#
$q_3\#\#$	
$q_3 1$	q_3

EXERCISE

Question – 1 :- Show that the following grammar is ambiguous.

$S \rightarrow AB \mid aaB$

$A \rightarrow a \mid Aa$

$B \rightarrow b$

2) – Give the derivation for $(((a+b)*c)) + a+b$, using the grammar.

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid a \mid b \mid c$

3) – Show that the grammar in exercise-2 is unambiguous.

4) – Find the reduced grammar –

$S \rightarrow abAB \mid ba$

$A \rightarrow aaa$

$B \rightarrow aA \mid bb$

5) – Eliminate useless productions from –

$S \rightarrow a \mid aA \mid B \mid C$

$A \rightarrow aB \mid \epsilon$

$B \rightarrow Aa$

$C \rightarrow cCD$

$D \rightarrow ddd$

6) – Convert the grammar with productions –

$S \rightarrow ABa$

$A \rightarrow aab$

$B \rightarrow Ac$

to CNF.

7)- Transform the grammar with productions -

$$S \to abAB$$
$$A \to bAB \mid \epsilon$$
$$B \to BAa \mid A \mid \epsilon$$

into Chomsky normal form.

8)- Convert the grammar -

$$S \to asb \mid bSa \mid a \mid b$$

into GNF.

9)- Convert the grammar into GNF-

$$S \to ABb \mid a$$
$$A \to aaA \mid B$$
$$B \to bAb$$

10)- Determine whether the string $w = aabbb$ is in the language generated by the grammar -

$$S \to AB$$
$$A \to BB \mid a$$
$$B \to AB \mid b$$

11)- Construct NPDA for the following languages -

1)- $L = \{a^n b^{n+m} c^m : n \geq 0, m \geq 1\}$

2)- $L = \{a^3 b^n c^n : n \geq 0\}$

3)- $L = \{a^n b^m : n \leq m \leq 3n\}$

4)- $L = \{w : n_a(w) + n_b(w) = n_c(w)\}$

12)- Find the NPDA for the language

$$L = \{ab(ab)^n b(ba)^n : n \geq 0\}$$

13)- Find an NPDA on $\Sigma = \{a, b, c\}$ that accepts the language-

$$L = \{w_1 c w_2 : w_1, w_2 \in \{a, b\}^*, w_1 \neq w_2^R\}$$

14)- What language is accepted by the PDA -

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a, b\}, \{0, 1, a\}, \delta, Z_0, q_0, \{q_5\})$,

with

$\delta(q_0, b, Z_0) = \{(q_1, 1Z_0)\}$

$\delta(q_1, b, 1) = \{(q_1, 11)\}$

$\delta(q_2, a, 1) = \{(q_3, \epsilon)\}$

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

$\delta(q_4, a, Z_0) = \{(q_4, Z_0), (q_5, Z_0)\}$?

15)- Construct an NPDA corresponding to the grammar -

$S \rightarrow aABB \mid aAA$

$A \rightarrow aBB \mid a$

$B \rightarrow bBB \mid A$

16)- Find an NPDA with two states that accepts

$L = \{a^n b^{2n} : n \geq 1\}$

17)- Find a context-free grammar that generates the language accepted by the PDA :

$M = (\{q_0, q_1\}, \{a, b\}, \{A, Z\}, \delta, q_0, Z, \{q_1\})$,

with the transitions -

$\delta(q_0, a, Z) = \{(q_0, AZ)\}$

$\delta(q_0, b, A) = \{(q_0, AA)\}$

$\delta(q_0, a, A) = \{(q_1, \epsilon)\}$

18)- Explain DPDA. Prove that the language

$L = \{a^n b^n : n \geq 0\}$ is deterministic

context free language.

19) - prove that the following languages are not context free - language -

1) - $L = \{a^n b^n c^n : n \geq 0\}$

2) - $L = \{a^{n!} : n \geq 0\}$

3) - $L = \{a^n b^j : n = j^2\}$

4) - $L = \{a^p : p \text{ is prime}\}$

5) - $L = \{a^n b^m : n \text{ and } m \text{ are both prime}\}$.

20) - Prove that the family of context free languages are not closed under intersection and complementation.

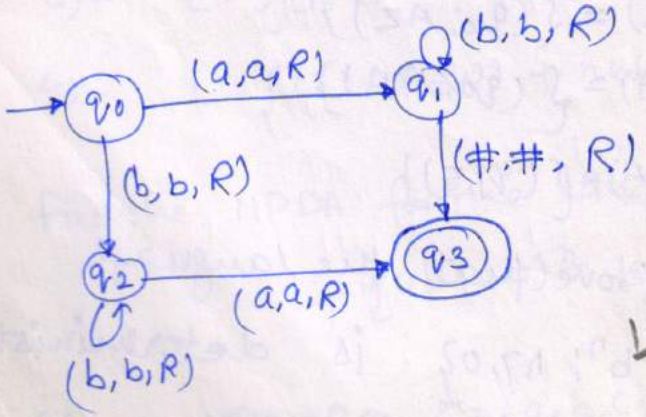
21) - Show that the following language is context-free.

$L = \{w \in \{a, b\}^* : n_a(w) = n_b(w)$; w contains aab as a substring$\}$.

22) - Show that the family of context-free languages are closed under union, concatenation, and Kleene closure.

23) - Design a Turing machine with no more than three states that accepts the language $L(a(a+b)^*)$.

24) - What language is accepted by the Turing machine -



25)- Construct Turing machines that will accepts the following languages on $\Sigma = \{a, b\}$

(a) - $L = \{w : |w| \text{ is even}\}$

(b) - $L = \{a^n b^m a^{n+m} : n \geq 0, m \geq 1\}$

(c) - $L = \{a^n b^{2n} : n \geq 1\}$

(d) : $L = \{ww : w \in \{a, b\}^+\}$

(e) - $L = \{a^n b^n a^n b^n : n \geq 0\}$

26) - Construct a Turing machine to compute the function $f(w) = wR$, $\Sigma = \{0, 1\}$.

27)- Design Turing machine to compute the following functions for x and y possitive integers represented in unary -

1)- $f(x) = 3x$

2)- $f(x,y) = 2x + 3y$

3)- $f(x) = \frac{x}{2}$ if x is even

$= \frac{x+1}{2}$ if x is odd.

4)- $f(x) = x \bmod 5$.

5)- $f(x) = 2^x$

6)- $f(x) = x!$

28)- Write short notes on Church thesis.

29)- Design a Non-deterministic Turing machine for the following language- $L = \{ww : w \in \{a,b\}^+\}$

30)- Explain universal Turing machine.

31)- Give a complete encoding for the Turing machine with

$\delta(q_1, a_1) = (q_1, a_1, R)$

$\delta(q_1, q_2) = (q_3, a_1, L)$

$\delta(q_3, a_1) = (q_2, a_2, L)$

32)- Design Two-stack PDAs for the following languages-

1)- $L = \{a^n b^n c^n : n \geq 1\}$

2)- $L = \{a^n b^n c^n d^n : n \geq 1\}$

3)- $L = \{a^n b^n a^n b^n : n \geq 1\}$

4)- $L = \{a^n b^{2n} c^n : n \geq 1\}$.

33)- Explain Chomsky-hierarchy.

34)- "Halting problem of Turing machine is undecidable" explain.

35)- Let $X = \{11, 100, 111\}$

$Y = \{111, 001, 11\}$

does the PCP has a solution?.

36)- Let $X = \{001, 0011, 11, 101\}$

$Y = \{01, 111, 111, 010\}$. Does the pair (X, Y) have a solution?. Does it have an MPCP solution?

37)- "The Modified post correspondence problem is undecidable" explain.

3)- MODIFIED POST CORRESPONDENCE PROBLEM

Given two lists $X = (x_1, x_2 \cdots x_n)$ and $Y = (y_1, y_2 \cdots y_n)$ does there exists a sequence of integers $i_1, i_2 \cdots i_k$ such that

$$x_1 x_{i_1} x_{i_2} \cdots x_{i_k} = y_1 y_{i_1} y_{i_2} \cdots y_{i_k} \; ?$$

The difference between the MPCP and PCP is that in the MPCP, a solution is required to start with the first string on each list.

Theorem:- IF PCP were decidable, then MPCP would be decidable. That is MPCP reduces to PCP.

Proof:- Let $X = (x_1, x_2 \cdots x_n)$ and $Y = (y_1, y_2 \cdots y_n)$ be an instance of MPCP. We convert this instance of MPCP to an instance of PCP.

⇒ create new list by adding new words -

List A	List B
$w_0 = \$ w_1$	$z_0 = z_1$
$w_{k+1} = \$$	$z_{k+1} = \$\$$

→ and w_i is obtained by adding symbol $ after each character of x_i for $i \geq 1$

→ z_i is obtained by adding symbol $ before each character of y_i for $i \geq 1$

⇒ if $(1, i_1, i_2 \cdots i_k)$ is a solution of MPCP then $(0, i_1, i_2 \cdots i_k, k+1)$ is a solution of PCP.

ex:- Let $\Sigma = \{0, 1\}$. Let $X = \{1, 10111, 10\}$ & $Y = \{111, 10, 0\}$

Solution:-

i	List X: x_i	List Y: y_i
1	1	111
2	10111	10
3	10	0

i	List A: w_i	List B: z_i
0	¢1¢	¢1¢1¢1
1	1¢	¢1¢1¢1
2	1¢0¢1¢1¢1¢	¢1¢0
3	1¢0¢	¢0
4	$	¢$

ex- Find the instance of PCP for the following instance of MPCP.

$X = (101, 0101, 001, 0)$

$Y = (10, 1101, 001, 00)$

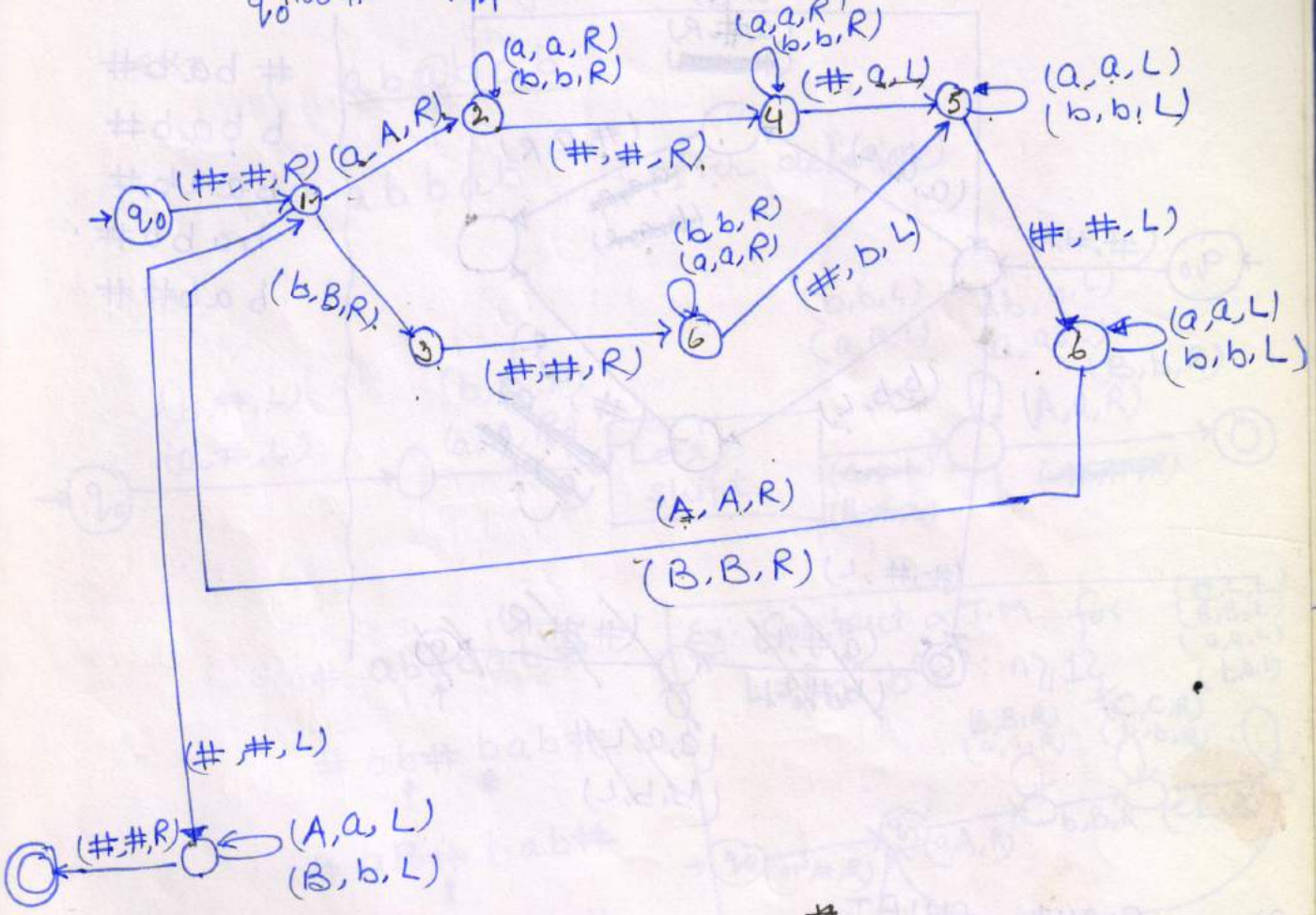
EX- Let $X = \{11, 100, 111\}$ & $Y = \{111, 001, 11\}$ does the MPCP has a solution?

Ans - (1, 2, 3)

EXAMPLES ON TURING MACHINE

1)- Construct a Turing machine to copy a string-

ex- $q_0 \#w\# \overset{*}{\underset{M}{\vdash}} q\#w\#w\#$



ex = #aba##

#Aba##

#Aba#a

#ABa#a

#ABa#ab

#ABA#ab

#ABA#aba

#ABa#aba

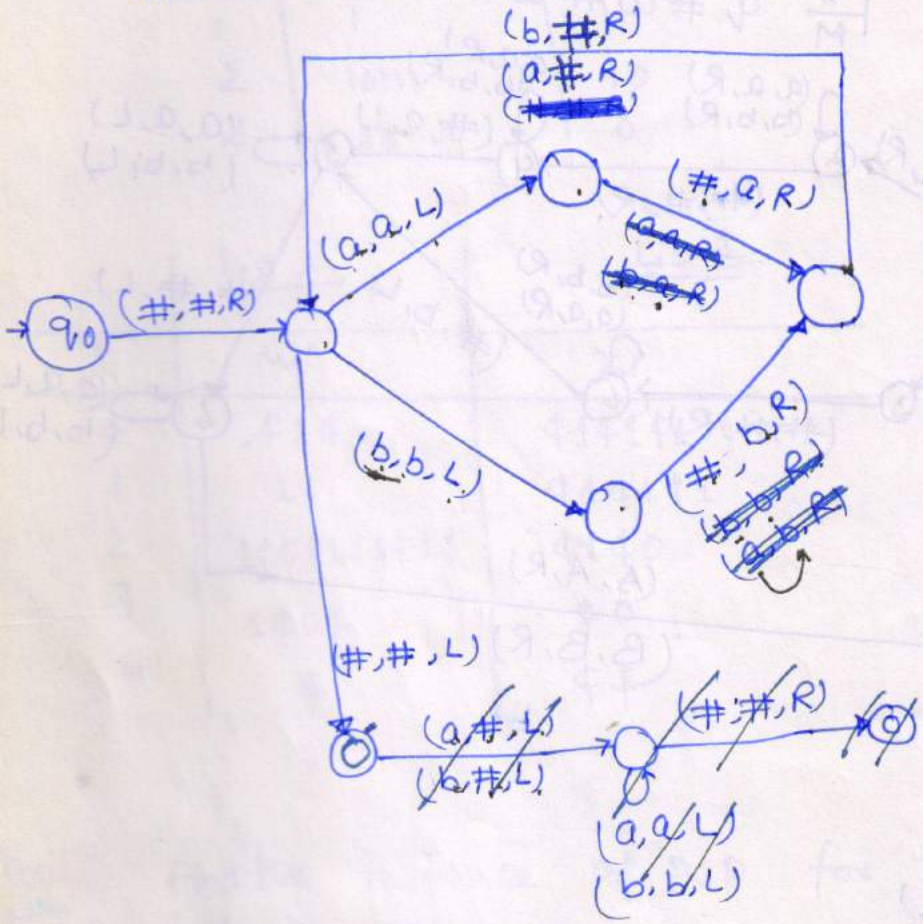
#Aba#aba

#aba#aba (Halt).

2)- LEFT SHIFTING :-

ex- ##baaba# (Before)

#baaba## . (After left shift).

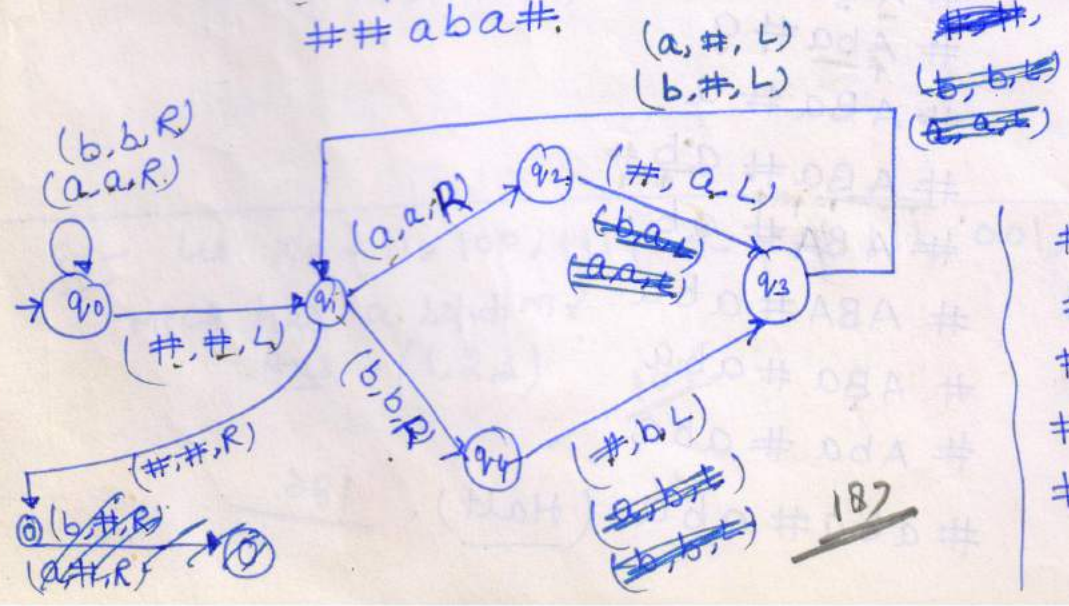


#bab#
bbab#
baab#
babb#.
bab##

3)- RIGHT SHIFT :-

ex- #ab@# :.

##aba#.



#bab#
#babb
#baab
#bbab
##bab

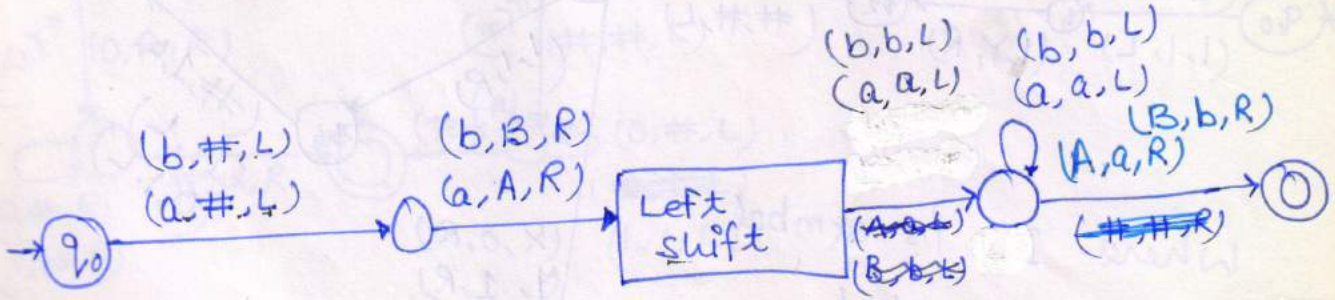
187

4)- Deleting a symbol

Sometimes in the running of a TM we may wish to delete a character from the string on the tape exactly from the currently pointing to the symbol.

ex- a b @ b a b

a b b a b (After deletion).



a b a b a b

a b # b a b

a B # b a b

a B b b a b

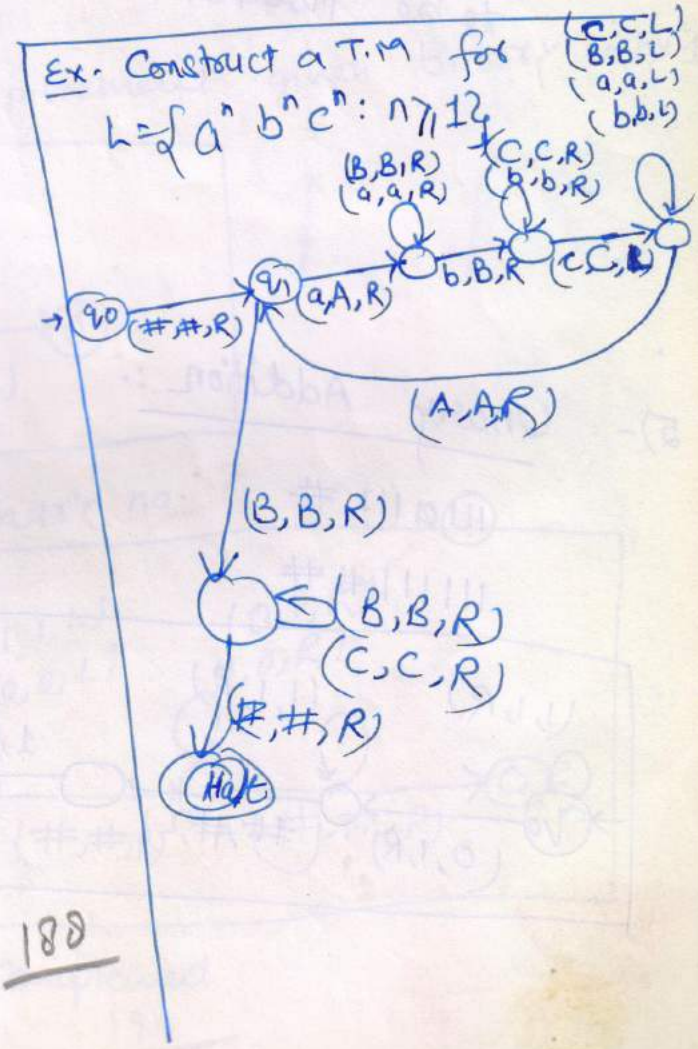
a B b a a b

a B b a b b

a B b a b #

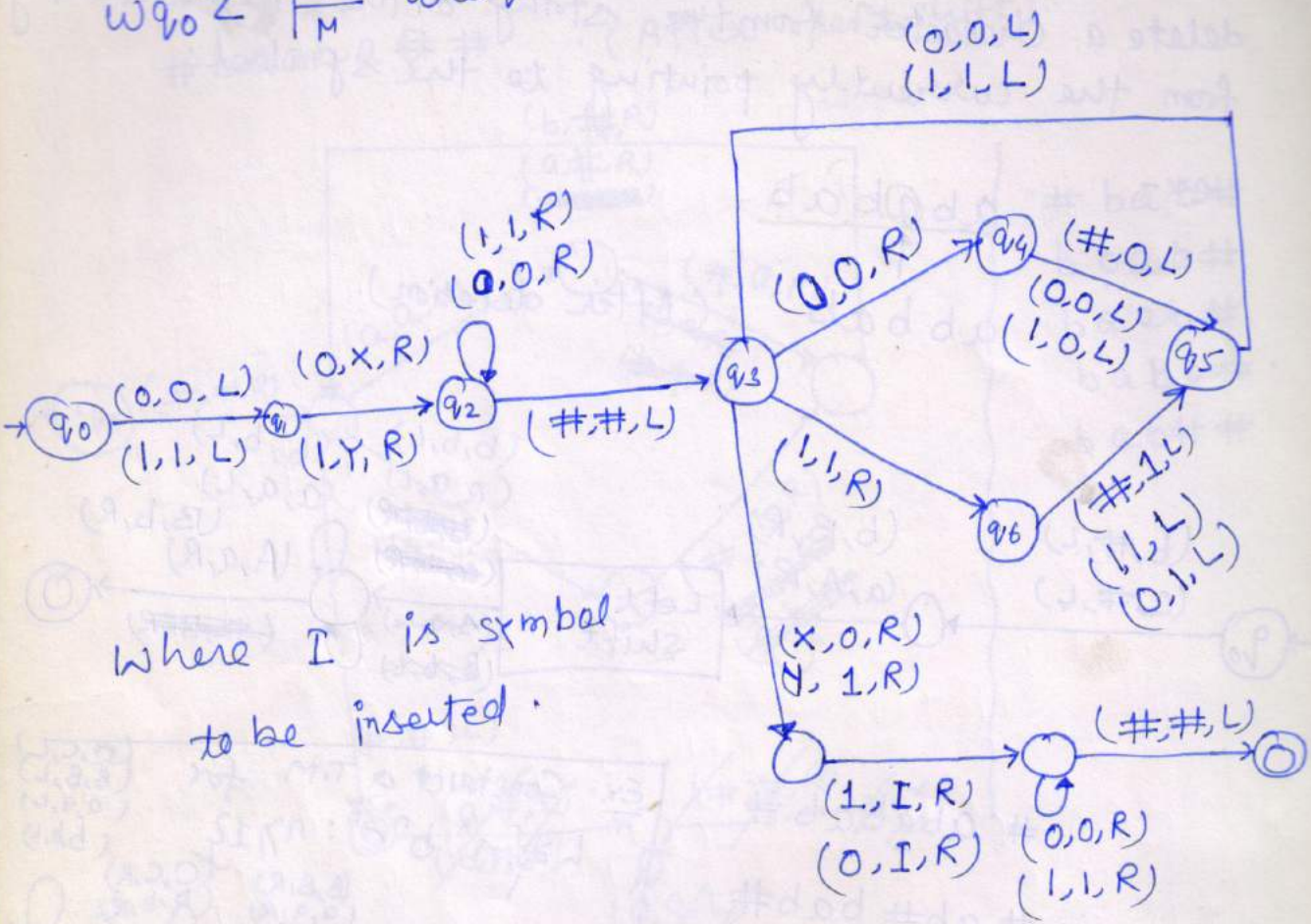
a b b a b #

Ex. Construct a T.M for $L = \{a^n b^n c^n : n \geq 1\}$

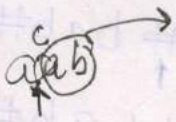


Inserting a symbol :-

$wq_0z \vdash_M^* waqz$ where $w, z \in \Sigma^*, a \in \Sigma$.



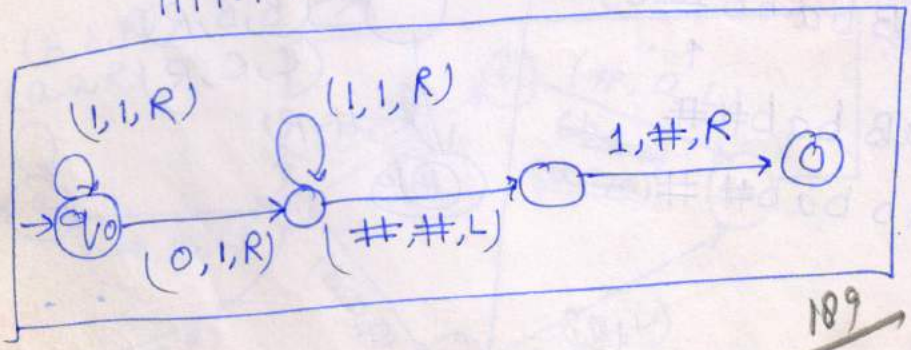
Where I is symbol to be inserted.



5)- Unary Addition :-

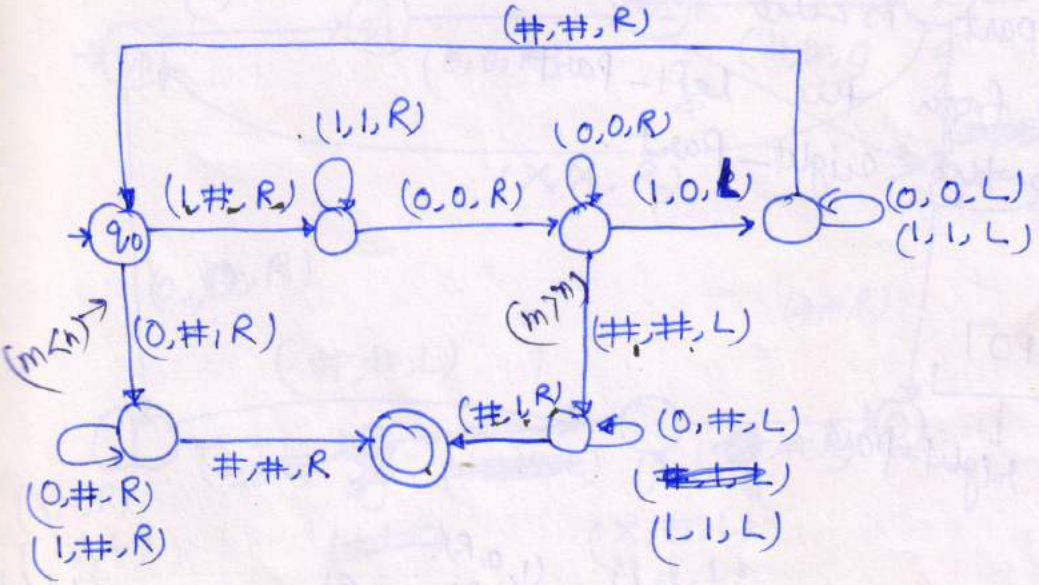
(111)a(11) #

111111 ##

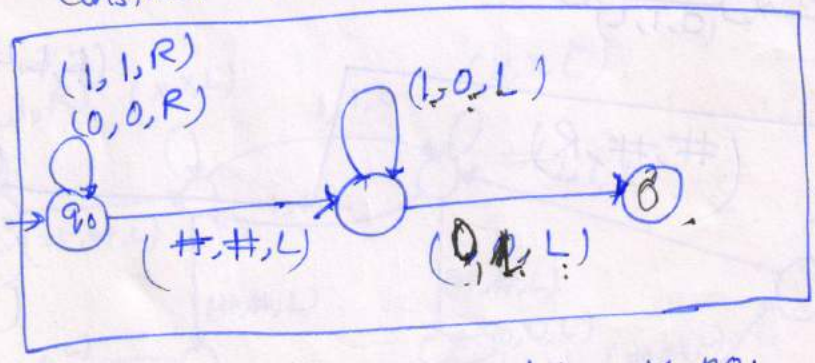


(6) - proper subtraction :-

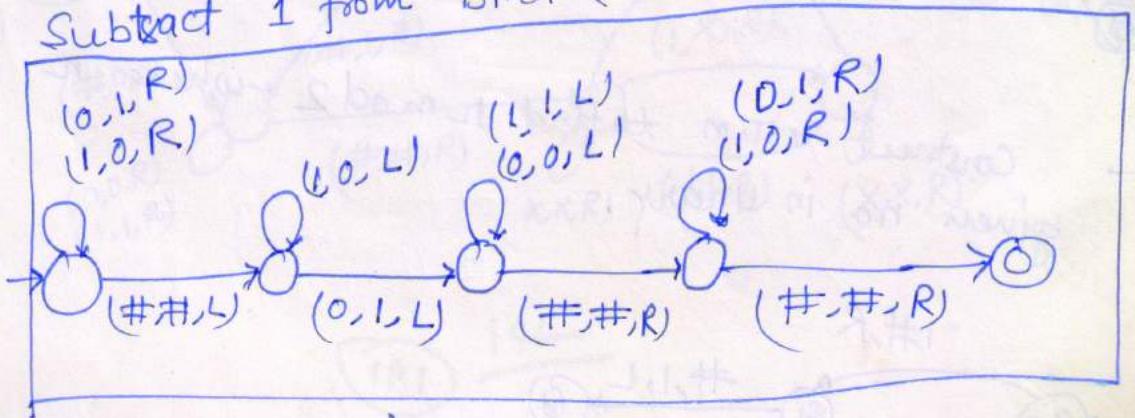
$$m \dot{-} n = \begin{cases} m-n & \text{if } m \geq n \\ 0 & \text{if } m \leq n \end{cases}$$



(7). Construct a T.M which increment given binary no. by 1



8)- Subtract 1 from binary no.



(1's complement + 1) 1's complement

9)- Addition of two binary numbers :-

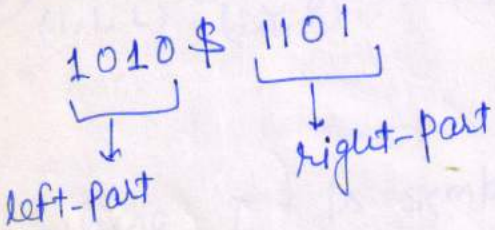
Let the binary nos. are written in tape and separated by $.

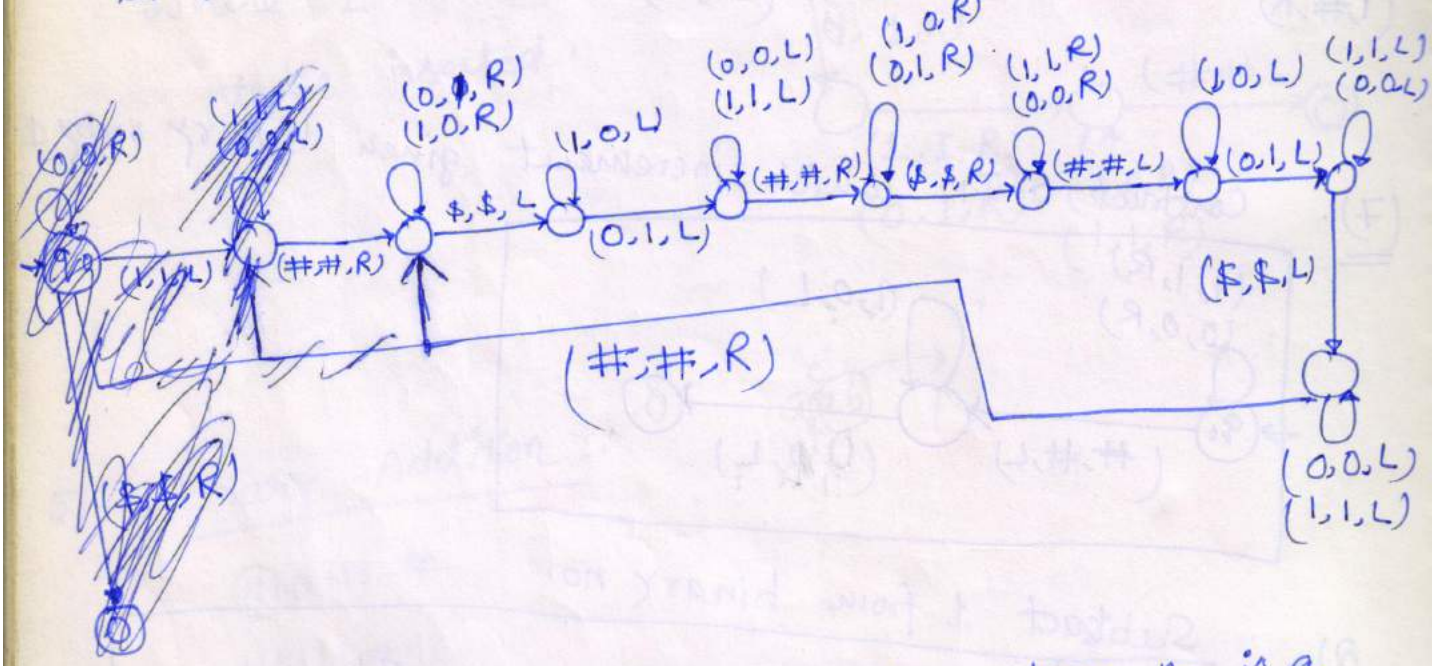
1)- If Left-part is zero then Halt else goto step-2

2)- Subtract 1 from the left-part.

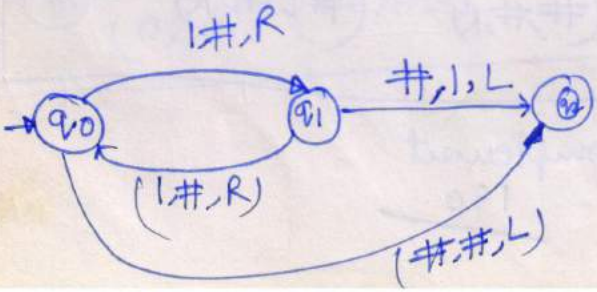
3)- Add 1 to the right-part.

4)- goto step 1.

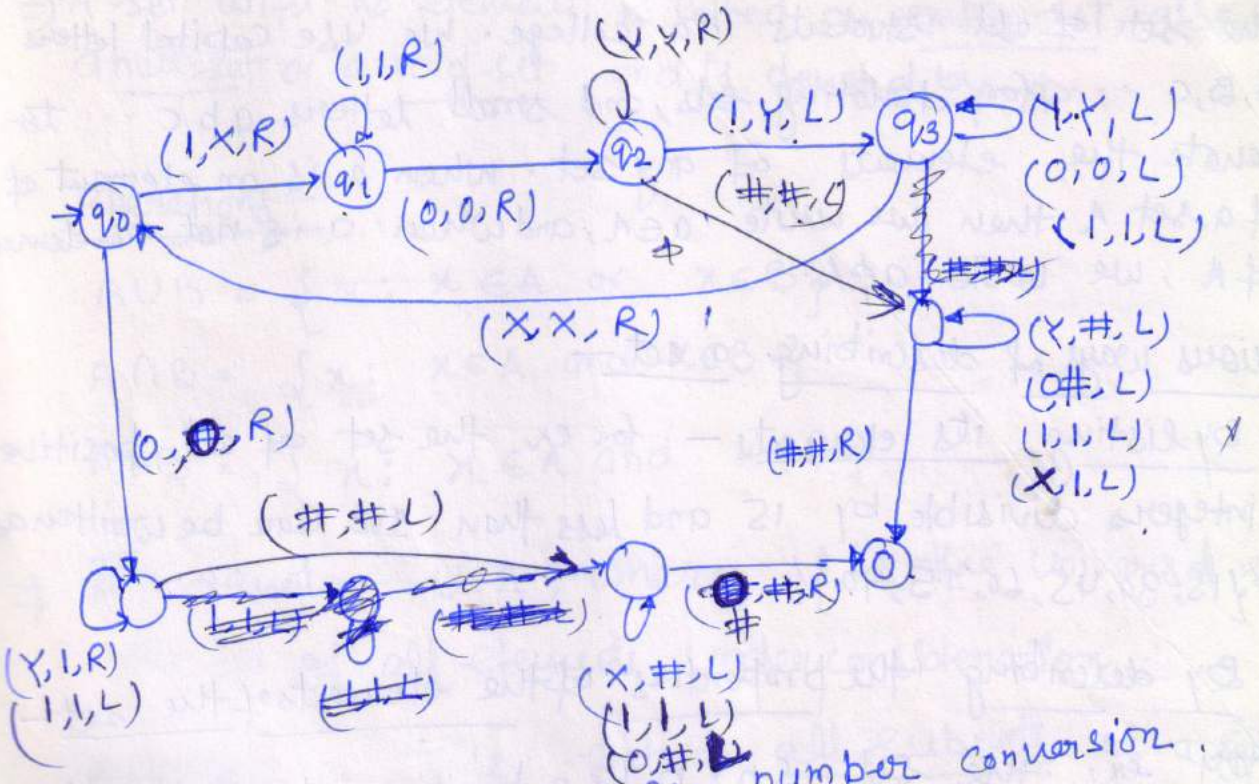




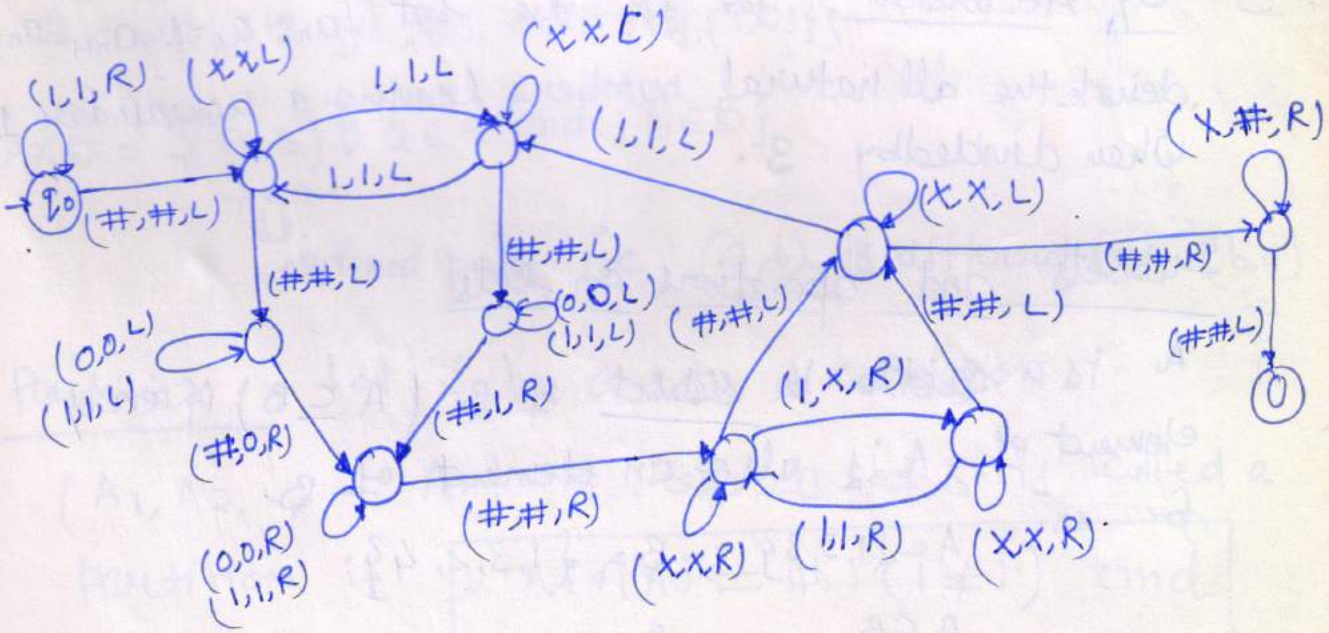
10)- Construct a T.M to find n mod 2. where n is a given no in unary.



11)- $Max(m,n) = \begin{cases} m & \text{if } m > n \\ n & \text{if } m \leq n \end{cases}$



12)- UNARY to BINARY number conversion.



SETS, RELATIONS, AND FUNCTIONS

SETS :- A set is a well-defined collection of objects, for ex. the set of all students in a college. We use capital letters A, B, C ... for denoting sets, and small letters a, b, c ... to denote the elements of any set. When a is an element of a set A then we write $a \in A$, and when a is not an element of A, we write $a \notin A$.

Various ways of describing a set—

1- By listing its elements — for ex. the set of all positive integers divisible by 15 and less than 100 can be written as $\{15, 30, 45, 60, 75, 90\}$

2- By describing the properties of the elements of the set — for ex, the set $\{n : n$ is a positive integer divisible by 15 and less than $100\}$ describe the set $\{15, 30, 45, 60, 75, 90\}$

3- By recursion — for ex, the set $\{a_n : a_0 = 1, a_{n+1} = a_n + 3\}$ denote the all natural numbers leaving a remainder 1 when divided by 3.

Subsets and operations on sets

A is a said to be subset of B ($A \subseteq B$) if every element of A is also an element of B.

for ex. $A = \{1, 2, 3\}$, $B = \{1, 3, 2, 4\}$

$A \subseteq B$

⇒ Two sets A & B are equal ($A = B$) if their members are the same.

To prove that $A = B$, we prove $\boxed{A \subseteq B \text{ and } B \subseteq A}$

⇒ A set with no element is called an empty set, also called a null set or a void set and is denoted by ϕ

Operations –

$A \cup B = \{x : x \in A \text{ or } x \in B\}$

$A \cap B = \{x : x \in A \text{ and } x \in B\}$

$A - B = \{x : x \in A \text{ and } x \notin B\}$

⇒ $\bar{A}$ denotes $U - A$, where U is the universal set, the set of all elements under consideration.

Powerset = The set of all subsets of a set A is called the power set of A.

$A = \{1,2,3\}$

$2^A = \{\phi, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

$A \times B = \{(a,b) : a \in A \text{ and } b \in B\}$

↓ ordered pair ie (a,b) is different from (b,a)

Partition : Let S be a set. A collection $(A_1, A_2, \ldots A_n)$ of subsets of S is called a partition if $\boxed{\begin{array}{l} 1) \; A_i \cap A_j = \phi \quad (i \neq j) \text{ and} \\ 2) \; S = \bigcup_{i=1}^{n} A_i \end{array}}$

ex – $S = \{1,2,3,4,5,6,7,8,9,10\}$ then $A_1 = \{2,4,6,8,10\}$

$A_2 = \{1,3,5,7,9\}$

are the partitions of S.

Relations :-

A relation R in a set S is a collection of ordered pairs of elements in S (ie a subset of $S \times S$). When (x, y) is in R, we write xRy. When (x, y) is not in R, we write $xR'y$.

properties of Relations

(i) A relation R in S is reflexive, if xRx for every x in S.

(ii) A Relation R in S is symmetric if for x, y in S, yRx whenever xRy.

(iii) - A relation R in S is transitive if for x, y and z in S, xRz whenever xRy and yRz.

⇒ A relation R in a set S is called an equivalent relation if it is reflexive, symmetric and transitive.

ex - If i, j, n are integers we say that i is congruent to j modulo n (written as $i \equiv j \bmod n$) if $(i-j)$ is divisible by n. The 'Congruence modulo n' is a relation which is reflexive and symmetric (if $i-j$ is divisible by n, so is $j-i$).

If $i \equiv j \bmod n$ and $j \equiv k \bmod n$, then we have $i-j = an$ for some a and $j-k = bn$ for some b. So,

$$i-k = i-j+j-k = an+bn = (a+b)n$$

which means $i \equiv k \bmod n$. So the relation is equivalence

Function :- A function or map f from a set X to a set Y is a rule which associates to every element x in X a unique element in Y, which is denoted by $f(x)$. The element $f(x)$ is called the image of x under f.

$$\boxed{f : X \to Y}$$

eg- $f : R \to R$ where $f(x) = x^2 + 2x + 1$ for every x in R (R denotes the set of real numbers).

⇒ $f : X \to Y$ is said to be one-to-one (or injective) if different elements in X have different images ie $f(x_1) \neq f(x_2)$ when $x_1 \neq x_2$.

⇒ $f : X \to Y$ is onto (surjective) if every element y in Y is the image of some element x in X.

⇒ $f : X \to Y$ is said to be a one-to-one correspondence or bijection if f is both one-to-one and onto.

ex. $f : Z \to Z$ given by $f(n) = 2n$ is one-to-one but not onto.

Suppose $f(n_1) = f(n_2)$. Then $2n_1 = 2n_2$

$$\Downarrow$$

$$\boxed{n_1 = n_2}$$

Hence one-to-one

⇒ It is not onto since no odd integer can be the image of any element in Z.

STRINGS and THEIR PROPERTIES :-

A string over an alphabet set Σ is a finite sequence of symbols from Σ.

Σ^* denotes the set of all strings (including ϵ, the empty string) over the alphabet set Σ. ie

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

⇒ let x & y are two strings then the concatenation of x & y is denoted by xy is obtained by placing y after x.

eg— $x = 010$ $y = 1$

$x = a\epsilon$ $y = ALGOL$

find xy & yx

xy: 0101, ϵALGOL

yx: 1010, ALGOLϵ

properties of Concatenation :-

1— Concatenation of a set Σ^* is associative since for each x, y, z in Σ^*, $x(yz) = (xy)z$

2- Identity element = ϵ is the identity element w.r.t the binary operation of Concatenation. as -

$$\epsilon a = a\epsilon = a$$

3- $zx = zy \Rightarrow x = y$ (left Cancellation).

$xz = yz \Rightarrow x = y$ (right Cancellation).

4. $|xy| = |x| + |y|$

Transpose operation—

for any x in Σ^* and a in Σ

$$(xa)^T = a(x)^T$$

⇒ palindrome is a string which is same whether written forward or backward. A palindrome of even length can be obtained by concatenating string & its transpose. ie. Let $x = abc$ is a string

abccba is a palindrome.

Prefix :- A prefix of a string is a substring of leading symbols of that string.

let $y = wy'$ w is a prefix of y

String 123 have four prefixes— ϵ, 1, 12, 123

Suffix :- a suffix of a string is a substring of trailing symbols of that string. ie.

$y = y'w$. w is a suffix of y.

string 123 have four suffixes— ϵ, 3, 23, 123

Principle of Induction :-

The process of reasoning from general observations, to specific truths is called induction.

step- prove $P(n)$ for $n = 0/1$. Proof for the basis

2- Assume the result/properties for $P(n)$: Induction Hypothesis.

3 — prove $P(n+1)$ using induction hypothesis.

Ex- prove that $1+3+5+\cdots+r=n^2$ for all $n>0$ <
r is an odd number, n is the number of terms

prove: Let 1) for $n=1$

L.H.S $= 1$, R.H.S $= 1^2 = 1$ True for $r.n=1$

2)- By induction Hypothesis.

$$1+3+5+\cdots+r=n^2$$

$$1+3+5+\cdots+(2n-1)=n^2$$

3)- We have to prove that

$$\underbrace{1+3+5+\cdots+(2n-1)}_{n^2}+\underbrace{r+2}_{r}=(n+1)^2$$

$$\Rightarrow n^2+r+2=n^2+2n-1+2$$

$$\Rightarrow n^2+2n+1$$

$$\Rightarrow (n+1)^2 = R.H.S$$

Ex- prove that $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ proved

Proof by Contradiction

Another powerful technique that often works when everything else fails. suppose we want to prove that some stmt P is true. We then assume, for the moment, that P is false and see where that assumption leads us. If we arrive at a conclusion that we know is incorrect, we can lay the blame on the starting assumption and conclude that P must be true.

Ex:- A rational number is a no. that can be expressed as the ratio of two integers m and n so that n and m have no common factor. A real no. that is not rational is said to be irrational.

⇒ Show that $\sqrt{2}$ is irrational. 199

As in all proofs by contradiction, we assume the contrary of what we want to show.

Here we assume that $\sqrt{2}$ is a Rational No. so that it can be expressed as—

$\sqrt{2} = \frac{n}{m}$ where n & m have no common factors.

$$2 = \frac{n^2}{m^2}$$

$$2m^2 = n^2$$

Therefore n^2 must be even. This implies that n is even, so that we can write

$$n = 2k \text{ . for some } k$$

$$2m^2 = 4k^2$$

$$m^2 = 2k^2$$

This implies m^2 is even. But this contradict our assumption that n & m have no common factors. Thus m, & n cannot exist and $\sqrt{2}$ is not a rational number.

ex - Show that $\sqrt{3}$, $\sqrt{8}$, and $2-\sqrt{2}$ are ~~irrel~~ irretional numbers.

Automaton :- An automaton is defined as a system where energy, materials and information are transformed, transmitted and used for performing some functions without direct participation of man.

Characteristics :-

1) Input - At each discrete instance of time input values are applied to the automaton

2- Output → $O_1, O_2 \cdots O_q$ are the outputs of the model.

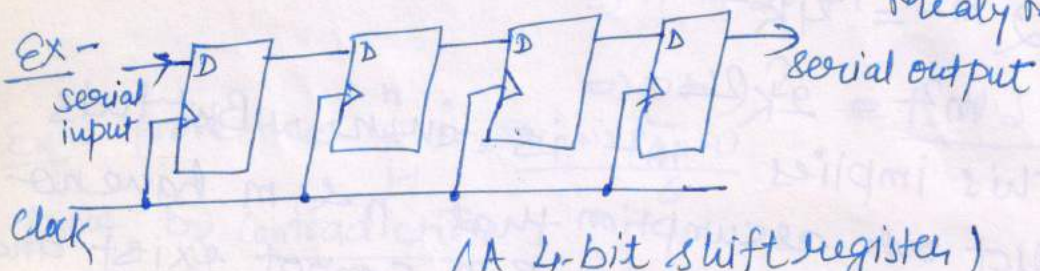
3- States - At any instant of time the automaton can be in one of the states $q_1, q_2 \cdots q_n$.

4- State relation - The next state of an automaton at any instant of time is determined by the present state & present input.

5- Output relation - The output is related to either state only or to both the input and the state

- state only → automaton without memory → Moore machine
- input and the state → automaton with finite memory → Mealy machine.

Ex -



(A 4-bit shift register)

There are 16 states (0000, 0001 --- 1111)

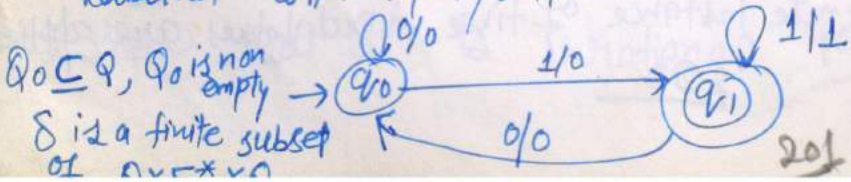
$\Sigma = \{0, 1\}$ input symbol

$O = \{0, 1\}$ output symbol.

Output depends both on input and the state so it is a mealy machine.

Transition System :- [* Every finite automaton can be viewed as a transition system.]

A transition graph or a transition system is a finite directed labelled graph in which each vertex represent a state and the directed edges indicate the transition of a state and the edges are labelled with input/output.



* A transition system need not to be a finite automaton as they may contain more than one initial state.