11/09/23

SET :-

Set :- Collection of object as a single unit.
Each element in a set is a set member / element.

Finite set
- Roster method :- $A = \{1, 2, 3, 4, 5\}$.
- Set builder method :- $\{x \mid x \in N \text{ and } x < b\}$.

Infinite set [$N = \{1, 2, 3, \dots\}$.

(*) In a set, elements are kept in a pair of braces.

Eg:- If $A = \{0, 1\} \Rightarrow |A| = 2$.

(*) empty set :- $\{\}$, ϕ , $\Rightarrow$ cardinality = 0.

(*) Cardinality of a set is the number of set elements in a set.

(*) In a set, order & repitition does not matter.

SEQUENCES & TUPLES :-

(*) A SEQUENCE is an ordered list.
→ objects are kept in a order within a pair of parantheses, also repitition matters.

Eg:- $P = (1, 1, 2)$, $Q = (0, 1)$, $S = (1, 2)$, $R = (1, 0)$ &

Here $\Rightarrow P \neq S$ & $Q \neq R$

$\Rightarrow$ P, Q, R, S are finite sequences.

$\Rightarrow$ if $T = (1, 2, 3, \dots)$, then it is an infinite sequence.

(*) A finite sequence is called a TUPLE.

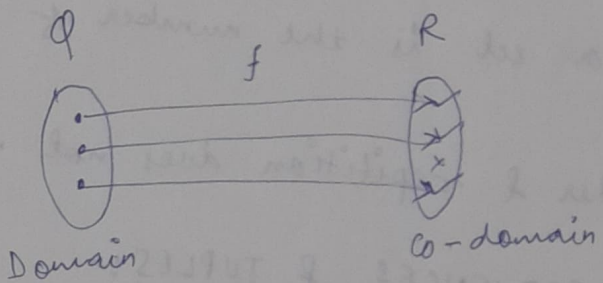
Eg :- $A = (1,1,2)$ ⇒ 3 tuple

$B = (1,0)$ ⇒ 2 tuple.

⇒ A tuple containing K elements, then it is known as a K-tuple.

FUNCTION & RELATION :-

(*) A FUNCTION is an object that set relationship between input and output, i.e., for every input, there is an output.

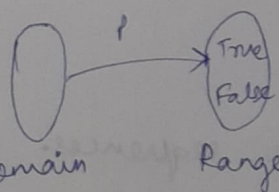


Range = set of all outputs.

domain = set of all inputs.

co domain = range + some other elements.

Eg :-



⇒ Predicate / Property.

(*) A RELATION is a predicate / property whose domain is a set of K-tuples.

* → It is also called as a K-ary relation.

Domain is a cartesian product of K-tup's

* BINARY RELATION :- cartesian product of two 2-tuples.

ALPHABET :- (Σ).

⇒ A non-empty finite set of symbols is known as ALPHABET.

$\Sigma_1 = \{0, 1\}$.

$\Sigma_2 = \{0, 1, 2, 3, \ldots, 9\}$

$\Sigma_3 = \{A, B, \ldots, Z, a, b, \ldots z\}$.

STRING :- (w)

⇒ Over an alphabet, a finite sequence of symbols is known as a STRING.

over Σ_1, $w_1 = 1010|$ $\quad w_2 = 000$

$|w_1| = 5$

$w_3 = 011$

over Σ_3, $w_4 = cse$ $\quad w_5 = cat$ $\quad w_6 = science$.

over Σ_2, $w_7 = 9507$ $\quad w_8 = 24$ $\quad w_9 = 112083$

The no. of elements in a string gives its length.

EMPTY STRING :- (ϵ)

⇒ It is a string that contains no symbols/characters.

⇒ It length is 0.

STRING OPERATIONS :-

⇒ Prefix :- If string x is a prefix of string y then there exists a string z such that $xz = y$.

$y = CSE$

Prefix (y) =				
x	ϵ	C	CS	CSE
z	CSE	SE	E	ϵ

⊛ ϵ is a prefix & suffix for every string.

⇒ Proper prefix :- If string x is a prefix of string y then there exists a string z such that $xz = y$ and $x \neq y$.

$y = CSE$

Proper Prefix (y) =				
x	ϵ	C	CS	
z	CSE	SE	E	

⇒ Suffix :- A string x is a suffix of string y if there exists a z such that $zx = y$.

$y = CSE$

Suffix (y) =				
x	ϵ	E	SE	CSE
z	CSE	CS	C	ϵ

⇒ Proper Suffix :-

$zx = y$, & $x \neq y$.

SUBSTRING :-

⇒ Substring x of a string w, is a sequence of symbols/characters that appears consecutively in w.

Eg:- w = HELLO;

Substring of length 0 = ϵ

length 1 = H, E, L, L, O.

length 2 = HE, EL, LL, LO

length 3 = HEL, ELL, LLO.

length 4 = HELL, ELLO

length 5 = HELLO

14/09/23

REVERSE:-

considering a string w, its reverse is known as w^R, i.e., the reverse of a string is obtained by writing the string in opposite order.

Eg:- A string w: 01011 $\quad \Sigma = \{0, 1\}$

① Reverse w^R: 11010 (01011)

②

$\Sigma = \{a-z\}$

w: cat

w^R: tac

CONCATENATION:-

Concatenation of 2 strings x & y is represented as 'xy' or 'x.y'

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

①

x = ab y = abb

x.y = xy = ababb (concatenating y to x)

= yx = abbab (concatenating x to y)

* Concatenation is not associative.

(2) Concatenating a string x to itself.

$x = ab$

$x^2 = x \cdot x = abab$

$x^3 = x \cdot x \cdot x = ababab.$

$x^K = x \cdot x \cdot \ldots K \text{ times} = ababab \ldots K \text{ times}.$

LANGUAGE :-

A language is a set of strings over an alphabet with some meaning.

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

$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$ → rooster method.

$= \{w \mid w \in \Sigma^*\}$

$= \{w \mid w \text{ is any string over } \{0,1\}\}$ → set builder method.

$L_2 = \{w \mid w \text{ is a string of length } 2\}$

$= \{00, 01, 10, 11\}.$

PROOF BY CONSTRUC^N :-

Many theorem states that (proof by construction) a particular type of object exists and such theorems are proved by demonstrating how to construct the object.

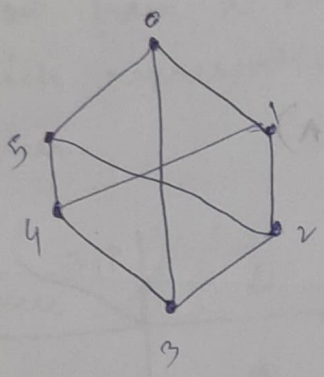
Q/ Prove that for each even number > 2 there exists some graph G which 3-regular.

Proof:- A graph $G(V, E)$ with n nodes when n is even & $n > 2$ can be constructed as:-

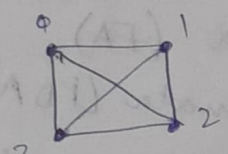
$$V = \{0, 1, 2, \ldots, n-1\}$$

$$E = \{\{i, i+1\} \mid 0 \leq i \leq n-2\} \cup \{0, n-1\} \cup \{\{i, i + \frac{n}{2}\} \mid 0 \leq i < \frac{n}{2}\}$$

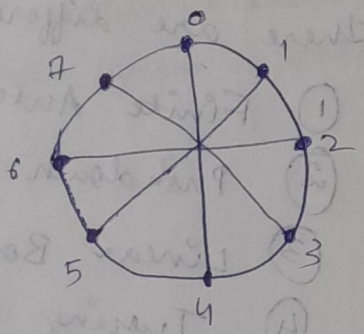
Eg:- $n=6$

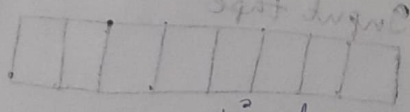


$n=4$



$n=8$





- Picturing the nodes of the graph written consecutively on the circumference of the circle (the edges between the adjacent pairs and opposite side of the circle).
- As the figure shows that every node's degree is 3, the graph is 3-regular.
- Every graph with 2 or more nodes contain 2 nodes that have same degree.

15/09/23

FINITE AUTOMATA (FA) / FINITE STATE MACHINE

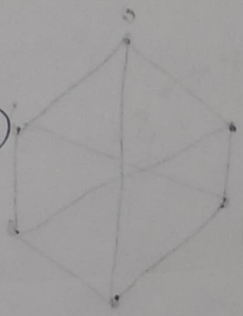
Automation is an algorithm or program which automatically recognizes if a particular string belongs to a language or not by checking the granamar of the string

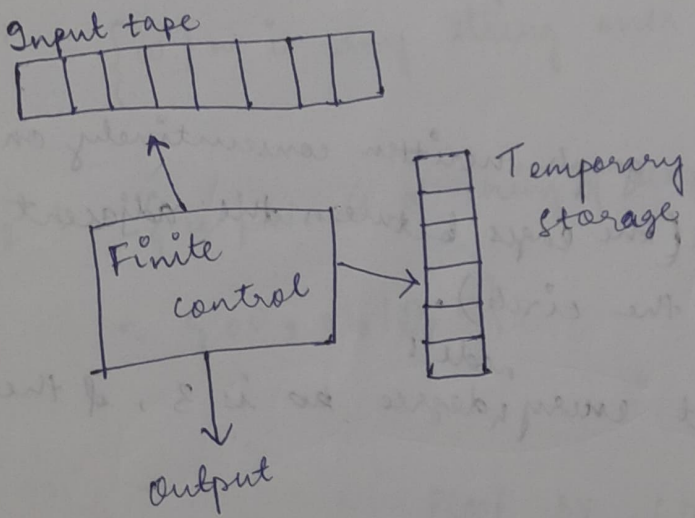
granimer —generates→ Language ←recognises— Automata

Finite Automata :- is an abstract computing machine or device.

There are different variety of such machines like:-

1. Finite Automata (FA)
2. Pushdown Automata (PDA)
3. Linear Bounded Automata (LBA)
4. Turing Machine (TM)
5. Post machine (PM)



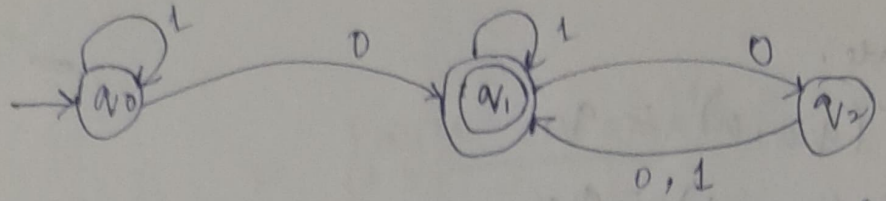


Temporary storage → Keeps the data for a temporary period of time.

Representation :-

Finite automata is represented by 3 ways:-

1) Graphical representation (Transition diagram)
2) Tabular representation (Transition table)
3) Mathematical representation (Transition function)

(1) 

(*) Directed graph associated with vertices called states and edges called transition from one state to another.

→ 3 states represented by circles labelled as q_0, q_1 & q_2.

→ Inputs are 0 & 1 (labelled as edges)

→ The start state / initial state is q_0 represented with an arrow from nowhere.

→ Acceptance state / final state is q_1 represented as double circle.

→ Arrow from a state to another are called transitions which represented the routes to move from 1 state to other.

(2)

I/P / state	0	1
→ q_0	q_1	q_0
q_1*	q_2	q_1
q_2	q_1	q_1

→ Transition table is a table that takes (2 arguments) as input states as row and inputs as columns. The entries are the next state obtained.

→ start state is marked with an arrow and final state marked with a star.

(3) → The transition function is a mapping function denoted as δ.

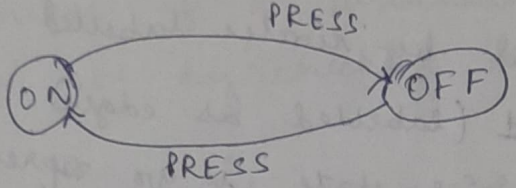
→ The parameter passed to this function are state and input symbol and the function returns

the next state.

$$\delta(q_0, 0) = q_1$$

$$\delta(q_0, 1) = q_0$$

Eg :-



Automaton of an ON, OFF switch.

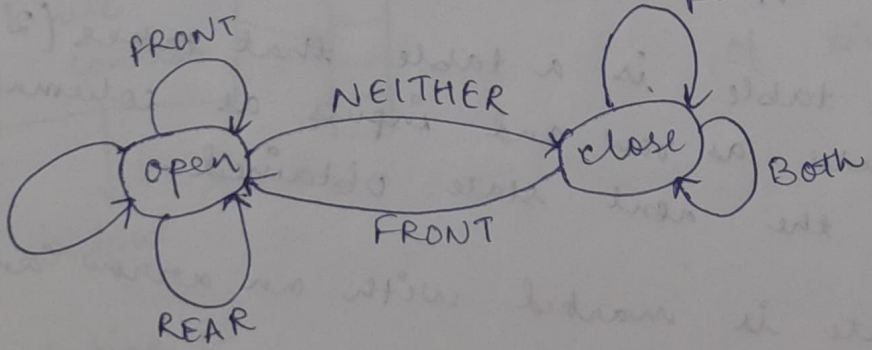
State \ I/P	PRESS
ON	OFF
OFF	ON

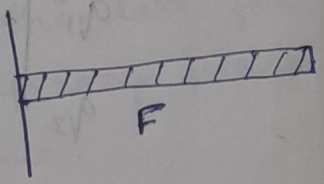
$\delta(ON, PRESS) = OFF$

$\delta(OFF, PRESS) = ON$

Input		PRESS	PRESS	PRESS	PRESS
initially	OFF	ON	OFF	ON	OFF

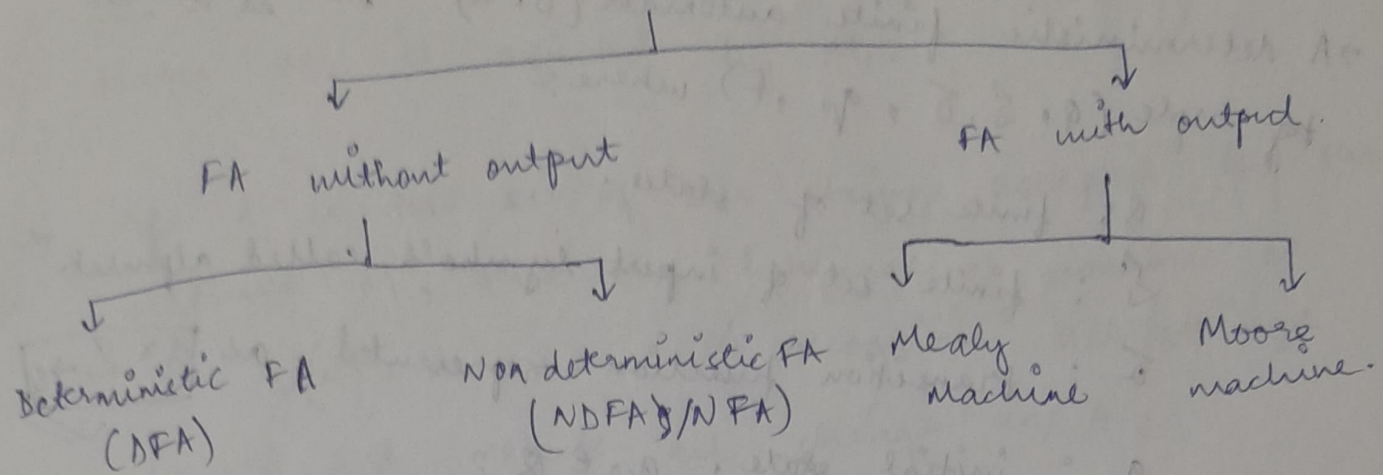
Eg :- Automatic door





18/09/23

FINITE AUTOMATA (FA).



⊛ DETERMINISTIC FINITE AUTOMATA (DFA):-

→ A finite automata is called DFA when it reads an input string one symbol at a time.

→ Deterministic refers to uniqueness of the solution. Therefore in DFA there exists only one path for a specific input from the current state to next state.

→ It does not accept null moves. It can not change its state without any input.

→ DFA contains only one initial state.

→ DFA can contains multiple number of accepting states.

→ The DFA contains 'm' number of states and 'n' number of input symbols then the total number of transitions will be $m \times n$.

⊛ Formal Definition of DFA :-

→ A deterministic finite automata (DFA) 'M' is a 5 tuple $(Q, \Sigma, \delta, q_0, F)$ where :-

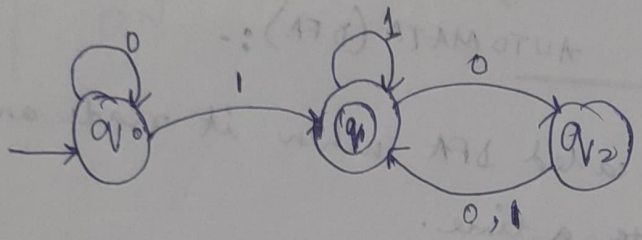
Q : finite set of states.

Σ : finite set of input symbols called alphabet.

δ : transition function represented as $Q \times \Sigma \to Q$

q_0 : initial state, $q_0 \in Q$.

F : ~~state~~ set of accepting states, $F \subseteq Q$



⊛ Acceptance of string :-

→ A string w is accepted by a machine M_1, if it reaches an accepting state starting from the initial state and after processing the string w.

$w_1 = 01101$ $\quad w_1$: $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{1} (q_1)$

→ After processing w_1, starting from q_0, the machine reaches at q_1, which is an accepting state. Hence w_1 is accepted by M_1.

$w_2 = 01010$ $\quad w_2$:

$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_1 \xrightarrow{0} q_2$

After processing w_2 starting from q_0, the machine reaches at q_2 which is a non accepting state. Hence w_2 is rejected by M_1.

→ If A is the set of all strings accepted by the machine M, then the language of machine M is $L(M) = A$, i.e., M recognizes all strings in A and rejects all strings in A.

$$A = \{1, 01, 11, 001, 100, 101, 111, \ldots\}$$

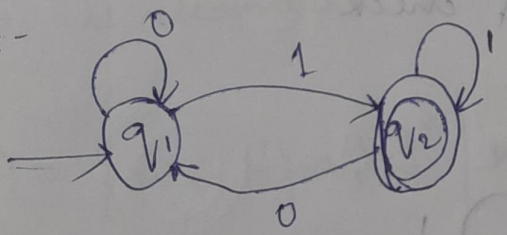
$$= \{w \mid w \text{ contains atleast a } 1 \text{ and an even number of 0's following the last } 1\}.$$

$$L(M) = A$$

21/09/23

Q// Represent this automata with the help of 5 ~~tuple~~ tuple.

Machine M_2 :-



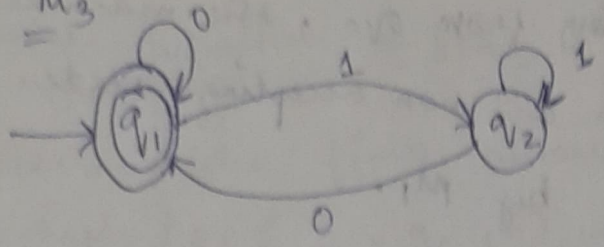
$$M_2 (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\})$$

δ :

Input / state	0	1
→ q_1	q_1	q_2
* q_2	q_1	q_2

$$L(M_2) = \{w \in \{0,1\}^* \mid w \text{ ends in } 1\}$$

Ex: M_3



$M_3(\{q_1, q_2\}, \{0,1\}, \delta, q_1, \{q_1\})$

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

$\delta(q_2,1) = q_2$

$A = \{\epsilon, 0, 00, 0\cdots0, 10, 1000\cdots0, 1\cdots10_0\cdots0, \cdots\}$

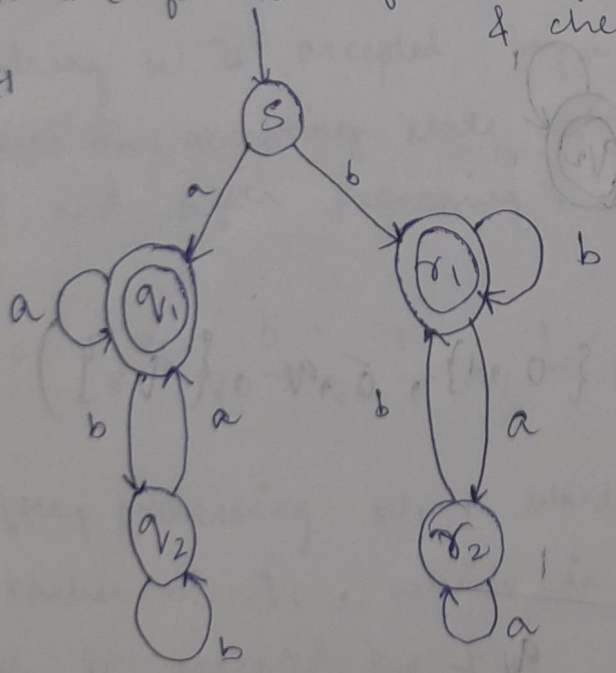
$L(M_3) = \{w \in \{0,1\}^* \mid w$ is an empty/null string or w ends in $0\}$.

(*) In a DFA, the initial state is Final state, then the DFA accepts ϵ.

(*) Else, ϵ doesn't have any transitions.

Ex: write the formal definition of this DFA. find the language & check

M_4



$M_4(\{S, q_1, q_2, r_1, r_2\}, \{a,b\}, \delta, S, \{q_1, r_1\})$

Input / State	a	b
→ s	q_1	r_1
q_1^*	q_1	q_2
q_2	q_1	q_2
r_1^*	r_2	r_1
r_2	r_2	r_1

$q_0 = s$

$F = \{q_1, r_1\}$.

$w_1 = a\ b\ b\ b\ a\ a$ ✓ accepted.

$s \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{b} q_2 \xrightarrow{b} q_2 \xrightarrow{a} q_1 \xrightarrow{a} q_1$

$w_2 = a\ b\ b\ a\ a\ b$. ✗ not accepted.

$s \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{b} q_2 \xrightarrow{a} q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_2$.

So, w_1 is recognized & w_2 is not recognized by M_4.

$L(M_4) = \{w \in \{a,b\}^* \mid w$ starts and ends in same symbol$\}$.

⑧ FORMAL DEFINITION OF COMPUTATION :-

→ Let $M = (Q, \Sigma, \delta, q_0, F)$ & $w = a_1 a_2 \cdots a_n$ be a string, where each a_i is a member of the alphabet set Σ $(a_i \in \Sigma)$. The M accepts w if a sequence of ~~steps~~ states $r_0, r_1, \cdots, r_n$ in $Q \in Q$ following 3 conditions :-

(i) $r_0 = q_0$ [initial state where processing starts]

(ii) $\delta(r_i, a_{i+1}) = r_{i+1}$ (machine goes from one state to another)

(iii) $r_n \in F$ (the machine accepts the input if the last state is an accepting state).

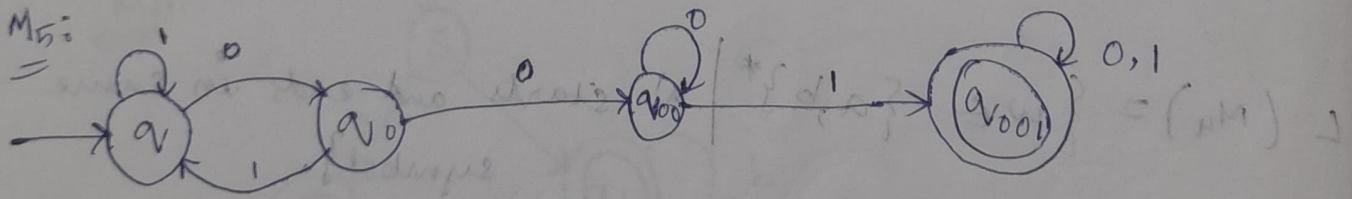
(*) REGULAR LANGUAGE :-

→ A language is called a regular language if some finite automaton recognizes it.

DESIGNING FINITE AUTOMATA

Q) write the formal description of the finite automaton & find the language accepted by it.

M_5 :



$$M(\{q, q_0, q_{00}, q_{001}\}, \{0,1\}, \delta, q, \{q_{001}\})$$

state \ input	0	1
→ q	q_0	q
q_0	q_{00}	q
q_{00}	q_{00}	q_{001}
q_{001}^*	q_{001}	q_{001}

$L(M_5) = \{$ all strings that contain 001 as substring over $\Sigma = \{0,1\}\}$.

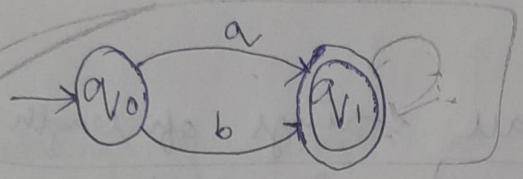
22/09/23

(*) while we are designing a finite automata:-
→ states are represented by circles.
→ transitions are represented by lines.

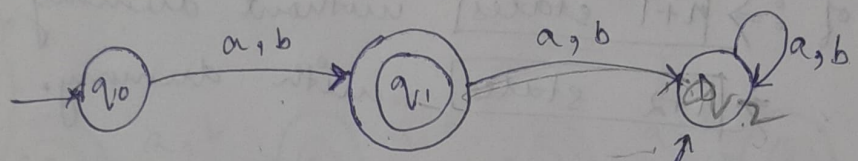
Q.1) Construct DFAs over the alphabet set, $\Sigma = \{a, b\}$, that recognises:- (i) All strings of length exactly 1; $|w| = 1$.

~~(iii)~~ $A = \{a, b\}$.

ans:-



(*) Parallel lines indicate 'OR' operation

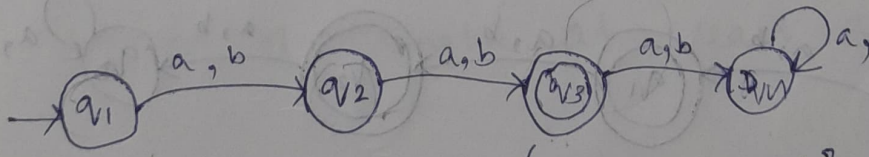


(*) Trap state/Dead state/Dummy state/ϕ state.
(can't escape this state).

(ii) All strings of length exactly 2, $|w| = 2$

$A = \{aa, ab, ba, bb\}$.

M_2:-

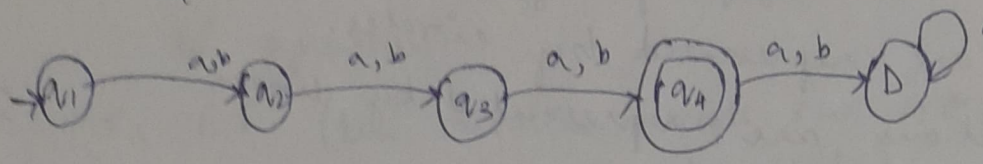


$M_2(\{q_1, q_2, q_3, D\}, \{a, b\}, \delta, q_1, \{q_3\})$

state \ input	a	b
→ q_1	q_2	q_2
q_2	q_3	q_3
q_3^*	D	D
D	D	D

(iii) All strings of length exactly 3, $|w|=3$.

$A = \{aaa, aab, aac, \ldots, ccc\}$.



$M(\{q_1, q_2, q_3, q_4, D\}, \{a,b\}, \delta, q_1, \{q_4\})$.

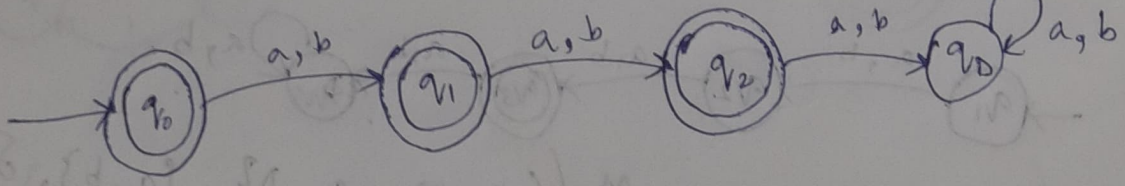
NOTE

for every DFA accepting all strings of length $= n$,
it consists of :⇒ $n+1$ states without dummy. (NFA)
⇒ $n+2$ states with dummy. (DFA)

(iv) All strings of length at most 2, $|w| \leq 2$.

$A = \{\epsilon, a, b, aa, ab, ba, bb\}$.

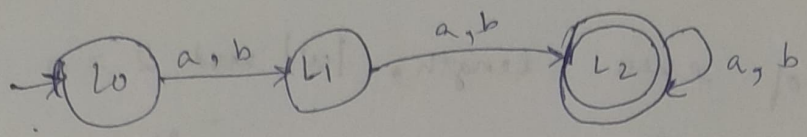
0, 1, 2



$M(\{q_0, q_1, q_2, q_D\}, \{a,b\}, \delta, q_0, \{q_0, q_1, q_2\})$

(v) All strings of length at least 2, $|w| \geq 2$.

$A = \{aa, ab, ba, bb, aaa, aab, aac, \ldots\}$.



$$M(\{L_0, L_1, L_2\}, \{a, b\}, \delta, L_0, \{L_2\})$$

(*) when designing a DFA :-

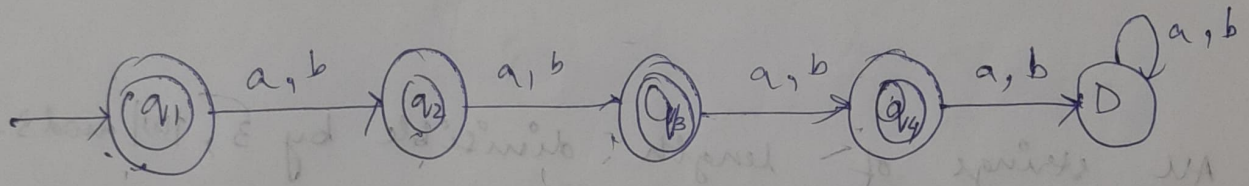
* → when constructing it for length at most 2,

$$\boxed{\text{states} = n+2}.$$

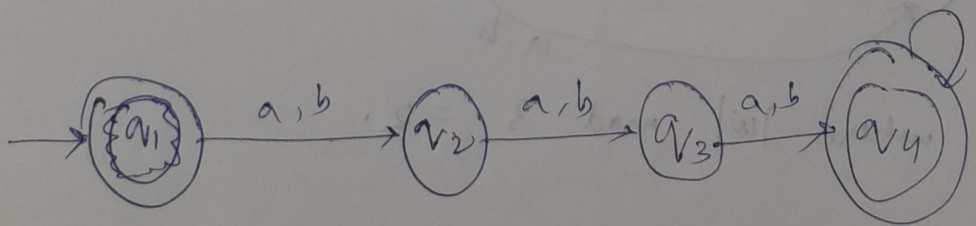
→ when constructing it for length at least 3,

$$\boxed{\text{states} = n+1}.$$

(vi) All strings of length at most 3, $|w| \leq 3$.

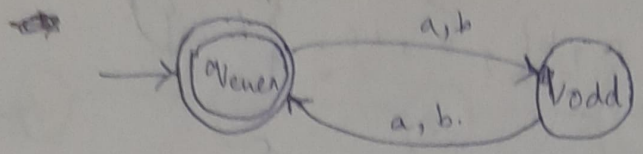


(vii) All strings of length at least 3, $|w| \geq 3$.

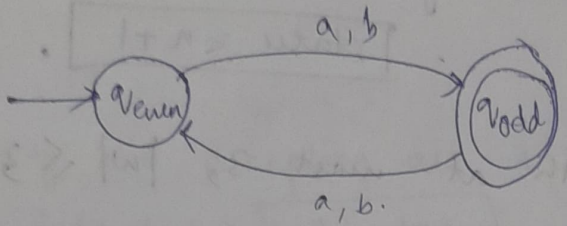


(viii) All strings of even length, $|w| \bmod 2 = 0$.

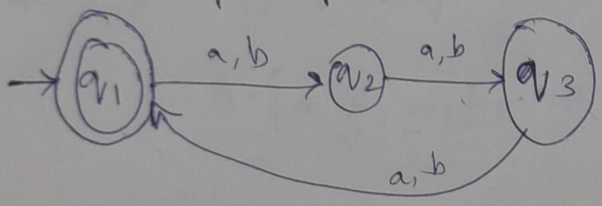
num % 2
= 0 → even
= 1 → odd.



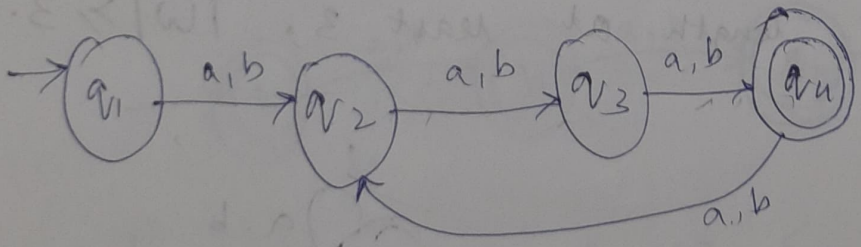
(ix) All strings of odd length, $|w| \bmod 2 = 1$.



(x) All strings of length divisible by 3, $|w| \bmod 3 = 0$.



OR



(xi) All strings where $|w| \bmod 3 = 2$.

Q1 DFA for :-

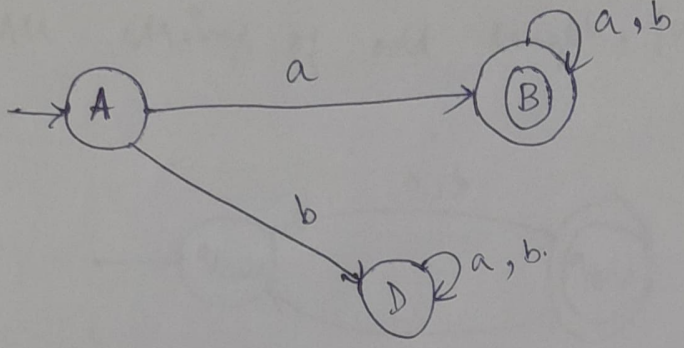
(a) All string starting in a

(b) All string starting in ab

(c) All string starts with abb.

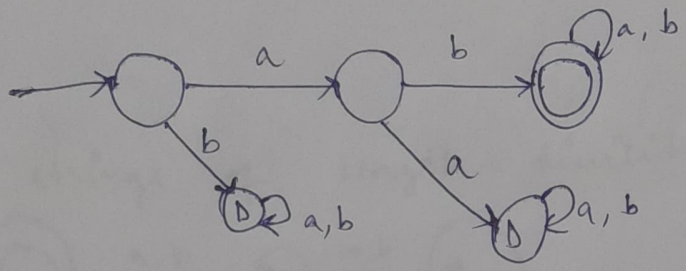
ans :-

(a)

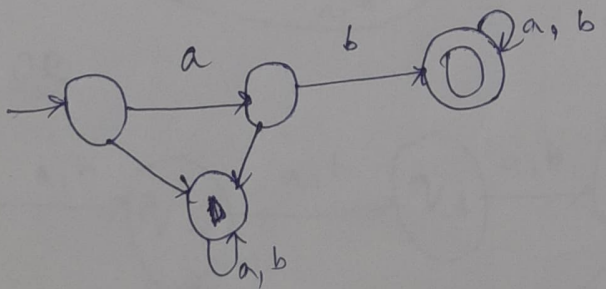


{a, aa, ababa, abba, aaabb, ..}

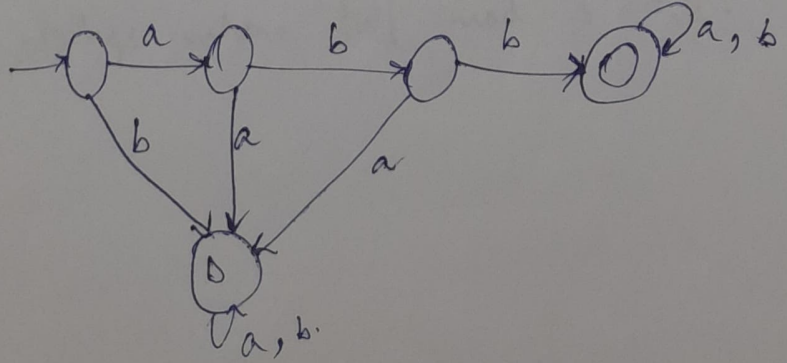
(b)



OR



(c)



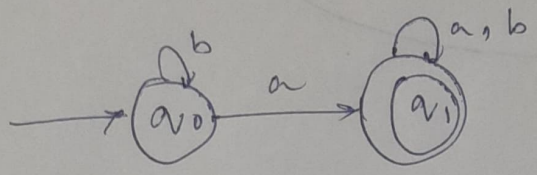
Q) (a) All strings containing 'a' as substring

(b) All strings containing 'ab' as substring.

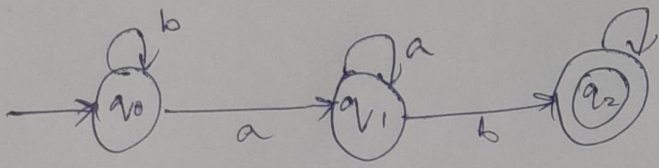
(c) All strings containing abb as substring.

DFA for the above:-

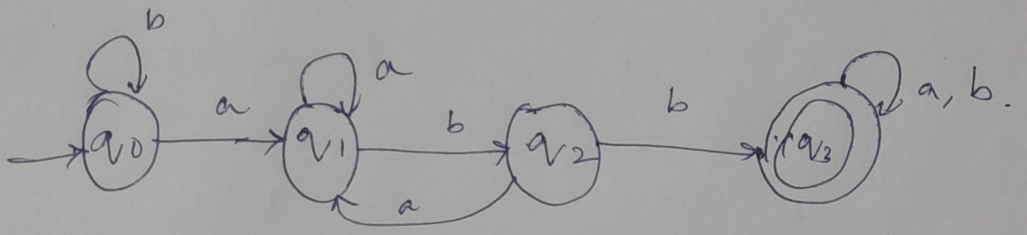
(a)



(b)

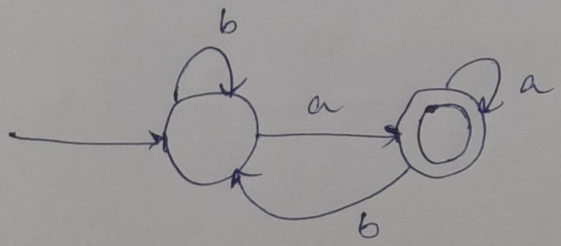


(c)

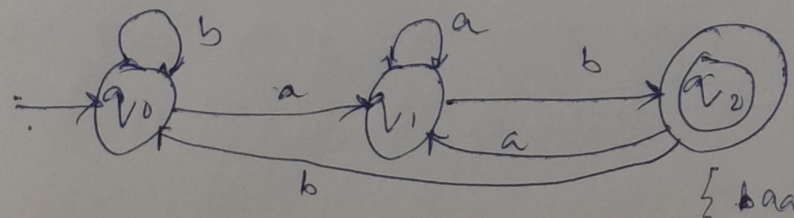


Q) DFA for :-

(a) all strings that end in 'a' :-

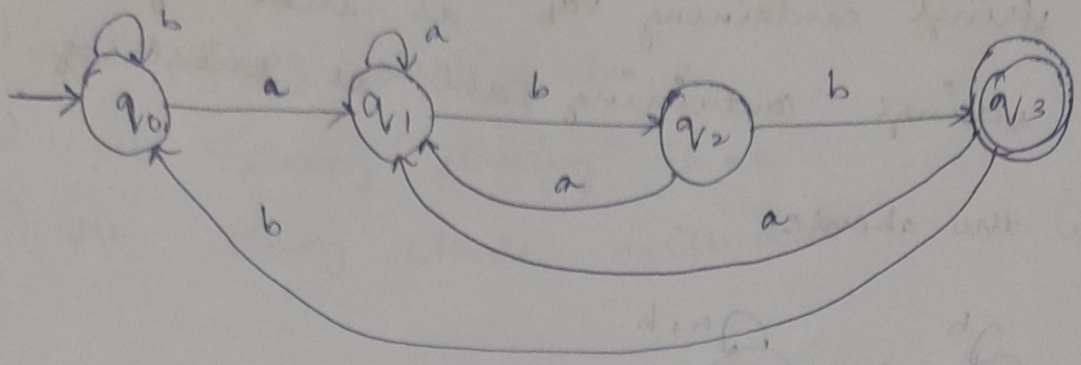


(b) end in 'ab' :-



{baabab, aabbabab, ...}

(c) end on 'abb'

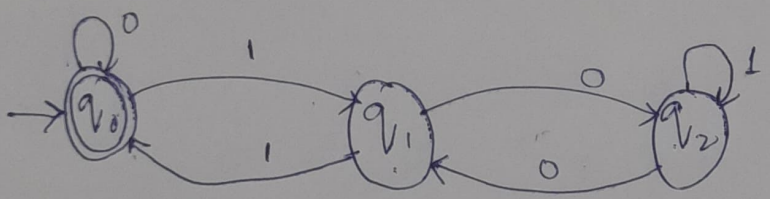


22/09/23

Q// Construct a DFA over $\Sigma = \{0, 1\}$ such that it accepts all strings represented in binary divisible by 3.

Ans :-

Binary	decimal	n = 3	state / rem
0	0	mod 3	0
01	1	mod 3	1
10	2	"	2
11	3	"	0
100	4	"	1
101	5	"	2
110	6	"	0
111	7	"	1



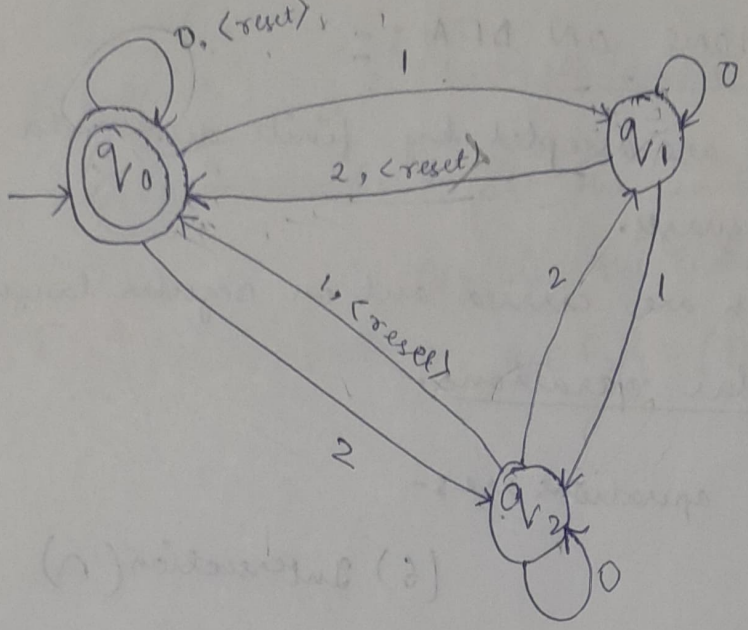
Transition table :-

states	0	1
→ q_0*	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

q_0, q_1, q_2, q_0, q_1, q_2, ... pattern follows in the state table.

Q// Design a Finite state machine such that it accepts all strings over $\Sigma = \{0, 1, 2, \langle RESET \rangle\}$ where the sum of the digits in the string are

when divisible by 3, are reset and accepted.



0, 1, 2,

01 012

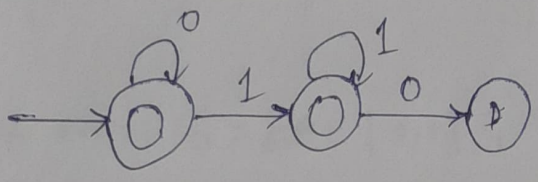
12 12

122

Q Construct a DFAs for the languages (a) $L_1 = \{ 0^m 1^n \mid m, n \geq 0 \}$

(b) $L_2 = \{ 0^m 1^n \mid m, n \geq 1 \}$.

(a)

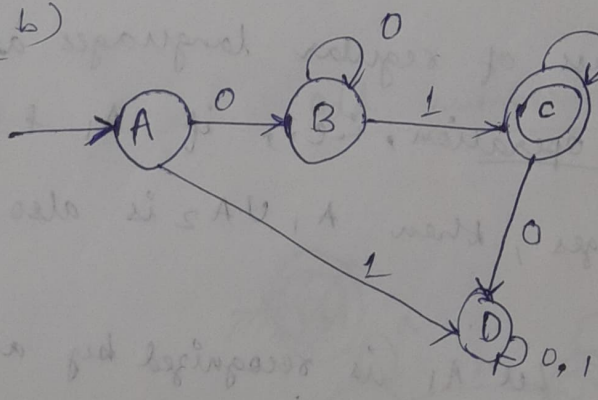


10101

$2^4 + 2^2 + 2^0$

4 + 4 + 1

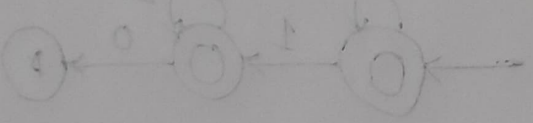
(b)



21

REGULAR OPERATIONS ON DFA :-

- The Languages that are accepted by finite automata are called regular languages.
- The operations which are carried out on regular languages are called regular operations.
- The basic regular operations are :-
  (1) union (U)
  (2) concatenation (·)
  (3) Kleen star/Kleen closure (*)
  (4) Complement (-)
  (5) Reversal (R)
  (6) Intersection (∩)
  (7) set difference (-)
  (8) Homomorphism
  (9) Inverse homomorphism.



(1) UNION (U) :-

The class of regular languages are closed under union operation, i.e., if A_1 & A_2 are two regular languages, then $A_1 \cup A_2$ is also a regular language.

Proof :- Let A_1 is recognized by a machine $M_1(Q_1, \Sigma_1, \delta_1, q_1, F_1)$ and A_2 is recognized by a machine $M_2(Q_2, \Sigma_2, \delta_2, q_2, F_2)$.

We have to construct a machine $M(Q, \Sigma, \delta, q, F)$ that recognizes the language $(A_1 \cup A_2)$.

1) $Q = Q_1 \times Q_2 = \{ (p,q) \mid p \in Q_1 \text{ \& } q \in Q_2 \}$

2) $\Sigma = \Sigma_1 \cup \Sigma_2$.

3) $\delta((p,q),a) = (\delta(p,a), \delta(q,a))$ where $a \in \Sigma$, $p \in Q_1$, and $q \in Q_2$.

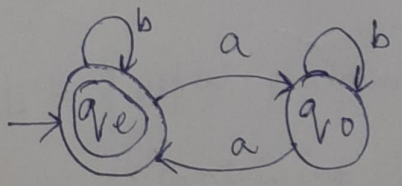
4) $q_0 (q_1, q_2)$ where q_1 is the initial state of M_1 and q_2 is the initial state of M_2.

5) i) $F = \{(p,q) \mid p \in F_1 \text{ or } q \in F_2\} = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

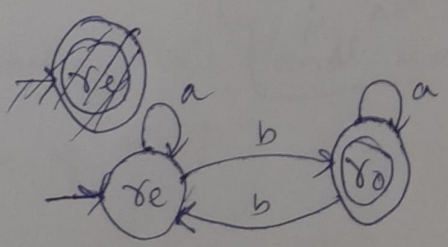
Q1. Design a DFA over $\Sigma = \{a, b\}$, such that it accepts all strings that contains even number of 'a's or odd number of 'b's.

Q2. Design a DFA over $\Sigma = \{0, 1\}$, such that it accepts all strings that contains even number of '0's or ends in a '1'.

Ans 1: even number of 'a's



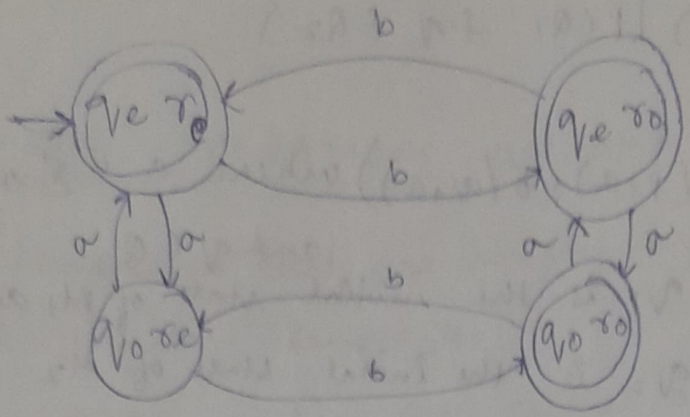
odd number of 'b's



$M_1(\{q_e, q_o\}, \{a, b\}, \delta, q_e, \{q_e\})$

$M_2(\{r_e, r_o\}, \{a, b\}, \delta, r_e, \{r_o\})$.

$M(\{q_e r_e, q_e r_o, q_o r_e, q_o r_o\}, \{a, b\}, \delta, q_e r_e, \{q_e r_e, q_e r_o, q_o r_o\})$.


b
b
a
a
b
a
a
b

29/09/23

(2) INTERSECTION (∩) :-

M recognizes $A_1 \cap A_2$

$F = \{(p,q) \mid p \in F_1 \text{ and } q \in F_2\}$.

Eg (i) According to the example of even 'a's and odd 'b's, both the cond's must satisfy in the case. The initial state will be only one and the final state will also be one.

$A_1 \cap A_2 = \{b, aab, baa, aba, \cdots\}$.

Formal description - $M(\{q_{ere}, q_{ero}, q_{ore}, q_{oro}\}, \{a,b\}, \delta, q_{ere}, \{q_{ero}\})$.

Language (L):- $\{w \mid w$ contains even number of 'a's and odd number of 'b's.

(3) SET DIFFERENCE (-):-

M recognizes $A_1 - A_2$.

$F = \{(p,q) \mid p \in F_1 \text{ and } q \notin F_2\}$

(*) ⇒ A_1's final state is the final state for $A_1 - A_2$.

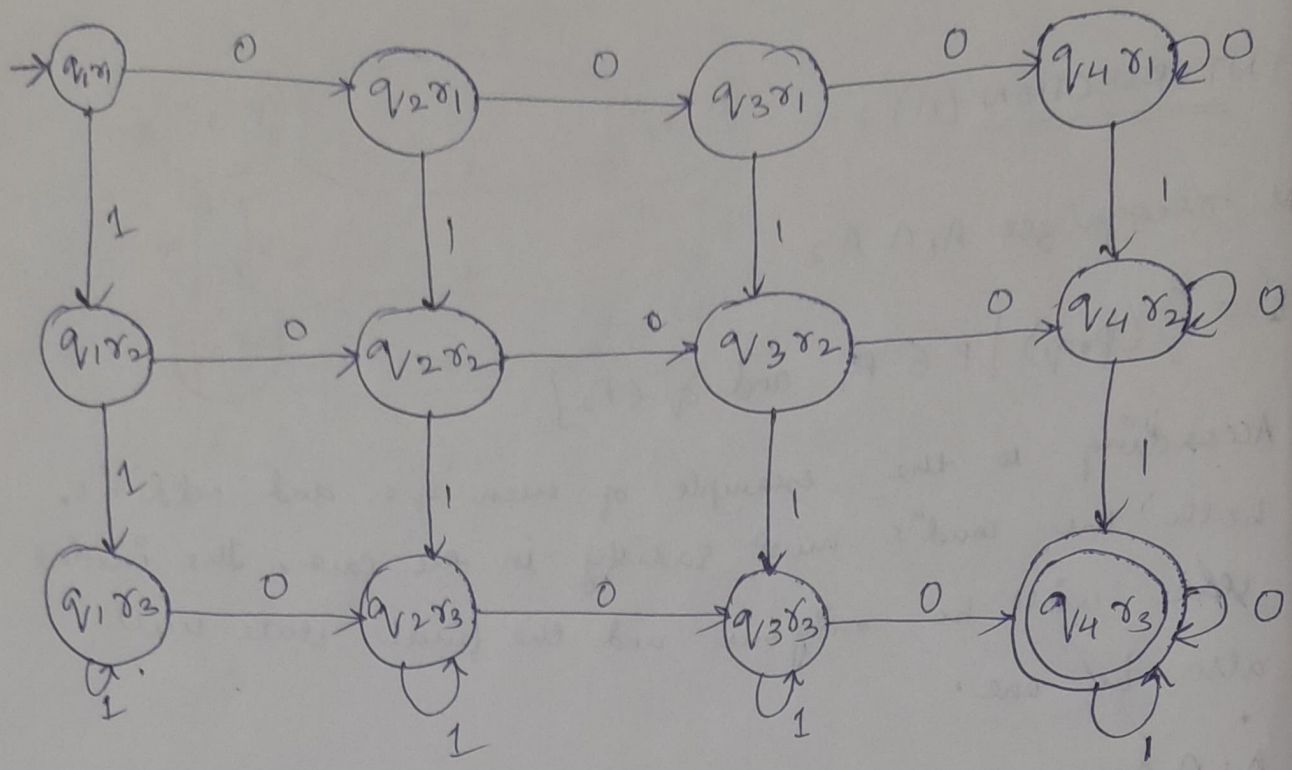
A_2's final state is not final and not (~)

$L = \{w \mid w$ contains even number of 'a's containing odd number of 'b's.$\}$

$= \{\epsilon, bb, aa, baba, aabb, abab, \cdots\}$.

Q) Design a DFA $\Sigma = \{0,1\}$ such that all strings containing atleast 3 zeroes and two '1's.

01010 11000 000 11

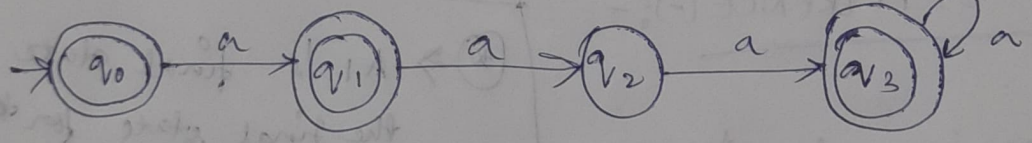


Q/ Construct DFA

$$L = \{a^n \mid n \geq 0 \text{ and } n \neq 2\}.$$

↳ $\{a, \epsilon, aaa, aaaa, \ldots\}$

~~aa~~



(4) COMPLEMENT (¯):-

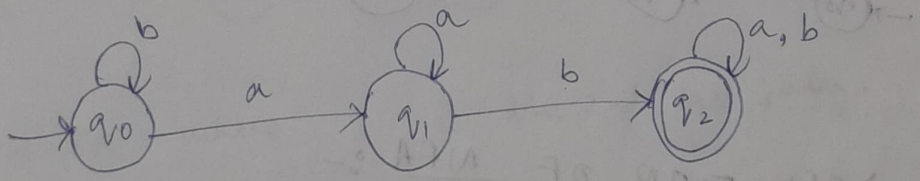
Theorem :- The class of regular language is closed under complement
i.e., If A is a regular language, then $\bar{A}$ (complement) is also regular.

Proof :- Considering a regular language A.
It's complement $\bar{A} = \Sigma^* - A$.

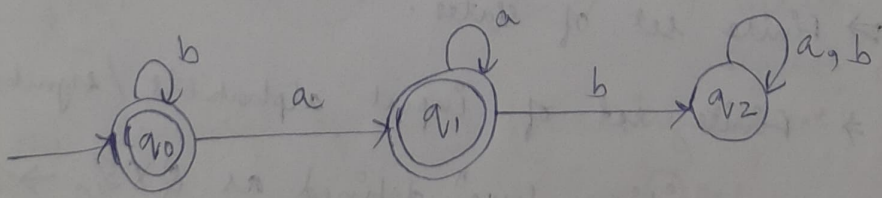
Let A is recognized by a machine $M_1(Q_0, \Sigma_0, \delta_0, q_0, f_0)$.
then M can be constructed as a machine that recognizes $\bar{A}$ such as $M(Q, \Sigma, \delta, q_0, \{Q - F\})$

Eg :- Construct a DFA that accepts all string not containing 'ab' as substring.

ans :- if 'ab' as substring :-



Now 'ab' not a substring :-

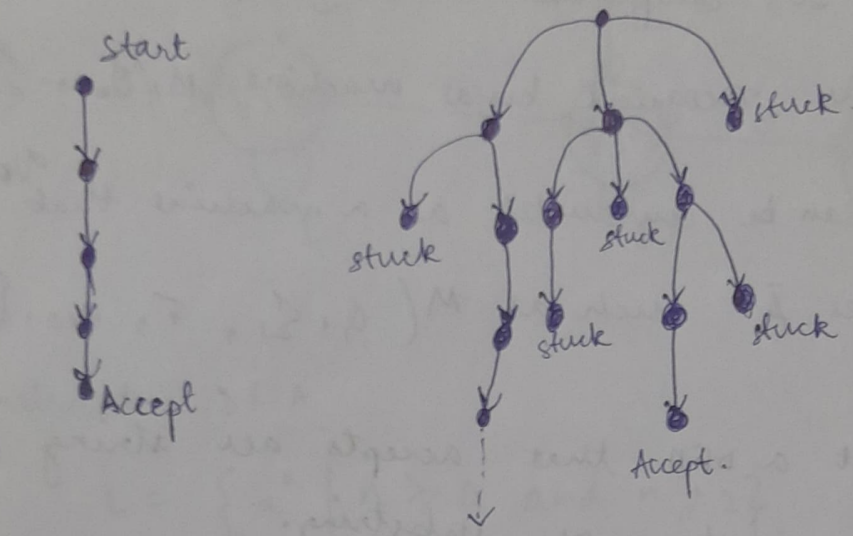


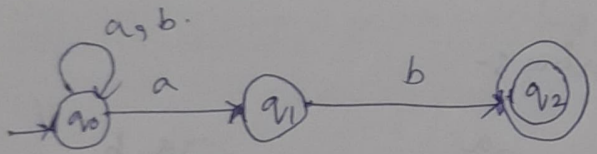
05/10/23

NON DETERMINISTIC FINITE AUTOMATA(NFA /NDFA)

The Finite automata are called NFA /NDFA when there exists multiple paths for a specific input from the current state to the next state.

- Hence in other words, in NFA with one step with input, we can transit into a set of possible next states.
- In NFA we can transit from one state to another without reading an input, i.e., through 'ϵ transitions'.





FORMAL DEFINITION OF NFA :-

A NFA is a 5 tuple $(Q, \Sigma, \delta, q_0, F)$, where:-

Q is ⇒ finite set of states.

Σ is ⇒ finite set of input alphabets / symbols.

δ is ⇒ transition func^n defined as $Q \times \Sigma_\epsilon \to P(Q)$

or $Q \times \Sigma_\epsilon \longrightarrow 2^Q$, where $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$

& $P(Q)$ is the power set of Q.

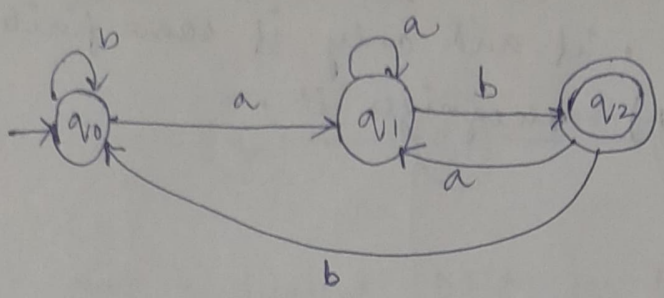
$q_0 \in Q$ is ⇒ the initial set.

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

Q// $L = \{ w \mid w \text{ ends in ab} \}$

$Q \times \Sigma \rightarrow Q$

DFA:-

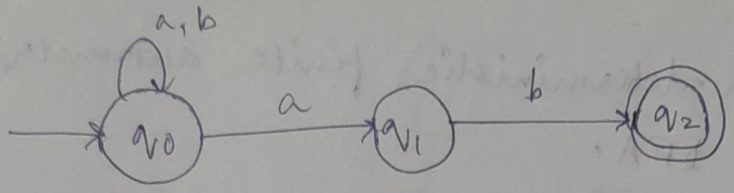


(q_0, a)
(q_0, b) (q_1, a)
(q_1, b) (q_2, a)
(q_2, b)

δ

q_0
q_1
q_2

NFA:-



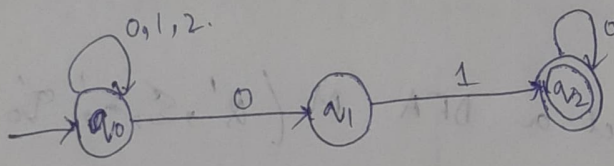
$Q \times \Sigma_\epsilon \rightarrow P(Q)$

(q_0, a)
(q_0, b) (q_0, ϵ)
(q_1, a) (q_1, b)
(q_1, ϵ) (q_2, a)
(q_2, b) (q_2, ϵ)

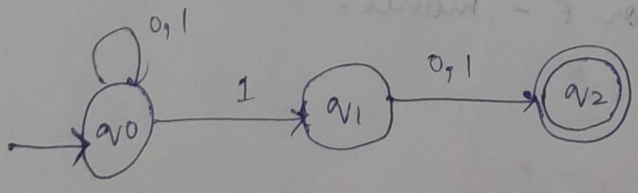
δ

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

Q// Construct a NFA that contains "01" as substring over the $\Sigma = \{0, 1, 2\}$

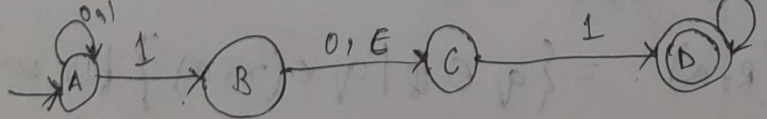


Q// Construct a NFA where 1 is the 2nd last string.



$L = \{ w \in \{0,1\}^* \mid w$ in which 2nd last symbol is 1$\}$.

Q// Find the language of the following element:



$L = \{ w = \{0,1\}^* \mid w$ containing 11 or 101 as substring$\}$.

EQUIVALENCE OF NFAs & DFAs :-

- DFAs and NFAs recognise the same class of Languages
- A language is Regular, if and only if some finite automata (DFA or NFA) recognizes it.

06/10/23

Theorem :- Every Non-deterministic finite automaton has an equivalent DFA.

Idea :- Two FAs M and N are equivalent if they both recognize the same language.

$$L(M) \Leftrightarrow L(N)$$

$$L(DFA) \Leftrightarrow L(NFA)$$

Proof :- Let $N(Q, \Sigma, \delta, q_0, F)$ be a NFA recognizing same language A.

- We can construct a DFA $M(Q', \Sigma, \delta', q_0', F')$ recognizing A.

Here we have considered that the NFA contains no ϵ transitions or ϵ - moves.

1. $Q' = P(Q)$
2. Σ
3. For $R \in Q'$ and $a \in \Sigma$

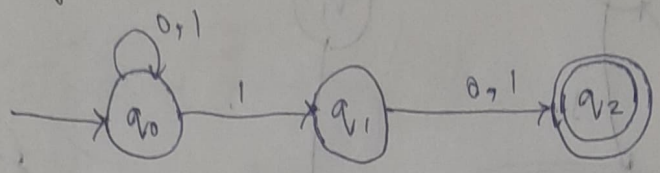
$$\delta'(R, a) = \{q \in Q \mid q \in (r, a)\} \text{ for some } r \in R\}.$$

$$= \bigcup_{r \in R} \delta(r, a)$$

4. $q_0' = q_0$
5. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

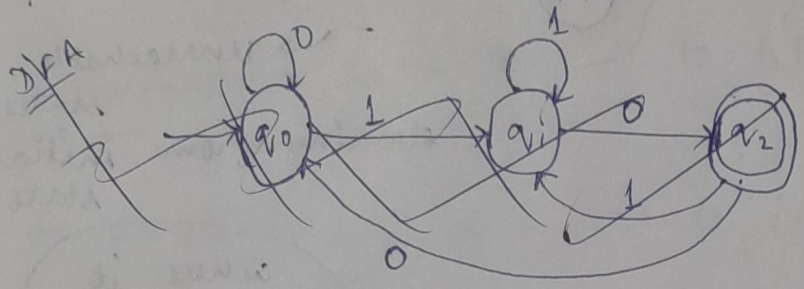
Q// Construct a NFA over $\Sigma = \{0, 1\}$ such that it accepts all strings when the 2nd last digit is one.

⇒ NFA



⇒ Find the equivalent DFA for the NFA design.

DFA



For NFA,

$N(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$

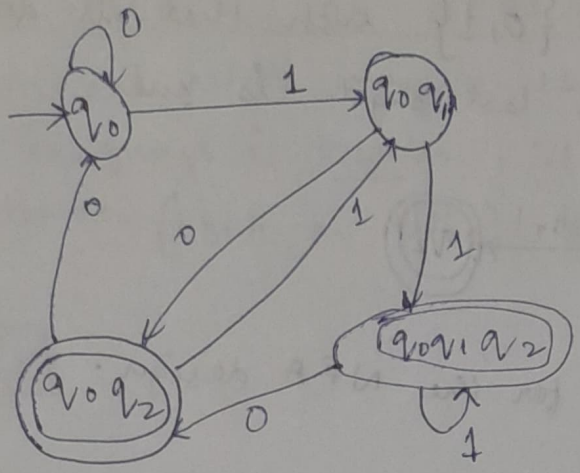
δ :	0	1
$\rightarrow\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$\{q_2\}$	ϕ	ϕ

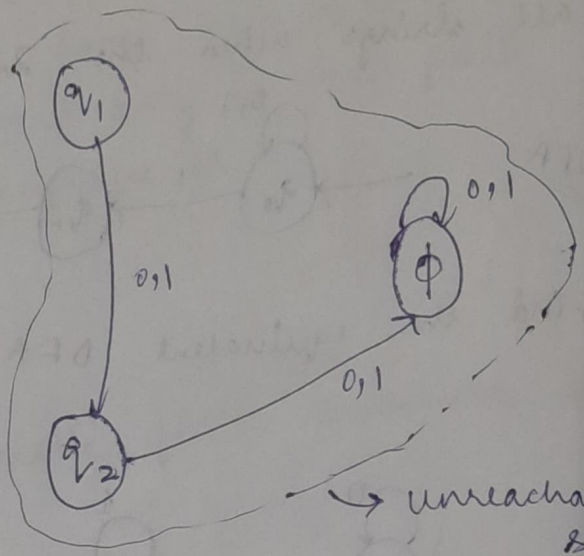
For DFA :-

	0	1
q_0	q_0	$q_0 q_1$
q_1	q_2	q_2
q_2	ϕ	ϕ
$q_0 q_1$	$q_0 q_2$	$q_0 q_1 q_2$
$q_0 q_2$	q_0	$q_0 q_1$
$q_0 q_1 q_2$	$q_0 q_2$	$q_0 q_1 q_2$
ϕ	ϕ	ϕ

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

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





→ unreachable state starting from initial state.

	0	1
→ q_0	q_0	q_0q_1
q_0q_1	q_0q_2	$q_0q_1q_2$
q_0q_2*	q_0	q_0q_1
$q_0q_1q_2$*	q_0q_2	$q_0q_1q_2$

where it contains no ε.

steps (optimized) :-

1) Write the initial state in the state column, and its transition from NFA transition table.

2) When a new state is found in the entris of NFA transition table then add it to a state columen.

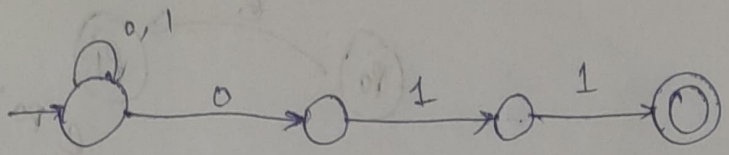
(*) Repeat the process untill not getting a new state in the table!

Q) Design NFA and find equivalent DFAs for following languages :- (a) ending with 011

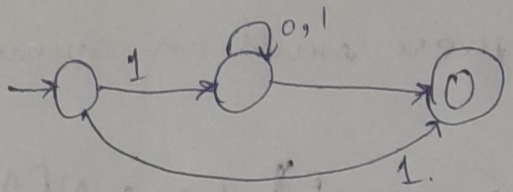
(b) starting with 1 and ending with 1.

NFA :-

(a)



(b)



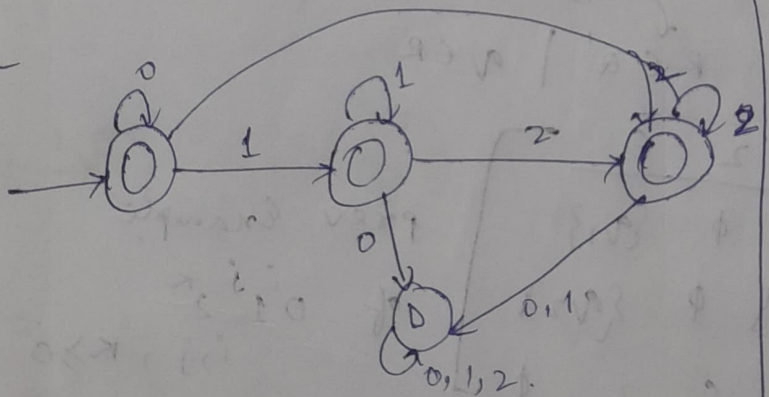
CASE-2 :- WHERE IT CONTAINS ε TRANSITIONS

Equivalence to NFA - ε to DFA

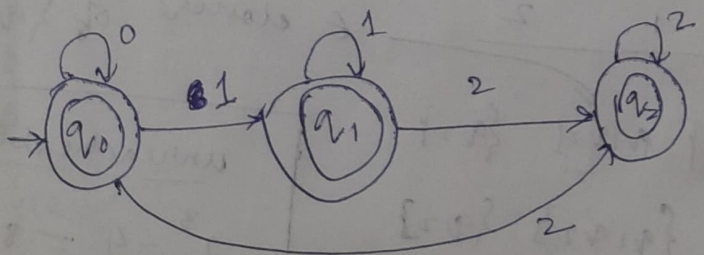
ε-closure :- of a state is the state itself alongwith the others states that are reachable from the state or a series of ε transitions.

Eg :- Construct a DFA, NFA without ε-transitions and a NFA with ε-transitions for the language $L = \{ 0^i 1^j 2^k \mid i, j, k \geq 0 \}$.

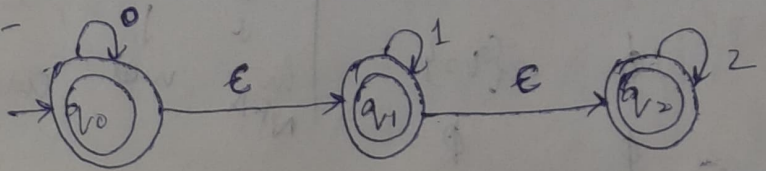
DFA :-



NFA :-



NFA - ε :-



ε-closure calc^n :-

ε-closure (q_0)

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

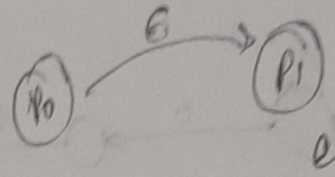
ε-closure (q_1)

$= \{ q_1, \overset{\epsilon}{q_2} \}$

ε-closure (q_2)

$= \{ q_2 \}$

(*) Theorem



* For every NFA-ϵ, there exists an equivalent DFA.

Proof :- Let $N(Q, \Sigma, \delta, q_0, F)$ be a NFA-ϵ recognizing some language A.

We can construct a DFA $M(Q', \Sigma', \delta', q_0', F')$ that recognizes A such that :-

DFA ↴

1. $Q' = P(Q)$ ↳ NFA.
2. $\Sigma' = \Sigma$
3. for $R \in Q'$ and $a \in \Sigma'$

$$\delta'(R, a) = \epsilon\text{-closure}\left(\bigcup_{r \in R} \delta(r, a)\right)$$

4. $q_0' = \epsilon$-closure (q_0).
5. $F' = \{q \in R \mid q$ is same accepting state of NFA$\}$.

$R \in Q' \mid q \in R$

NFA

δ :	0	1	2	ϵ
q_0	q_0	ϕ	ϕ	$\{q_1\}$
q_1	ϕ	$\{q_1\}$	ϕ	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$	ϕ

prev Example of $0^i 1^j 2^k$ $i, j, k \geq 0$

ϵ closure of $\{q_1\}$

DFA

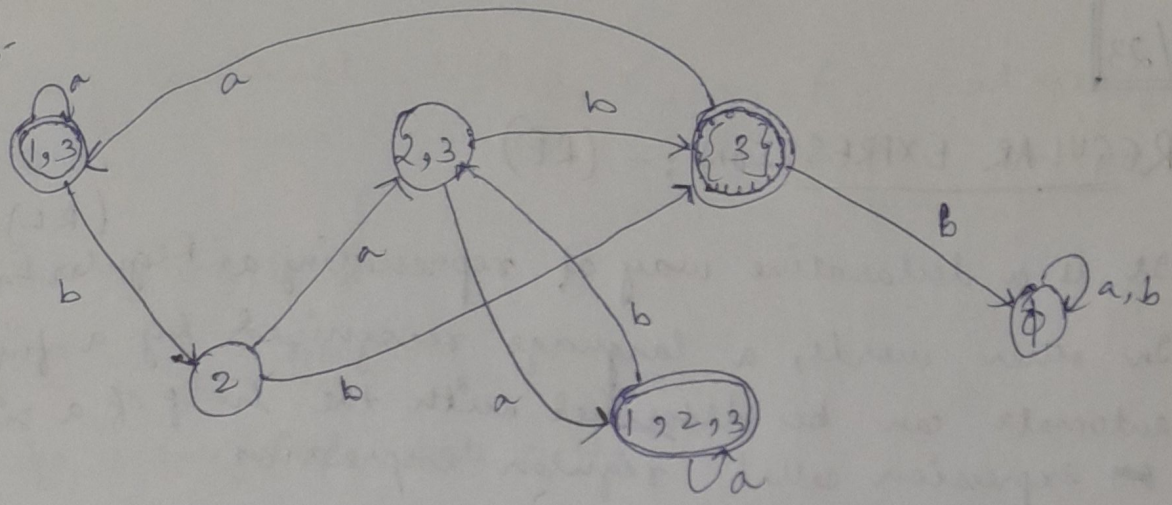
	0	1	2
→ $\{q_0 q_1 q_2\}$	$\{q_0 q_1 q_2\}$	$\{q_1 q_2\}$	$\{q_2\}$
$\{q_1 q_2\}$	ϕ	$\{q_1 q_2\}$	$\{q_2\}$
$\{q_2\}$	ϕ	ϕ	$\{q_2\}$
ϕ	ϕ	ϕ	ϕ

unreachable states

$2^3 - 4 = 8 - 4 = 4$

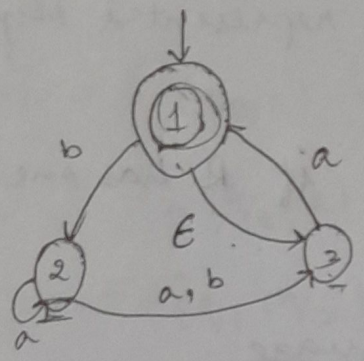
↓ in NFA ↓ used ↓ unreachable states.

DFA :-



Conv. DFA

NFA :-



δ : NFA :-

	a	b	ϵ
→ {1}*	ϕ	{2}	{3}
{2}	{2,3}	{3}	ϕ
{3}	{1}	ϕ	ϕ

ϵ - closure 1 : {1,3}

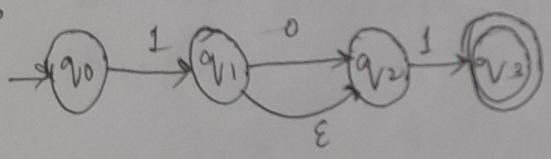
ϵ - closure 2 : {2}

ϵ - closure 3 : {3}

δ : DFA :-

	a	b	c
→ {1,3}*	{1,3}	{2}	
{2}	{2,3}	{3}	
{2,3}*	{1,2,3}	{3}	
{1,2,3}*	{1,2,3}	{2,3}	
{3}	{1,3}		
ϕ	ϕ	ϕ	

Q.26



ϵ-NFA

	0	1	ϵ
→ q0	ϕ	q1	ϕ
q1	q2	ϕ	q2
q2	ϕ	q3	ϕ
q3*	ϕ	ϕ	ϕ

ϵ - closure :-

q0 → {q0}

q1 → {q1, q2}

q2 → {q2}

q3 → {q3}

δ - DFA :-

	0	1
→ q0	ϕ	{q1, q2}
{q1 q2}	{q2}	{q3}
{q2}	ϕ	{q3}
{q3}	ϕ	ϕ
ϕ	ϕ	ϕ

12/10/23

REGULAR EXPRESSION :- (RE) (RL)

→ It is a declarative way of representing a Regular Language.
→ In other words, a language recognized by a finite automata can be described with the help of a simple expression called regular expression.
→ The language that can be represented by a RE are reffered as RL.
→ R is said to be a RE if R has one of the following forms:-

RE	language
ϕ	$\{\}$ // does not accept any string
ϵ	$\{\epsilon\}$
a	$\{a\}$

RE	language
$R_1 \cup R_2$ (union)	$L(R_1) \cup L(R_2)$
$R_1 \cdot R_2$ (concatenation)	$L(R_1) \cdot L(R_2)$
(star) R_1^*	$(L(R_1))^*$
(R_1)	$L(R_1)$

REGULAR OPERATIONS :-

→ Basically, three types of operations are carried out:-
- Union (+, ∪, |)
- Concatenation (·)
- star/Kleen closure (*)

→ Precedence :- Parenthesis > star > Concatenation > Union
() > * > · > +

Theorem-1

Rule 1:- The class of RL are closed under union operation.

Proof:- Let $N_1(Q_1, \Sigma_1, \delta_1, q_1, F_1)$ recognizes a language A_1 and $N_2(Q_2, \Sigma_2, \delta_2, q_2, F_2)$ recognizes a language A_2

we can construct $N(Q, \Sigma, \delta, q_0, F)$ that recognizes $A_1 \cup A_2$.

1. $Q = q_0 \cup Q_1 \cup Q_2$
2. $\Sigma = \Sigma_1 \cup \Sigma_2$
3. q_0 is the start state of N.
4. $F = F_1 \cup F_2$
5. $\delta(q, a) = \delta_1(q, a)$ if $(q \in Q_1)$
   $= \delta_2(q_0, a)$ if $(q \in Q_2)$
   $= \{q_1, q_2\}$ if $q = q_0$ and $a = \epsilon$
   $= \phi$ if $q = q_0$ and $a \neq \epsilon$

Theorem 2 :- The class of RLs are closed under concatenation operation.

Proof :- Let $N_1 (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ recognizes a language A_1 and $N_2 (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ recognizes a language A_2. we can construct $N(Q, \Sigma, \delta, q_0, F)$ that recognizes $A_1 \cdot A_2$.

1. $Q = Q_1 \cup Q_2$
2. $\Sigma = \Sigma_1 \cup \Sigma_2$
3. $q_0 = q_1$ (initial state of N_1 will be the initial state of N).
4. $F = F_2$ (set of final states in N_2 will be the final states of N).
5. $\delta(q_0, a) = \delta_1(q_1, a)$ if $q \in Q_1$ and $q \in F_1$
   $= \delta_2(q, a)$ if $q \in Q_2$.
   $= \delta_1(q, a) \cup \{q_2\}$ if $q \in F_1$ and $a = \epsilon$

Theorem-3 :- The class of RLs are closed under star oper^n (zero or more instances)

Proof :- Let $N_1 (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ recognizeza language A_1.

We can construct $N(Q, \Sigma, \delta, q_0, F)$ that recognizes A_1^*.

1. $Q = q_0 \cup Q_1$
2. $\Sigma = \Sigma_1$
3. q_0 is the start state of N.
4. $F = q_0 \cup F_1$
5. $\delta(q_0, a) = \delta_1(q, a)$ if $q \in Q_1$ & $q \notin F_1$
   $= \delta_1(q, a)$ if $q \in F_1$ and $a \neq \epsilon$
   $= \delta_1(q, a) \cup \{q_1\}$ if $q \in F_1$ & $a = \epsilon$
   $= \{q_1\}$ if $q = q_0$ & $a = \epsilon$
   $= \phi$ if $q = q_0$ & $a \neq \epsilon$

13/10/23

EXAMPLE OF RE :-

R.E	Language
01	{01}
01+1	{01, 1}
(01+ϵ)1	{011, 1}
(0+1)*0	{0, 1}
(0+1)*	{ϵ, 0, 1, 00, 01, 10, 11, ...}

$(0+1)^*1$ $\{1, 01, 11, 001, 011, 101, 111, \ldots\}$
$= \{w \mid w \text{ ends in } 1\}$

Q// Find the RE for the languages defined over $\Sigma = \{0,1\}$ such that :-

(i) $\{w \mid w$ contains ~~only~~ a single $1\}$

ans :- 0^*10^* (there can be no or any number of zeroes before and after '1').

$= \{1, 01, 10, 001, 010, 100, 1000, 0001, 0010, \ldots\}$.

(ii) $\{w \mid w$ has ~~as~~ string '001' as substring$\}$

ans :- $(0+1)^*001(0+1)^*$

$= \{001, 0001, 01001, 0010, 00101, 00101, \ldots\}$

(iii) $\{w \mid w$ in which every 0 is followed by atleast one '1'$\}$.

ans :- $1^*(011^*)^*$

$= \{1, 011, 0111, 1011, 11011, 011011\}$

(iv) $\{w \mid w$ is a string of even length$\}$

ans :- $((0+1)(0+1))^*$

(N) $\{w \mid$ ~~where that~~ w is a string in which no. of zero is multiple of 3$\}$.

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

(vi) $\{w \mid w$ starts & ends in same symbol$\}$.

$$0 + 1 + \boxed{0\,(0+1)^*\,0} + \boxed{1\,(0+1)^*\,1}$$

$$= 0\,(\epsilon + (0+1)^*\,0) + 1\,(\epsilon + (0+1)^*\,1)$$

$$= \{0, 1, 00, 11, 000, 010, 101, 111, \ldots\}$$

(vii) $\{w \mid w$ contains any number of as and bs over $\Sigma = \{a, b\}\}$

$$(a+b)^*$$

$$\Rightarrow (a^* b^*)^*$$

(viii) $\{w \mid w$ starts in 'a' and end in 'ab'$\}$

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

(ix) $\{w \mid w$ is of even length of $\Sigma = \{0\}\}$

$$(00)^*$$

PROPERTIES :-

If R_1, R_2, R_3 are 3 REs :-

1) Associative :-

$$R_1 + (R_2 + R_3) = (R_1 + R_2) + R_3$$

$$R_1 \cdot (R_2 \cdot R_3) = (R_1 \cdot R_2) \cdot R_3$$

2) Distributive :-

$$(R_1 + R_2) R_3 = R_1 R_3 + R_2 R_3$$

$$R_1 (R_2 + R_3) = R_1 R_2 + R_1 R_3$$

3) Commutative :- (holds true only for union).

$$R_1 + R_2 = R_2 + R_1$$

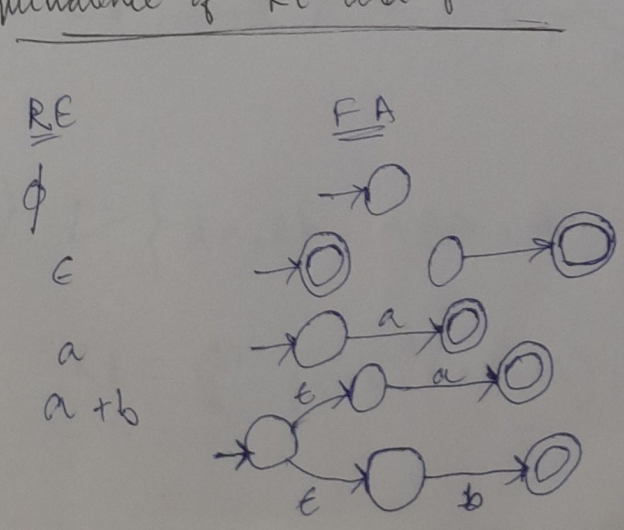
4) More properties :-

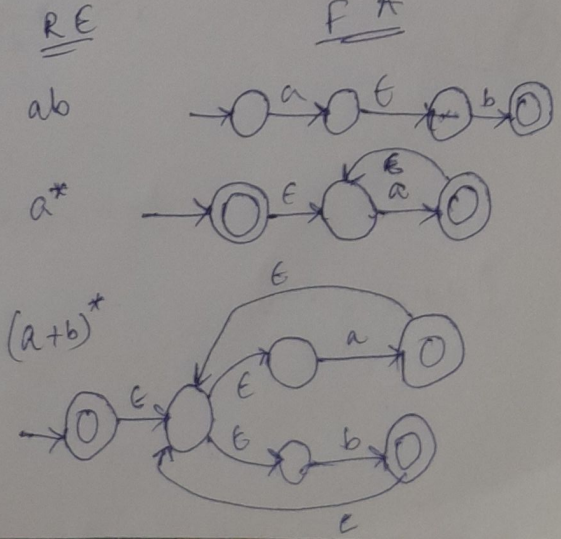
(i) $R_1 + \phi = R_1$ (iii) $(R_1^*)^* = R_1^*$

(ii) $R_1 \in = R_1$

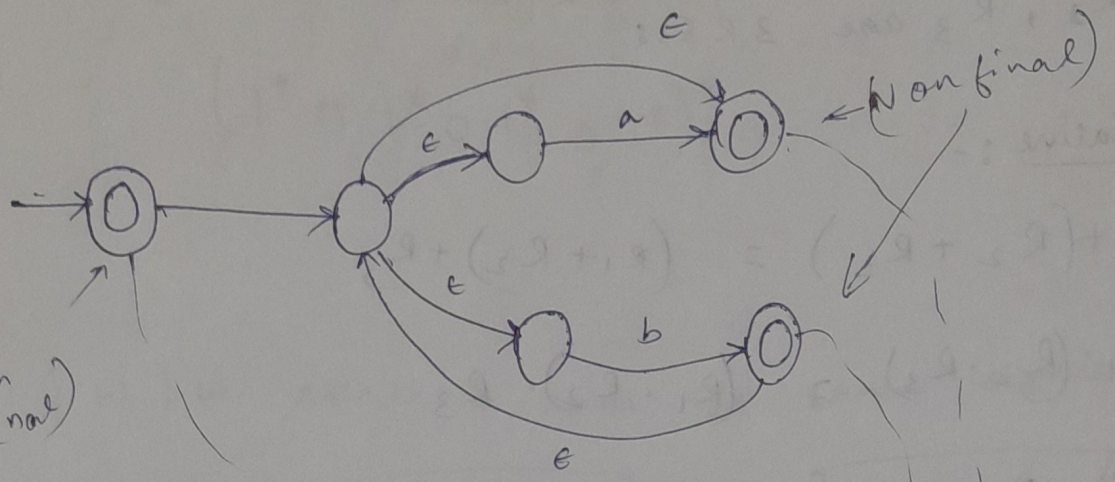
19/10/23

Equivalence of RE and finite Automata :-

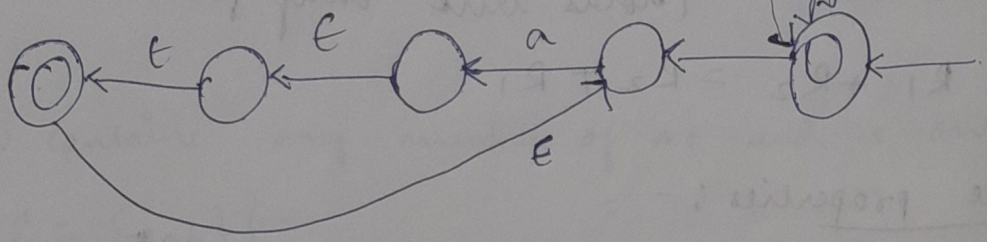




$(a+b)^*$



$(ab)^*$



Q.1) Write down the Regular expression for the language that contains 'aa' or 'bb' as substring. Find the NFA ϵ for the regular expression generated.

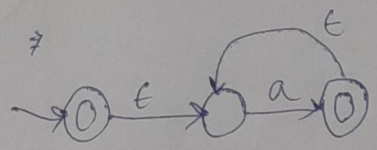
Q.2) Find the NFA ϵ for the regular expression $a^*(a \cup b)(bb)^*$

Q.3) Write down the RE for the language defined over $\Sigma = \{l, d\}$ that accepts all strings for an identifier in 'C' programming language where 'l' represents letter or '_' & 'd' represent digits. Find the NFA ϵ for the regular expression generated. Construct a DFA for the NFA ϵ design.

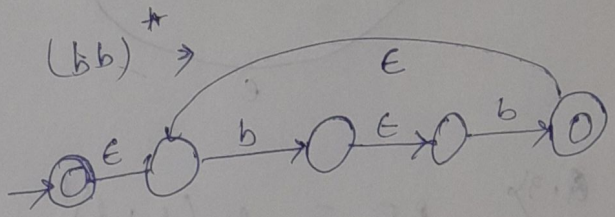
Answer

Q.2)

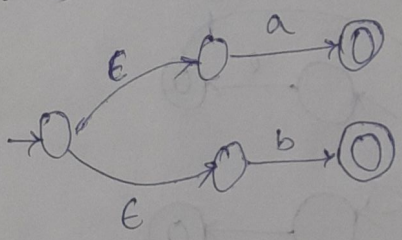
a^* :



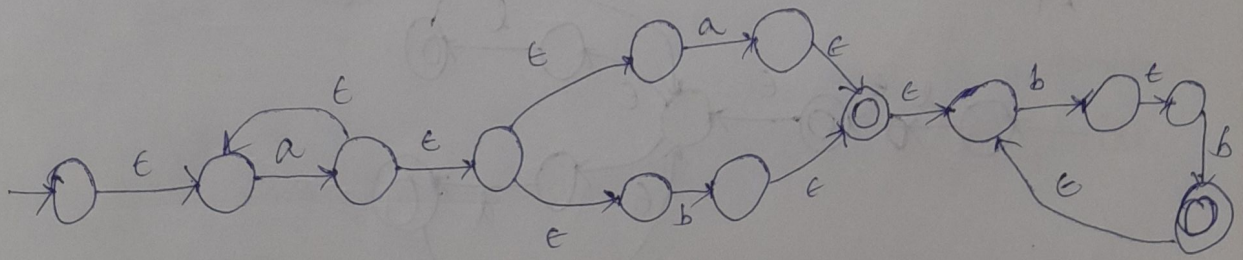
$(bb)^*$:



$(a \cup b)$:



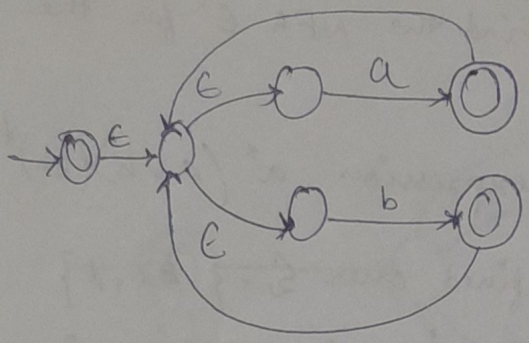
$a^*.(a \cup b)(bb)^*$



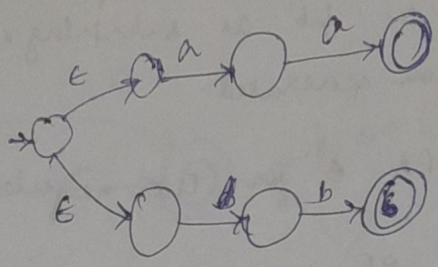
Q.1) $L = \{aa, bb, a\underline{aa}, \underline{aa}a, ba\underline{a}, \underline{aab}, bba, abb, \underline{bbb}, b\underline{bb}\}$

$RE = (a+b)^*(aa+bb)(a+b)^*$

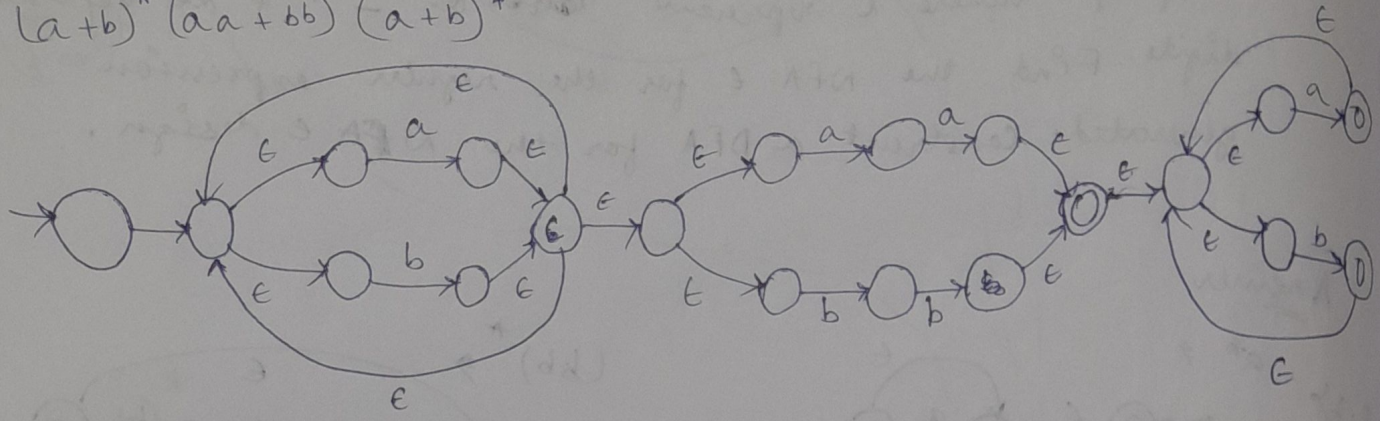
$(a+b)^*$ ⇒



$(aa+bb)$

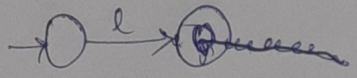


$(a+b)^*(aa+bb)(a+b)^*$

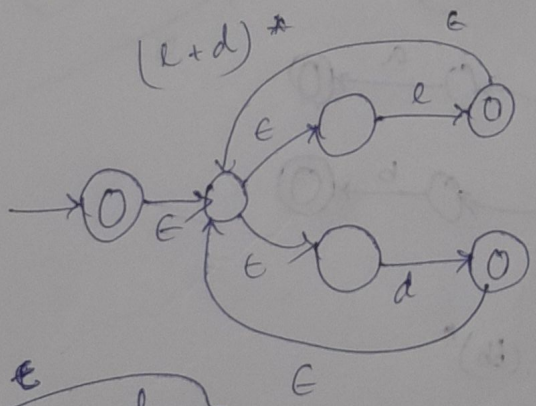


Q.3) $RE = l(l+d)^*$

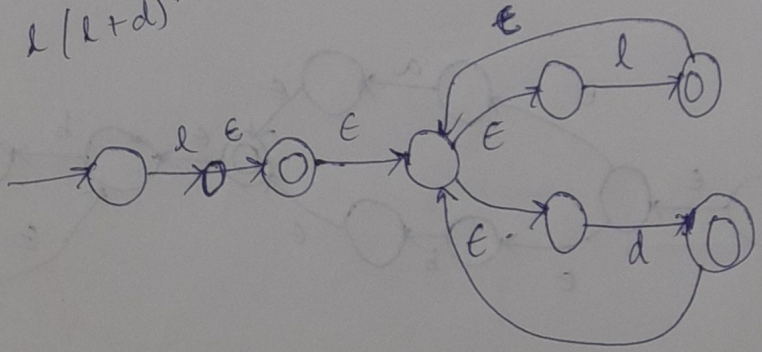
l ⇒



$(l+d)^*$



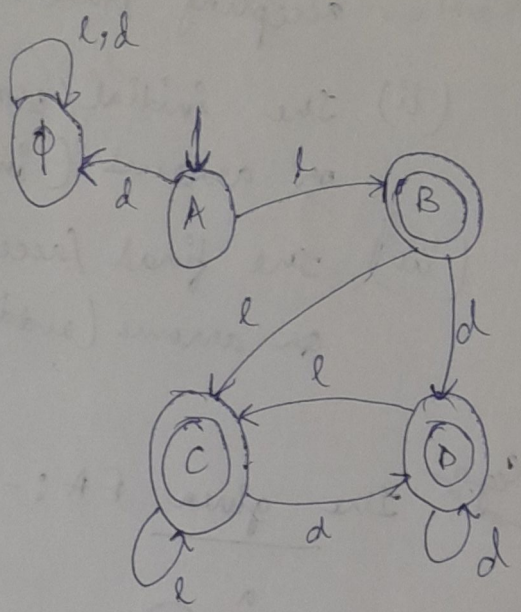
$l(l+d)^*$



26/10/23

NFA → DFA

	l	d
{1} A	{2,3,4,5,7} B	ϕ
*{2,3,4,5,7} B	{4,5,6,7} C	{4,5,7,8} D
*{4,5,6,7} C	{4,5,6,7} C	{4,5,7,8} D
*{4,5,7,8} D	{4,5,6,7} C	{4,5,7,8} D
ϕ	ϕ	ϕ



Equivalence of FA to RE :-

- If a language is regular then a regular expression can describe it and it is accepted by a FA.
- We can ~~find~~ convert a FA to RE by using a 2-state procedure :-
  (i) The FA is transformed into a new type of automatan called GNFA (Generalized Non-deterministic finite automatan)
  (ii) The RE is found out from the GNFA using state elimination method.

FA → GNFA → RE.

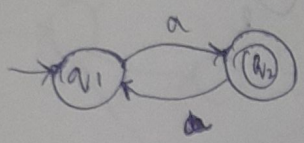
- GNFA can read block of symbols and has the following characteristics :-

(i) contains exactly one initial/start state and one final/accepting state.

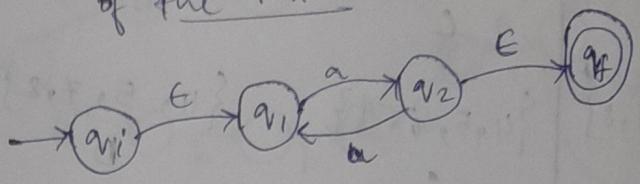
(ii) The initial/start state has no incoming transitions or arrows (indegree = 0)

(iii) The final/accepting state has no outgoing transitions or arrows (outdegree = 0)

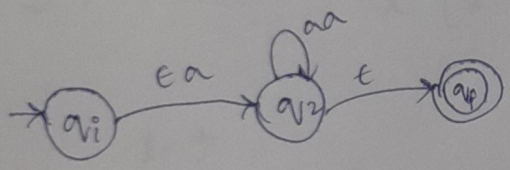
Eg:- The given FA :-



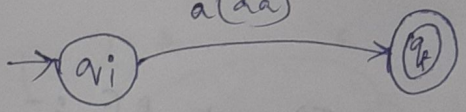
The GNFA representation of the FA is :-



Eliminating state q_1

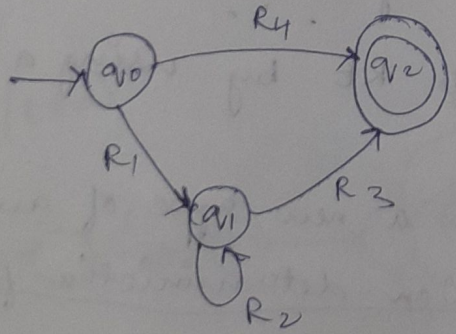


Eliminating state q_2

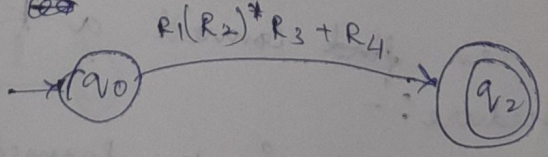


So, R.E = $a(aa)^*$

Eg :-



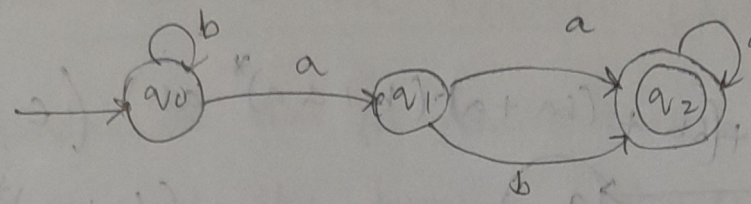
Eliminating q_1



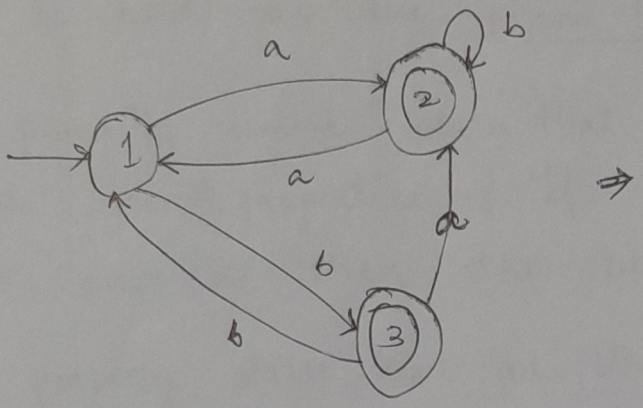
So, R.E = $R_1(R_2)^* R_3 + R_4$

Q// Find the RE for the following automata by converting them into GNFA & then by using state elimination method.

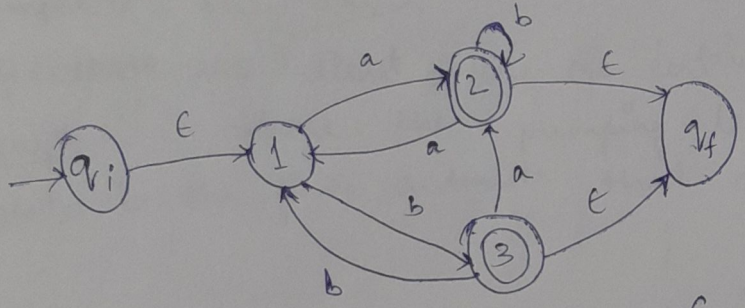
(i)



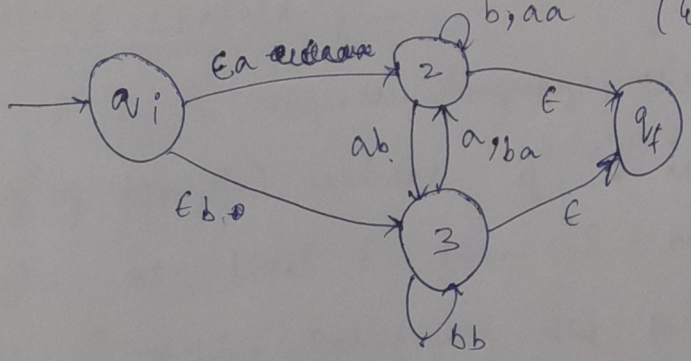
(ii)



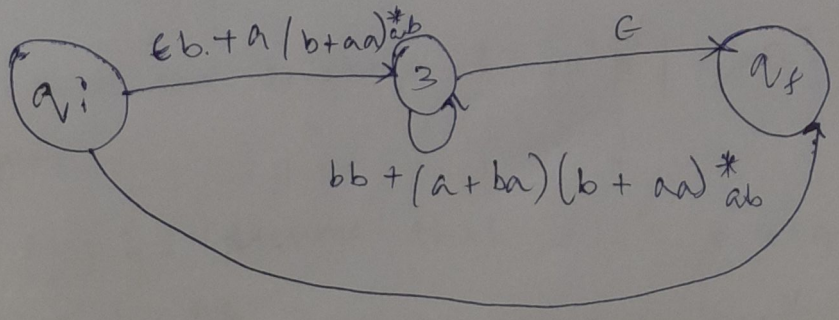
⇒



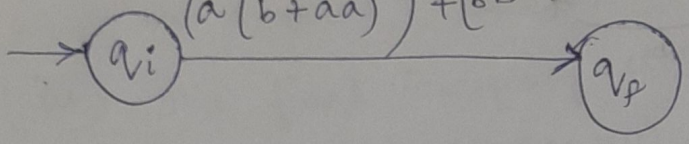
(Eliminating (1))



(Eliminating (2)) :-



So finally,



30/10/23

PUMPING LEMMA FOR REGULAR LANGUAGES :-

* The fundamental tool for proving that a language is not regular is known as the pumping lemma..
* It is based on the pigeon hole principle.
* The pumping lemma states that all regular languages have some special properties & if a language does not have those properties then the language is not regular.
* The property states that all strings in the language can be pumped if they are.
* ~~Specially~~ atleast as long as certain value called pumping length.(P), where the pumping length(P) is the number of states in the finite automata that recognizes the language.

PUMPING LEMMA :-

If 'A' is a regular language, then there is a number P (the pumping length) where if 's' is any string in A of length at least P, then 's' may be divided into 3 parts, $s = xyz$, satisfying the following conditions:-

i) for each $i \geq 0$, $xy^iz \in A$.

ii) $|y| > 0$

iii) $|xy| \leq P$

Prove that the language $A = \{a^n b^n \mid n \geq 0\}$ is not a regular language.

Proof:- step 1: Assume that A is a regular language,

step 2:- It has a pumping length.

step 3:- All strings longer than p can be pumped i.e., find string s in A such that $|s| = p$.

step 4:- Divide s into 3 parts x, y, & z.

step 5:- Show that the cond^{n}s for pumping lemma doesn't hold good for the string s.

step 6:- Getting a contradiction & proof that the length is not regular.

Ans:-

Now:-

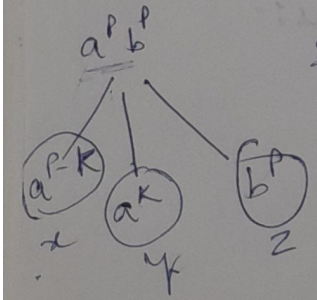
Proof:- Let $A = \{a^n b^n \mid n \geq 0\}$ i.e., 'n' no. of a's followed by 'n' no. of b's for $n \geq 0$ is a regular language.

- considering p be the pumping length given by the pumping lemma.
- Choosing string $s = a^p b^p$

$$|s| = |a^p b^p| = 2p > p$$

- s can be divided into 3 parts x, y & z, i.e. $s = xyz$ such that.

$$x = a^{p-K} \quad y = a^K \quad z = b^p \text{ for } \forall\ K > 0$$



- Here $|y| = |a^K| = K > 0$ and $|xy| = |a^{p-K} a^K| = p \leq p$
- for each $i \geq 0$, $xy^i z = a^{p-K} (a^K)^i b^p$

So, if $i = 2$, $xy^2z = a^{p-K} (a^K)^2 b^p = a^{p-K} a^{2K} b^p$

$$= a^{p+K} b^p \notin A$$

$a^{p+k} b^p$ for $\forall k > 0$, does not belong to A as the number as a's is greater than the number of b's.

$\therefore$ Hence, this is a contradiction & $A = \{a^n b^n \mid n \geq 0\}$ is not a regular language.

Q) Prove that the following languages are not regular:-

(i) $L = \{0^n 1^n 2^n \mid n \geq 0\}$

(ii) $L = \{ww \mid w \in \Sigma^*\}$

(iii) $L = \{ww^R \mid w \in \Sigma^*\}$, (iv) $\{wcw^R \mid w \in \Sigma^*\}$

(v) $L = \{a^p \mid p \text{ is prime}\}$

(vi) $L = \{0^{n^2} \mid n \geq 0\}$

(vii) $L = \{0^{2^n} \mid n \geq 0\}$

(viii) $L = \{0^{n^2} \mid n \geq 0\}$

(ix) $L = \{a^n b^{2n} \mid n \geq 0\}$.

(v) $|a^p| = |xyz| = p$

Let $y = a^i$ $\quad |xy^iz|$

For $i = 2$, $|xy^2z| = |xyz| + |y|$

$= p + 1$

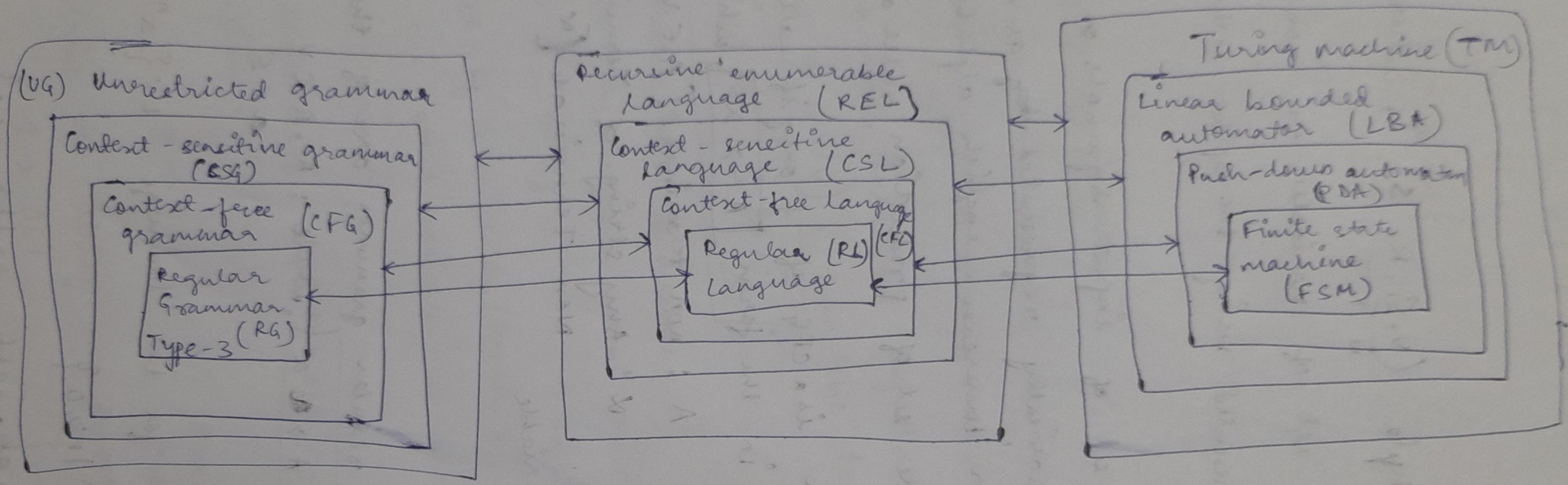
06/11/2023

CHOMSKY CLASSIFICATION

Grammar

Language

Machine



RG ⊂ CFG ⊂ CSG ⊂ UG.

RL ⊂ CFL ⊂ CSL ⊂ REL

CONTEXT FREE GRAMMAR & LANGUAGE :-

⊛ Formal defⁿ of CFG :-

→ A CFG 'G' is a 4-tuple $(\underset{N}{V}, \underset{T}{\Sigma}, \underset{P}{R}, S)$, where

① N / V → a finite set of variables or non-terminals
↳ basically represented using the capital letters of english alphabets.

② Σ or T → a finite set of input symbols or ~~alphabets~~ terminals.
↳ these are basically represented ~~by~~ as digits, symbols or lowercase english alphabets.

③ R or P → a finite set of production rules or substitution rules.
↳ a grammar is a CFG if & only if all production rules are in the form $\boxed{A \to \alpha}$, where :-
A : non terminal $(A \in V)$
α : Any string of Terminals or Non Terminals $[\alpha \in (V \cup \Sigma)^*]$

④ S → start variable
↳ $(S \in V)$

(* → All productions of a regular grammar are of the form
$\boxed{A \to aB \quad \text{or} \quad A \to a}$, where : $A, B \in V$
$a \in \Sigma$

Q// Write down the CFG for the language $L = \{a^n b^n \mid n \geq 0\}$.

$$L = \{a^n b^n \mid n \geq 0\}$$
$$= \{\epsilon, ab, aabb, aaabbb, \ldots\}$$

$$R = \{ S \to aSb \mid S \to \epsilon \}$$

$$G(L) = (\{S\}, \{a, b, \epsilon\}, R, S)$$

Q) $L = \{0^n \mid n \geq 1\}$

$\to \{ S \to 0S \mid S \to 0 \}$

09/11/23

Language of Grammar :-

If u, v, w are strings of variables & terminals $(u, v, w \in (V \cup \Sigma)^*)$ and $A \Rightarrow w$ (A derives w) is a prodn rule of the grammar, then we say that uAv yields uwv $(uAv \Rightarrow uwv)$, then the language of the grammar is $\boxed{\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}}$.

- Say that $u \overset{*}{\Rightarrow} v$ (u derives v) if $u = v$ or if a sequence $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and

$$u_1 \Rightarrow u_2 \Rightarrow u_3 \Rightarrow \cdots \Rightarrow u_k \Rightarrow u.$$

- The set of all strings (terminals), which are derivable from the start variable is known as the language of the grammar.
- The sequence of steps to obtain a string is called derivation.

$$S \to 0S1 \mid \epsilon \quad \left\{ \begin{array}{l} S \to 0S1 \\ S \to \epsilon \end{array} \right.$$

Q) Derive the string $w = 000111$

$S \to 0S1$
$\to 00S11$
$\to 000S111$
$\to 000\epsilon111$
$\to 000111$

Q) Consider the grammar S derives a AB ($S \Rightarrow aAB$).

$A \Rightarrow 0A \mid 0$ $\quad$ $B \Rightarrow 1B \mid 1$

Derive the string $w = a00\,11$ & find the language accepted by the grammar.

$S \rightarrow aAB$

$\rightarrow a0AB$

$\rightarrow a00B$

$\rightarrow a001B$

$\rightarrow a0011.$

$L = \{a01, a001, a011, a0011, \ldots\}$

$S \rightarrow aAB \rightarrow a0B \rightarrow a01$

$L = \{a0^i1^j \mid i,j \geq 1\}$

(*) Left most derivation is the process of deriving a string by applying the substitution to the left most variable.

(*) Right most derivation : vice-versa.

Eg: $S \rightarrow aAB \rightarrow aA1 \rightarrow a01.$

Eg: $aAB \rightarrow a0B \rightarrow a01$

Q) Derive the string $xxx\#yyy$, $A \rightarrow xAy$ $\quad$ $A \rightarrow B$, $B \rightarrow \#$

$A \rightarrow xAy$

$\rightarrow xxAyy$

$\rightarrow xxxAyyy$

$\rightarrow xxxByyy$

$\rightarrow xxx\#yyy.$

$L = \{x^n\#y^n \mid n \geq 0\}.$

$= \{\#, x\#y, xx\#yy, \ldots\}$

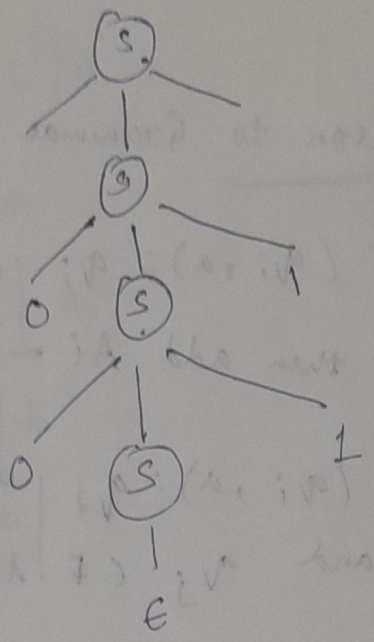
Derivation Tree / Parse tree :-

The pictorial representation of a derivation is known as a derivation tree or a parse tree.

Eg: $w = 000111$

$S \to 0S1 \mid \epsilon$

$S \to 0S1$
$\to 00S11$
$\to 000S111$
$\to 000\epsilon111$
$\to 000111$

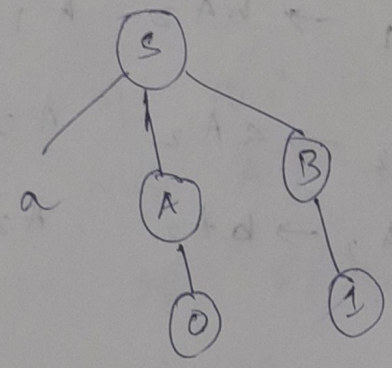


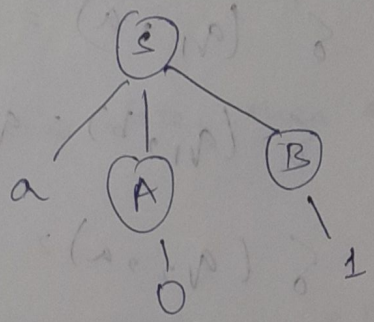
Eg: $w = a01$

$S \to AB \quad A \to 0A \mid 0 \quad B \to 1B \mid 1$

$S \to aAB$
$\to a0B$
$\to a01$

$S \to aAB$
$\to aA1$
$\to a01$





By considering the grammar $E \to E+T \mid T$

$T \to T*F \mid F \qquad F \to (E) \mid num \mid id$

Derive the string $(a+a)*a$

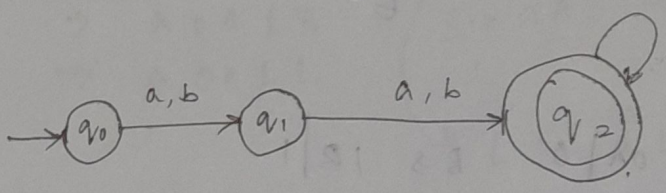
10/4/2023

Finite automaton to Grammar :-

step-1 :- if $\delta(q_i, a) = q_j$ where $q_i, q_j \in Q$ and $a \in \Sigma$, then add $A_i \rightarrow a A_j$.

step-2 :- if $\delta(q_i, a) = q_j$ where $q_i, q_j \in Q$ and $a \in \Sigma$ and $q_j \in F$. then Add $A_i \rightarrow a$

Ex



$\delta(q_0, a) = q_1$ $\quad A_0 \rightarrow a A_1$

$\delta(q_0, b) = q_1$ $\quad A_0 \rightarrow b A_1$

$\delta(q_1, a) = q_2$ $\quad A_1 \rightarrow a A_2$ $\quad A_1 \rightarrow a$

$\delta(q_1, b) = q_2$ $\quad A_1 \rightarrow b A_2$ $\quad A_1 \rightarrow b$

$\delta(q_2, a) = q_2$ $\quad A_2 \rightarrow a A_2$ $\quad A_2 \rightarrow a$

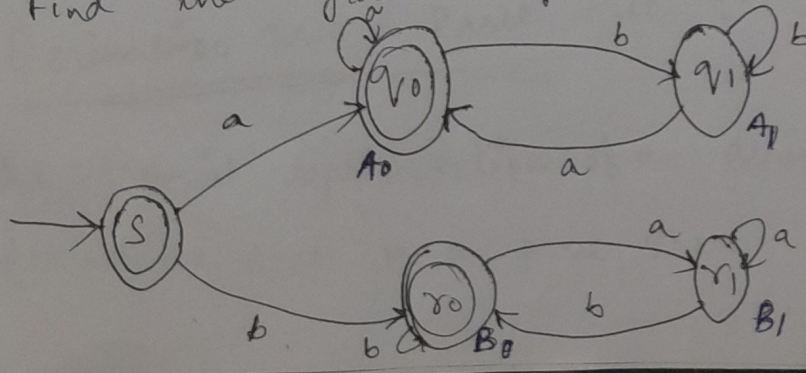
$\delta(q_2, b) = q_2$. $\quad A_2 \rightarrow b A_2$ $\quad A_2 \rightarrow b$.

$A_0 \rightarrow a A_1 \mid b A_1$

$A_1 \rightarrow a A_2 \mid b A_2 \mid a \mid b$

$A_2 \rightarrow a A_2 \mid b A_2 \mid a \mid b$.

Ex Find the grammar for the following automata.



$S \to aA_0 \mid bB_0 \mid a \mid b \mid \epsilon$

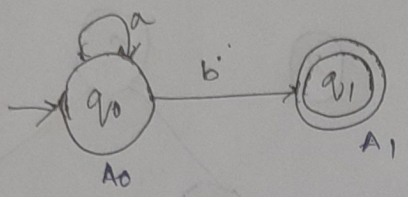
$A_0 \to aA_0 \mid bA_1 \mid a$

$A_1 \to aA_0 \mid bA_1 \mid a$

$B_0 \to aB_1 \mid bB_0 \mid b.$

$B_1 \to aB_1 \mid bB_0 \mid b.$

Q)



$A_0 \longrightarrow aA_0 \mid bA_1 \mid b.$

Rule-1 Step-3 :- If $\delta(q_i, a) = q_j$ then $A_i \to aA_j$

Rule-2 :- If q_i ($q_i \in F$) is an accepting state, then add a rule $A_i \to \epsilon$.

Rule-3 :- If $\delta(q_i, a) = q_j$ and $q_j \in F$ then $A_i \to a$

$A_0 \to aA_1 \mid bA_1$

$A_1 \to aA_2 \mid bA_2 \mid \epsilon$

$A_2 \to aA_2 \mid bA_2 \mid \epsilon.$

Q) Consider a grammar :- $G = (\{S\}, \{a, b\}, \{S \to aSbS \mid bSaSS \mid \epsilon, S\})$

$w = abab.$

find the derivn & parse tree.

$S \rightarrow aSb \mid bSa \mid SS \mid \epsilon$ $\quad w = abab$.

① $S \rightarrow aSb$

$\rightarrow abSab$

$\rightarrow ab\epsilon ab$

$\rightarrow abab$.

② $S \rightarrow SS$

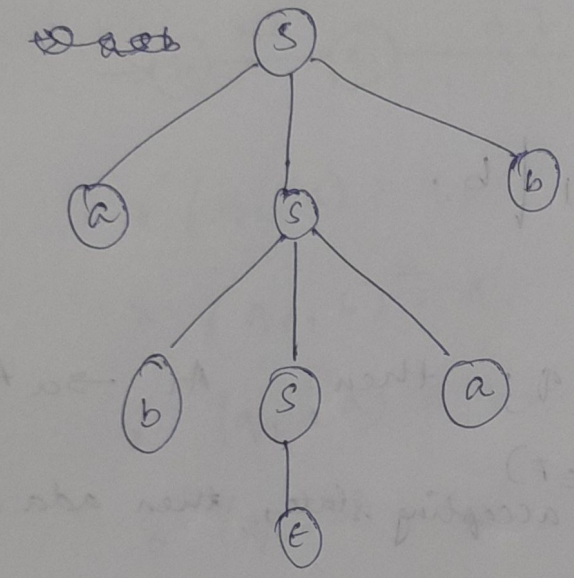
$\rightarrow aSbS$

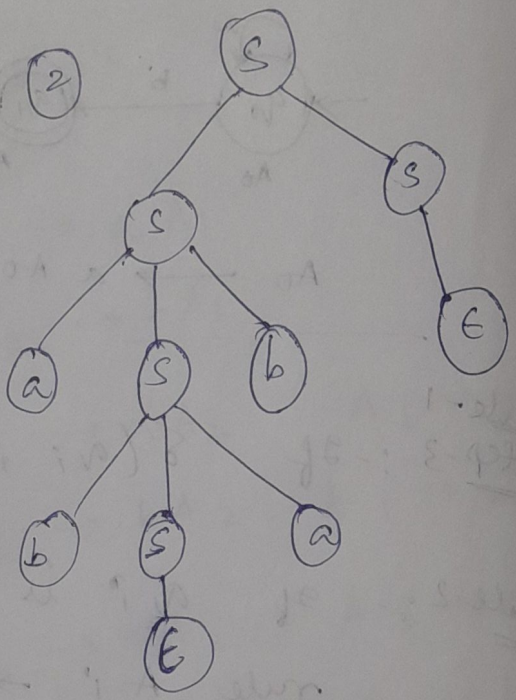
$\rightarrow aSb\epsilon$

$\rightarrow abSab\epsilon$

$\rightarrow ab\epsilon ab$

$\rightarrow abab$.

① 

② 

12/11/22

Consider the following grammar, find the deriv^n & parse tree.

Justify the ~~Lastname~~ grammar is ambiguous or not.

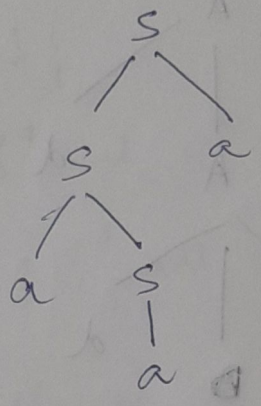
$W = aaa$ $\quad S \to aS \mid Sa \mid a$.

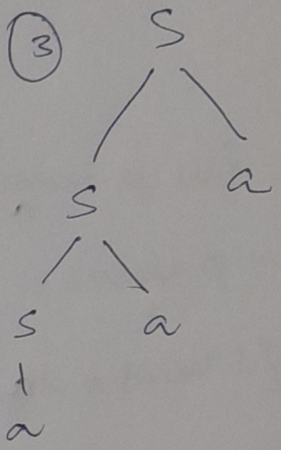
① $S \to a\underline{S}$
$\to aa\underline{S}$
$\to aaa$

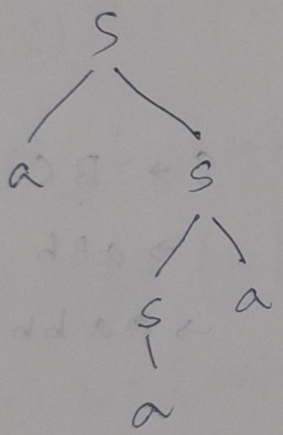
② $S \to a\underline{S}$.
$\to a\underline{S}a$
$\to aaa$

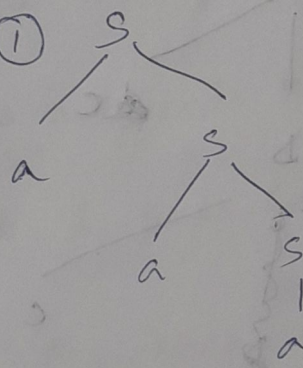
③ $S \to \underline{S}a$
$\to \underline{S}aa$
$\to aaa$

④ $S \to \underline{S}a$
$\to a\underline{S}a$
$\to aaa$

④


③


②


①


$\therefore$ ambiguous

Inherently Ambiguous Language :-

- In some cases, few context free languages can only be generated using ambiguous ~~languages~~ grammars.
- These grammars are known as inherently ambiguous grammars.

- Eg :- $L = \{a^n b^n c^m d^m \mid n \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1\}$

$B \to aBb \mid ab$. $\quad C \to aCd \mid cd$. $\quad S \to BC$

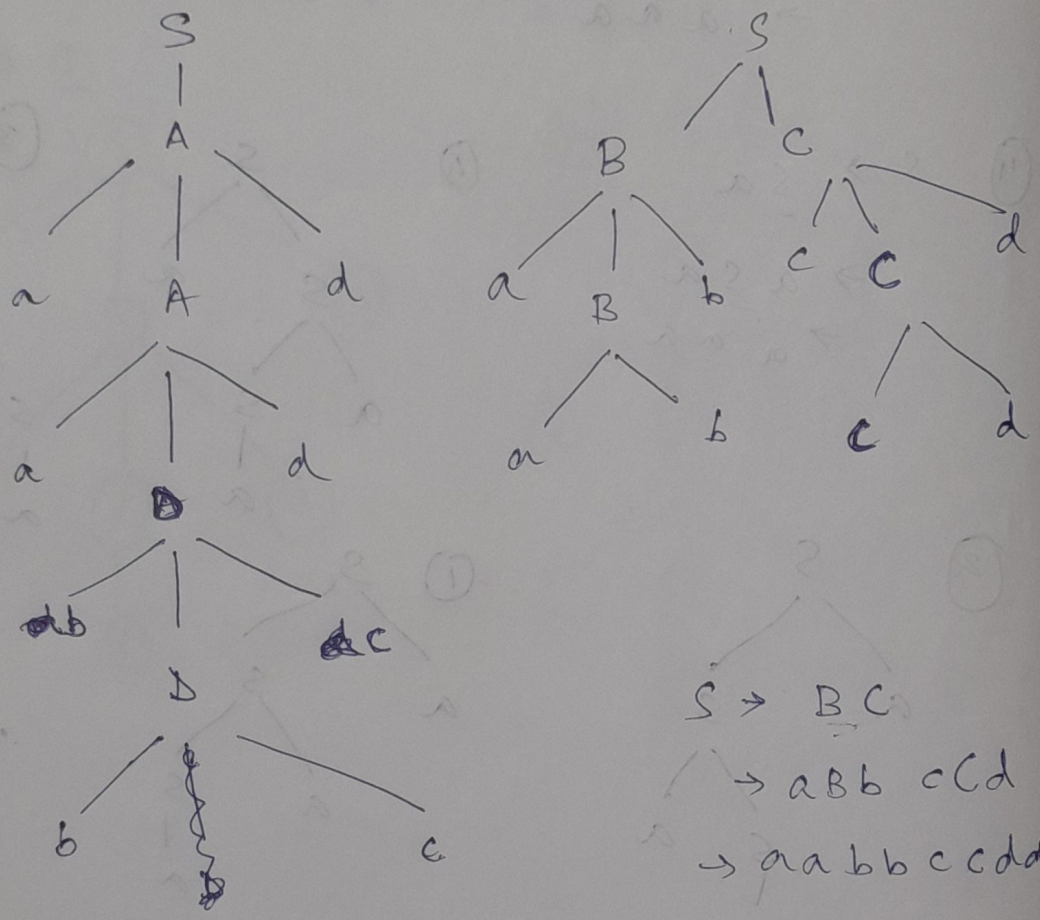
$D \rightarrow bDc \mid bc$ $\quad A \rightarrow aAd \mid aDd$

$S \rightarrow BC \mid A$

Eg:- $w = aabbccdd$.

2 different LMD/RMD

⇒ The grammar will be ambiguous.



$S \rightarrow BC$
$\rightarrow aBb\ cCd$
$\rightarrow aabbccdd$.

$S \rightarrow A$
$\rightarrow aAd$
$\rightarrow aaDdd$
$\rightarrow aabDcdd$
$\rightarrow aabbccdd$

Designing CFG :-

Design CFG for the following languages defined over $\Sigma = \{a, b\}$.

(i) $L = \{w \mid w$ are strings of length atleast $2\}$

(ii) The regular expression for L is $(a+b)^*$

(iii) $L = \{a^n b^n \mid n \geq 1\}$

(iv) $L = \{a^n b^m \mid m, n \geq 1\}$.

(v) $L = \{(ab)^n \mid n \geq 1\}$

(vi) $L = \{ww^R \mid w \in \Sigma^*$ and $w^R =$ reverse of $w\}$

(vii) $L = \{wcw^R \mid w \in \Sigma^*$ and $w^R =$ reverse of $w\}$

(viii) $L = \{w \mid w$ are strings of length atmost $2\}$

(ix) $L = \{w \mid w$ starts and ends with different symbol$\}$

(x) $L = \{w \mid w$ starts & ends in same symbol$\}$

(xi) $L = \{w \mid w$ is of even length$\}$.

(xii) $L = \{w \in \Sigma^* \mid |w| \bmod 3 = 0\}$

(xiii) $L = \{a^n b^{2n} \mid n \geq 1\}$.

(*) **Property of union :-** If A & B are 2 variables representing grammars G_1 and G_2 then their union is represented $G = G_1 \cup G_2$ as $S \to A \mid B$ where S is a new variable.

Property of concatenation :- If A and B are 2 start variables representing Grammars G_1 & G_2 then their concatenation is represented $G = G_1 \cdot G_2$ as $S \to AB$, where S is a new variable.

Kleen closure property :- If A is a start variable represent. grammar G_1 then their Kleen closure/ star is represented as $G = G_1^*$.

$$S \to AS \mid \epsilon$$

positive closure / plus ... as $G = G^+$

$$S \to AS \mid A$$

Answers :-

(i) $L = \{aa, ab, ba, bb, aaa, aab, aba, abb, \ldots\}$

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

$S \to BA$

$A \to CA \mid \epsilon$

$B \to CC$

$C \to a \mid b$

(ii) $S \to AS \mid \epsilon$, where $A \to a \mid b$.

OR

$S \to aS \mid bS \mid \epsilon$

(iii) $S \to aSb \mid ab$.

(iv) $a \to aA \mid a$

$b \to bB \mid b$.

$S \to AB$.

OR

$S \to aS \mid bS \mid ab$

16/11/23

(iv)

L_1 L_2

G_1 G_2

start variable S_1 S_2

$G(L_1 \cup L_2) = S \to S_1 \mid S_2$

$G(L_1 \cdot L_2) = S \to S_1 S_2$

$G(L_1^*) = S \to S_1 S \mid \epsilon$

$G(L_1^+) = S \to S_1 S \mid S_1$

$L = \{ (ab)^n \mid n \geq 1 \}$

$= \{ ab, abab, ababab, \ldots \}$

$S \to abS \mid ab$

(vi) Even length palindrome $\{ w w^R \mid w \in \{a,b\}^* \}$

$= \{ \underset{S}{\epsilon}, \underset{\underset{S}{w\ w^r}}{aa}, \underset{\underset{S}{w\ w^r}}{bb}, aaaa, abba, baab \ldots \}$.

$S \to aSa \mid bSb \mid \epsilon$

(vii) $\{ wcw^r \mid w \in \{a,b\}^* \}$.

$= \{ c, aca, bcb, abcba, \ldots \}$

$S \to aSa \mid bSb \mid c$

(viii) $\{ w \mid w$ is a palindrome string $\}$

$S \to aSa \mid bSb \mid \epsilon \mid a \mid b$

(viii) $L = \{ \epsilon, a, b, aa, bb, ab, ba \}$.

$S \to AA \mid A \mid \epsilon$

$A \to a \mid b$

(ix) $L = \{ ab, ba, aabb, baab, \ldots \}$

$S \to aAb \mid bAa$

$A \to aA \mid bA \mid \epsilon$

(x) $L = \{\epsilon, a, b, aa, bb, aba, bab, aaaa, babb, \ldots\}$

$S \to aAa \mid bAb \mid a \mid b \mid \epsilon$

$A \to aA \mid bA \mid \epsilon$

(xi) $L = \{ \epsilon, aa, bb, ab, ba, aaaa, bbbb, \ldots \}$

$S \to AS \mid \epsilon$

$A \to BB$

$B \to a \mid b$

(xii)

$S \to AS \mid \epsilon$

$A \to BBB$

$B \to a \mid b$

(*) Ans $w \mid |w| \bmod 3 = 1$

~~$S \to a \mid b \mid aAS \mid bAS$~~

$S \to B \mid BAS$

$A \to BBB$

$B \to a \mid b$

(xiii) $L = \{ abb, aabbbb, aaabbbbbb, \ldots \}$.

$S \to aSbb \mid abb$.

Q) Find the CFG for the language,

$L = \{ 1^x 2^y 3^y 4^x \mid x, y \geq 1 \}$.

⊛ Simplific^n or Reduc^n of CFG :-

→ If a CFG 'G' may not use all the symbols for the deriv^n of the string starting with the start variable.

→ So we may eliminate some symbols from the produc^n of G.

→ ① We will consider to eliminate useless symbols / useless variables

② Null productions / ϵ-productions.

③ Unit produc^{n}s.

① elimin^n of useless symbols / variables :-

- A variable 'x' is useful for a grammar G if there is a deriv^n of the form $S \xrightarrow[G]{*} \alpha X \beta \xrightarrow[G]{*} w$, where,

  w : a string of terminals $[w \in \Sigma^*]$

  α, β : are any strings of terminals or non terminals

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

  X : variable.

  S : start variable $[X, S \in V]$

otherwise 'x' is useless.

Q) Consider the grammar and eliminate the useless symbols if existing. $S \to AB \mid a \quad A \to a$

$S \to AB \mid a$

$A \to a$.

B :- Here no derivable string from B, so B is useless & needs to be eliminated after eliminating B, we get $S \to a$

$A \to a$.

→ Now A is useless as it is unreachable from the start variable. After eliminating A, we get $S \to a$.

17/11/23

Q) Consider the following grammar & eliminate the useless symbols if exists.

$S \to aAa \mid aBCD \mid bB$.

$A \to aS \mid bD$

$B \to aBa \mid b$

$C \to abb \mid DD$

$D \to aDa$

ans :- D is not deriving any terminal string. So D is useless and to be eliminated, After eliminating variable D. The productions are:-

$S \to aAa \mid bB$.

$A \to as$

$B \to aBa \mid b$

$C \to abb$.

→ Now C is unreachable from S. So C is useless and to be eliminated. After eliminating C, the grammar is :-

$S \rightarrow aAa \mid bB$

$A \rightarrow aS$

$B \rightarrow aBa \mid b.$

Elimination of Null (ϵ) production

A null production is of the form $A \rightarrow \epsilon$, for some variable 'A'. ($A \in V$), & to be eliminated by using a production $A \rightarrow \alpha$, where α = string of terminals & non terminals. $\hookrightarrow [\alpha \in (V \cup \Sigma)^*]$

⇒ Considering the grammar $S \rightarrow aS \mid bA$

$A \rightarrow aA \mid \epsilon$

$A \rightarrow \epsilon$ is a null production which is to eliminated

$A \rightarrow \epsilon$.

$A \rightarrow \alpha$ ⟹ string found by replacing A in right hand side of a production by ϵ.

So in $S \rightarrow aS \mid bA$

$A \rightarrow aA \mid \epsilon$

putting ϵ in right hand side in place of A :-

$S \rightarrow aS \mid bA \mid b$

$A \rightarrow aA \mid a.$

Q Eliminate the null productions from the following grammar :-

(i) $S \to ABC$

$A \to BC \mid a$

$B \to bAC \mid \epsilon$

$C \to cAB \mid \epsilon$

ans:- (1) $B \to \epsilon$.

$S \to ABC \mid AC$

$A \to BC \mid a \mid C$

$B \to bAC$

$C \to cAB \mid cA \mid \epsilon$

(2) $C \to \epsilon$.

$S \to ABC \mid AC \mid AB \mid A$

$A \to BC \mid a \mid C \mid B \mid \epsilon$

$B \to bAC \mid bA$.

$C \to cAB \mid cA$

(3) $A \to \epsilon$.

$S \to ABC \mid AC \mid AB \mid A \mid BC \mid C \mid B \mid \epsilon$

$A \to BC \mid a \mid C \mid B$

$B \to bAC \mid bA \mid bC \mid b$

$C \to cAB \mid cA \mid cB \mid c$

(ii) $S \to AaB \mid aaB$

$A \to \epsilon$

$B \to bbA \mid \epsilon$

(iii) $S \to AB$

$A \to aAA \mid \epsilon$

$B \to bBB \mid \epsilon$

(iv) $S \to aAB$

$A \to aAA \mid AA \mid \epsilon$

$B \to bBB \mid \epsilon$

(v) $S_0 \to S$

$S \to ASA \mid aB.$

$A \to B \mid S$

$B \to b \mid \epsilon$

(vi) $S \to AB \mid AC$

$A \to aAb \mid bAa \mid a$

$B \to bbA \mid aaB \mid AB$

$C \to abcA \mid aDB.$

$D \to bD \mid aC \mid \epsilon$

(vii) $S \to ABC \mid BaB.$

$A \to aA \mid BaC \mid aaa$

$C \to @cA \mid aC \mid \epsilon$

Elimination of unit produc^n :-

The produc^n of the form $A \to B$ for the variables A & B $[A, B \in N]$ is called a unit produc^n.

→ Substitution method is used to eliminate unit produc^n from the CFG.

Q) Eliminate the unit produc^n :-

$S \to A \mid bb$

$A \to B \mid b$

$B \to S \mid a$

20/11/23

ans:- The unit produc^{n}s are $S \to A$, $A \to B$ and $B \to S$.

$S \to A$ gives $S \to b$ and $S \to a$

$A \to B$ gives $A \to a$ and $A \to bb$

$S \to B$ gives $S \to a$ & $S \to S$ ✗

$A \to S$ gives $A \to bb$ and $A \to A$ ✗

$B \to S$ gives $B \to bb$ and $B \to b$

$B \to S$ gives $B \to b$ and $B \to B$ ✗

After elimination of the unit produc^{n}s, the grammar is:-

$S \to a \mid b \mid bb$

$A \to a \mid b \mid bb$

$B \to a \mid b \mid bb$

Q// Eliminate the unit produc^{n}s from the grammar

$S \to AaB$

$A \to a \mid bb \mid B$

$B \to bc \mid A$

ans:- Here unit produc^{n}s are:- $A \to B$

$B \to A$

$A \to B$ gives $A \to bc$

$B \to A$ gives $B \to a \mid bb$

$\therefore$ $S \to AaB$

$A \to a \mid bb \mid bc$

$B \to a \mid bb \mid bc$

Normal form :-

These are basically classified into 2 types:-

1. Chomsky Normal Form (CNF)
2. Greibach Normal Form (GNF)

Transformⁿ into Chomsky Normal Form (CNF):-

- A context free grammar $G(V, \Sigma, R, S)$, is in CNF if all the produc^{n}s are of the form

$A \to BC$ or $A \to a$, where $A, B, C \in V$ and $a \in \Sigma$ (terminals)

STEPS TO TRANSFORM a GRAMMAR INTO CNF:-

step 1:- Add a new start variable S_0 along the produc^{n} rule $S_0 \to S$ to the grammar if the start variable S of the grammar appears at the right hand side of ~~the~~ any produc^{n}.

step 2:- Eliminate all the null produc^{n}s of the form $A \to \epsilon$ where $A \to V$ from the grammar.

step 3:- Eliminate all the unit produc^{n}s of the form $A \to B$ from the grammar.

step 4:- Bring all the produc^{n}s into the proper format i.e., $A \to BC$ or $A \to a$.

Q) Convert the following Grammar into CNF:-

$S \to ASA \mid aB$

$A \to B \mid S$

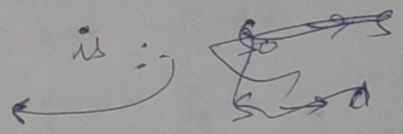
$B \to b \mid \epsilon$.

Ans:- step 1:- $S_0 \to S$ // Adding a new start variable So the grammar is :-

$S \to ASA \mid aB$

$A \to B \mid S$

$B \to b \mid \epsilon$.



step-2 :- eliminating the null product $B \to \epsilon$, the grammar is

$S_0 \to S$

$S \to ASA \mid aB \mid a$

$A \to B \mid S \mid \epsilon$

$B \to b$.

eliminating the null product $A \to \epsilon$,

$S_0 \to S$

$S \to ASA \mid SA \mid AS \mid S \mid aB \mid a$.

$A \to B \mid S$

$B \to b$

step-3 :- To eliminate the unit product :-

$S_0 \to S$, $S \to S$, $A \to B$, $A \to S$.

$S_0 \to ASA \mid SA \mid AS \mid S \mid aBa$

$S \to ASA \mid SA \mid AS \mid aB \mid a$

$A \to b \mid ASA \mid SA \mid AS \mid aB \mid a$

$B \to b$

The grammar in CNF is :-

$S_0 \to AV_1 | SA | AS | T_1B | a$ $\quad V_1 \to SA$

$S \to AV_1 | SA | AS | T_1B | a$ $\quad T_1 \to a$

$A \to AV_1 | SA | AS | T_1B | a | b$

$B \to b$

Q/ Convert the following grammar into CNF :-

(i) $S \to AACD$

$A \to aAb | \epsilon$

$C \to aC | \epsilon$

$D \to aDa | bDb | \epsilon$

(ii) $A \to BAB | B | \epsilon$

$B \to 00 | \epsilon$

23/11/23

(i) $S \to AACD$

$A \to aAb | \epsilon$

$C \to aC | \epsilon$

$D \to aDa | bDb | \epsilon$

→ Eliminating $D \to \epsilon$

$S \to AACD | AAC$

$A \to aAb | \epsilon$

$C \to aC | \epsilon$

$D \to aDa | bDb | aa | bb$

Elim $C \to \epsilon$.

$S \to AACD | AAC | AAD | AA$

$A \to aAb | \epsilon$ $\quad C \to aC$ $\quad D \to$ same

Elim $A \to \epsilon$

$S \to AACD \mid AAC \mid AAD \mid AA \mid ACD \mid CD \mid AC \mid C \mid AD \mid D \mid A \mid \epsilon$

$A \to aAb \mid ab$

$C \to aC \mid a$

$D \to aDa \mid bDb \mid aa \mid bb$

⇒ Elim. unit productⁿs :-

$S \to A \quad S \to C \quad S \to D$

$S \to AACD \mid AAC \mid AAD \mid ACD \mid \epsilon$

$S \to AA \mid AC \mid AD \mid CD$

$S \to aAb \mid ab \mid aC \mid a \mid aDa \mid bDb \mid aa \mid bb.$

⇒ Bringing the grammar into CNF.

$S \to V_1 V_2 \mid A V_3 \mid A V_4 \mid A V_2 \mid \epsilon$

$S \to AA \mid AC \mid AD \mid CD.$

$S \to T_1 V_5 \mid T_1 T_2.$

$S \to T_1 C \mid a.$

$S \to T_1 V_6 \mid T_2 V_7 \mid T_1 T_1 \mid T_2 T_2$

$A \to T_1 V_5 \mid T_1 T_2.$

$C \to T_1 C \mid a$

$D \to T_1 V_6 \mid T_2 V_7 \mid T_1 T_1 \mid T_2 T_2$

$V_1 \to AA$

$V_2 \to CD$

$V_3 \to AC$

$V_4 \to AD$

$T_1 \to a$

$T_2 \to b$

$V_5 \to A T_2$

$V_6 \to D T_1$

$V_7 \to D T_2$

23/11/22

PDA :- PUSH DOWN AUTOMATA

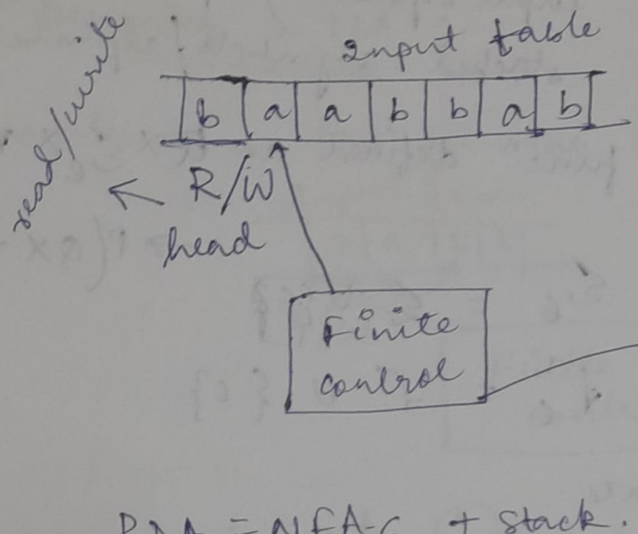
Regular grammar —described→ FSM ←recognized— Regular Language

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

DFA: $Q \times \Sigma \to Q$

NFA: $Q \times \Sigma \to P(Q)$

NFA-ϵ: $Q \times \Sigma_\epsilon \to P(Q)$



stack

Access ⇒ LIFO

oper^n : PUSH/POP

⇓

done at the top of the stack

PDA = NFA-ϵ + stack.

CFG —described→ PDA ←recognized— CFL

$(Q, \Sigma, \delta, q_0, f) \cup (\Gamma)$

↳ Big Gamma [stack alphabet]

⇒ A PDA is a non-deterministic finite Automaton (NFA-ϵ) have an extra component :- stack. The stack provides additional memory beyond the limited amount of memory available in the finite control.

- The stack allows the PDA to recognize some of the non-regular languages.

PDA = NFA-ϵ + stack

Formal Def^n of PDA

→ A PDA 'P' is a six tuple,

$$P = (Q, \Sigma, \Gamma, \delta, q_0, F), \text{ where :-}$$

Q = Finite set of states.

Σ = Finite set of input symbols / alphabets.

Γ = Finite set of stacked symbols / alphabets.

δ = Transition func^n defined as $Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to P(Q \times \Gamma^*)$

where, $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$

$\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$

q_0 = initial state.

F = Finite set of accepting / final states.

(*) * symbol [$] → or 'B', indicates the initial component of the stack.

Stack oper^{n}s :-

(1) PUSH :- Read a Push X.

$$a, \epsilon \to X$$

(input) (stack input)

(2) POP :- Read a stack content X

$$a, X \to \epsilon$$

(input) (stack)

(3) No oper^n :- Read a

$$a, \epsilon \to \epsilon$$

(4) Replacement :- Read a Push X
stack content Y

$$a, Y \to X.$$

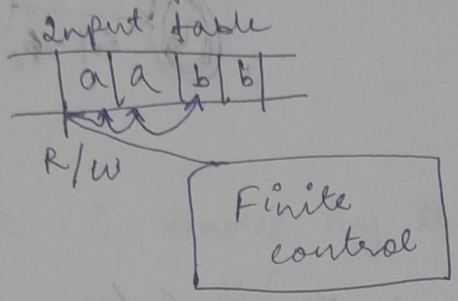
where, $a \in \Sigma$ (input symbol)
$X, Y \in \Gamma$ (stacks symbols).

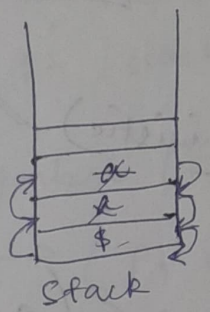
24/11/22

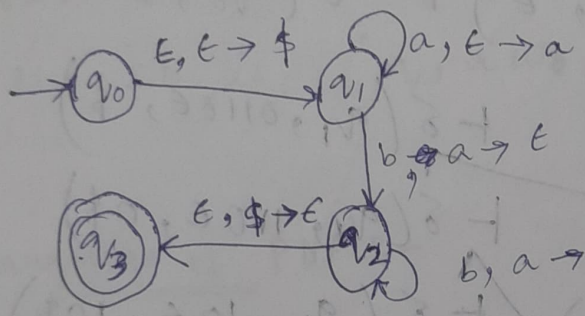
Designing a PDA :-

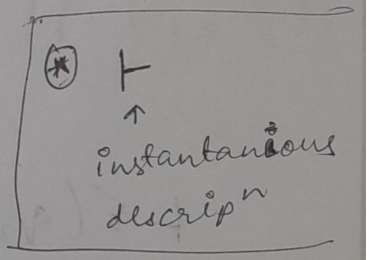
→ construct a PDA for the language $L = \{a^n b^n / n \geq 0\}$

ans:-







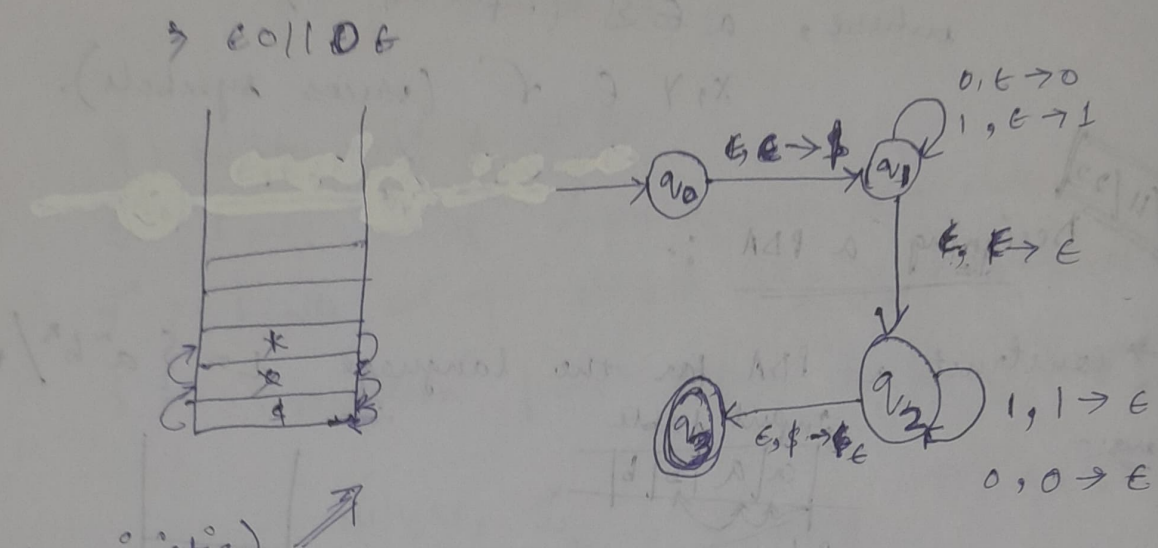


Q) Process the string $w = aaibb\epsilon$ with the above designed PDA.

$$\begin{aligned}
\delta(q_0, w, \epsilon) &\vdash \delta(q_0, \epsilon aabb\epsilon, \epsilon) \\
&\vdash \delta(q_1, aabb\epsilon, \$\epsilon) \\
&\vdash \delta(q_1, abb\epsilon, a\$) \\
&\vdash \delta(q_1, bb\epsilon, aa\$) \\
&\vdash \delta(q_2, b\epsilon, a\$) \\
&\vdash \delta(q_2, \epsilon, \$) \\
&\vdash \delta(q_3, \epsilon, \epsilon)
\end{aligned}$$

Q) Design a PDA that recognizes a language $L = \{ww^R / w \in \{0,1\}^*\}$

and process a string: 0110



(Non deterministic)

$\delta(q_0, w, \epsilon) \vdash \delta(q_0, \epsilon 0110\epsilon, \epsilon)$

$\vdash \delta(q_1, 0110\epsilon, \$)$

$\vdash \delta(q_1, 110\epsilon, 0\$)$

$\vdash \delta(q_1, 10\epsilon, 10\$)$

$\vdash \delta(q_2, 10\epsilon, 10\$)$

$\vdash \delta(q_2, 0\epsilon, 0\$)$

$\vdash \delta(q_2, \epsilon, \$)$

$\vdash \delta(q_3, \epsilon, \epsilon)$

$\vdash \delta(q_2, 0110\epsilon, \$)$

(The path is stuck)

$\vdash \delta(q_2, 110\epsilon, 0\$)$

(The path is stuck)

21/11/23

Instatenous Description (ID).

To formally describe the configuration of a PDA at a given instance, we define an Instatenous description (ID):

- The ID is defined as a 3 tuple / triple
  - ⇒ (q, w, γ), where, q :- a state, $q \in Q$
    w :- remaining input
    γ :- stack content.
- The ID is denoted by the symbol $(\vdash)$.
- For transition $\delta(q, w, \gamma) \rightarrow (p, \alpha)$, then the ID is $(q, aw, X\beta) \vdash (p, w, \alpha\beta)$

Language of PDA :-

$$L_{PDA} = \{ w \in \Sigma^* \mid (q_0, w, \epsilon) \vdash^* (q, \epsilon, \epsilon), \text{ where } q \in F \}.$$

⇒ For acceptance by empty stack :-

$$L_{PDA} = \{ w \in \Sigma^* \mid (q_0, w, \epsilon) \vdash^* (q, \epsilon, \epsilon), \text{ where } q \in Q \}.$$

⇒ For acceptance by Final state :-

$$L_{PDA} = \{ w \in \Sigma^* \mid (q_0, w, \epsilon) \vdash^* (q, \epsilon, \alpha), \text{ where } q \in F \text{ and } \alpha \in \Gamma^* \}.$$

↳ Big gamma

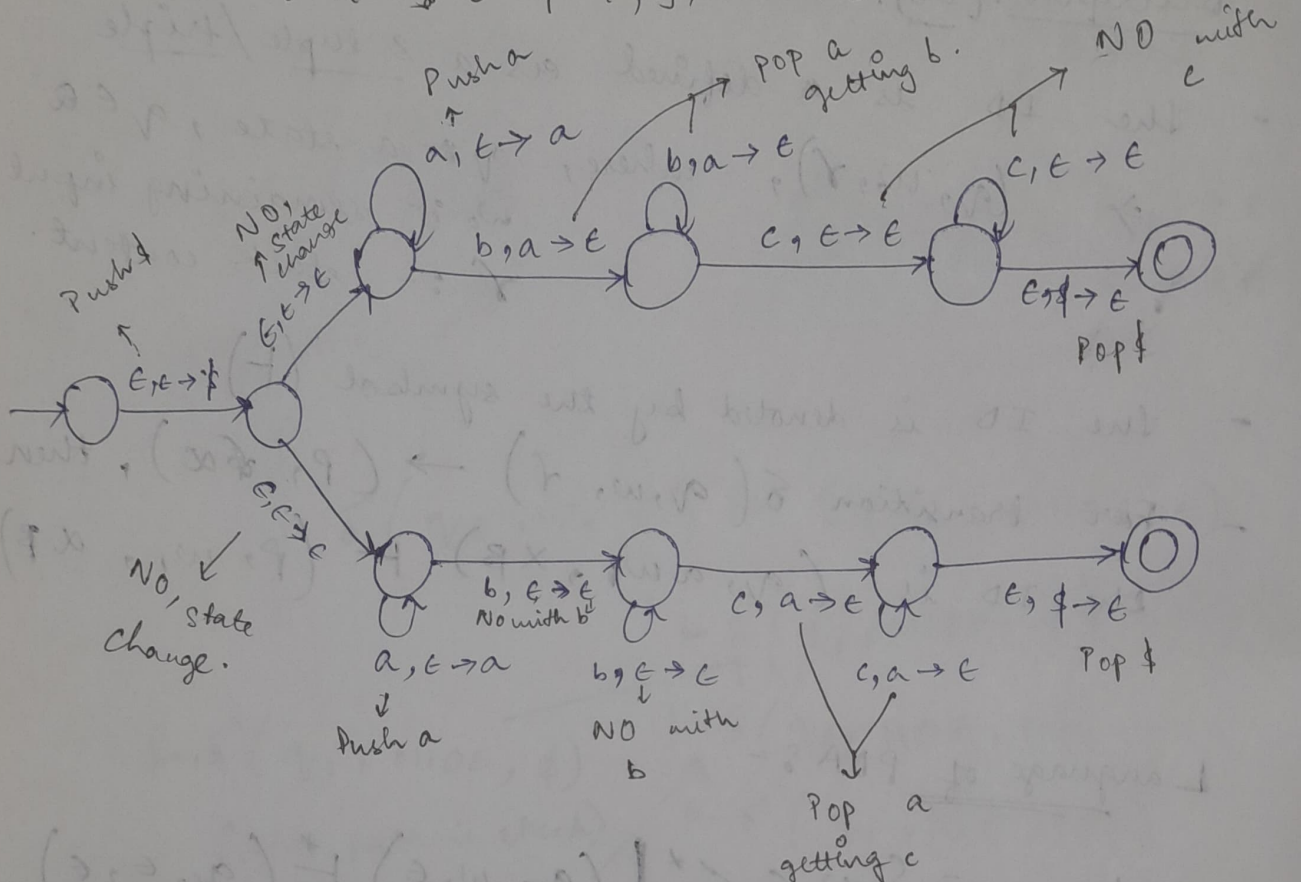
Q// Construct a PDA for the following Language:-

$$L = \{ a^i b^j c^K \mid i = j \text{ or } i = K \}$$

$$\forall i, j, K \geq 1$$

$$L_1 = \{ a^i b^i c^K \mid i, j, K \geq 1 \}.$$

$$L_2 = \{ a^i b^j c^i \mid i, j, K \geq 1 \}.$$



18/12/23

Equivalence of PDA & CFG:-

type-1: A Language is context free if some push down automaton recognizes it.

type-2:- If a PDA recognizes some language, then the language is a context free language.

Theorem -1 :- CFG to PDA :-

- Let a CFG $G(V, \Sigma_G, R, S)$ generates a language $L(G)$.
- we can construct a PDA $P(Q, \Sigma, \Gamma, \delta, q_0, F)$ that recognizes the same language $L(G)$:-

where, $Q = \{q_{start}, q_{loop}, q_{accept}\} \cup \{E\}$

where, E :- is the set of states for short hand implementation.

$\Sigma = \Sigma_G \cup \{\epsilon\}$

$\Gamma = (V \cup \Sigma_G) \cup \{\$\}$

δ = is the transition function defined as :-

1) $\delta(q_{start}, \epsilon, \epsilon) = (q_{loop}, \$S)$

2) For any rule $A \to w$ in R, where, $A \in V$, and $w \in (V \cup \Sigma_G)^*$

$\delta(q_{loop}, \epsilon, A) = (q_{loop}, w)$

3) For any terminal $a \in \Sigma_G$

$\delta(q_{loop}, a, a) = (q_{loop}, \epsilon)$

4) $\delta(q_{loop}, \epsilon, \$) = (q_{accept}, \epsilon)$

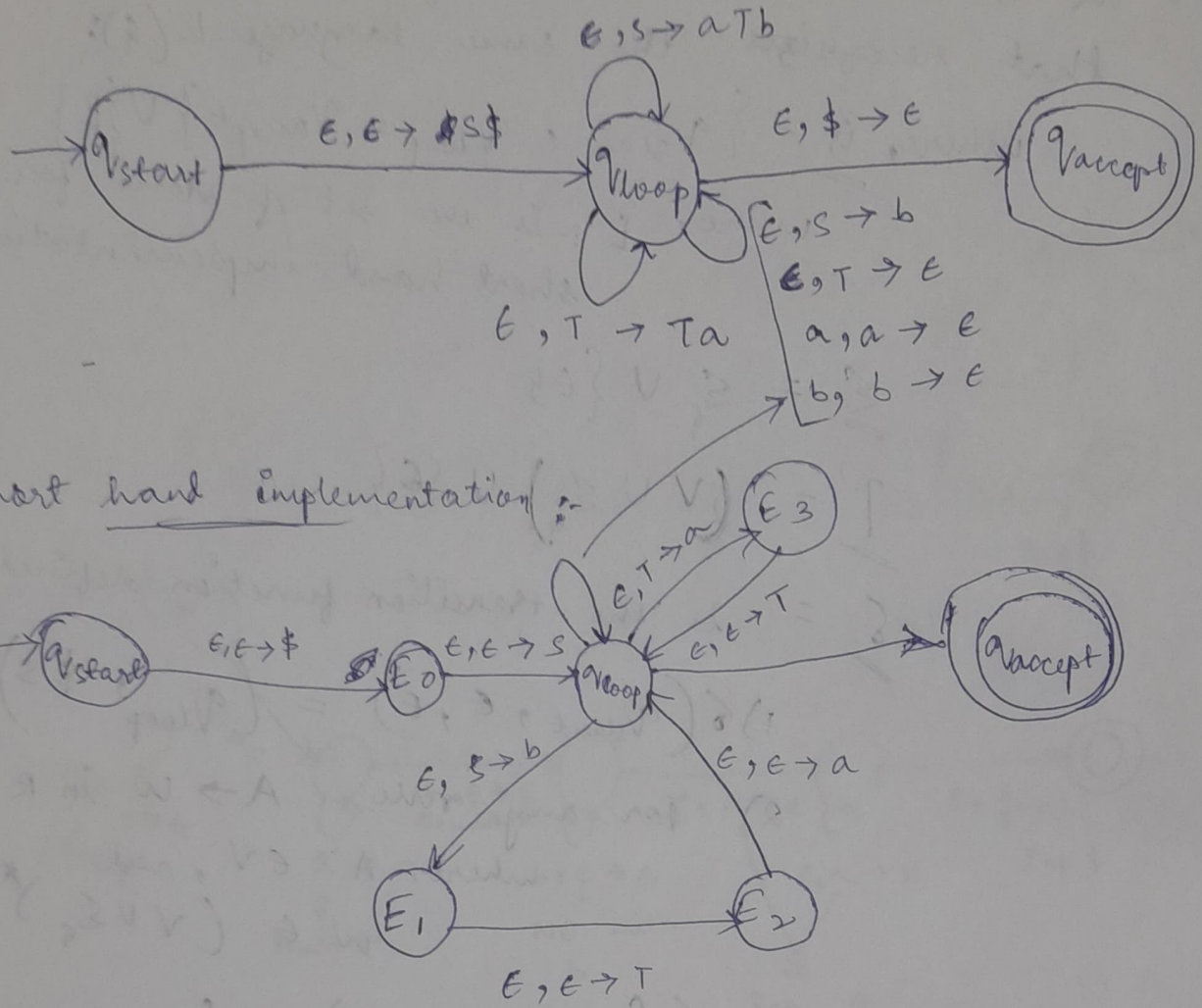
q_0 = initial state :- q_{start}.

F = final state :- $q_{accept} \in F$.

Q/ Construct a PDA for the following grammar.

$S \rightarrow aTb \mid b$

$T \rightarrow Ta \mid \epsilon$



Short hand implementation :-

Q/ Design a PDA for the grammar :-

(a) $S \rightarrow OBB$

$B \rightarrow OS \mid 1S \mid 0$

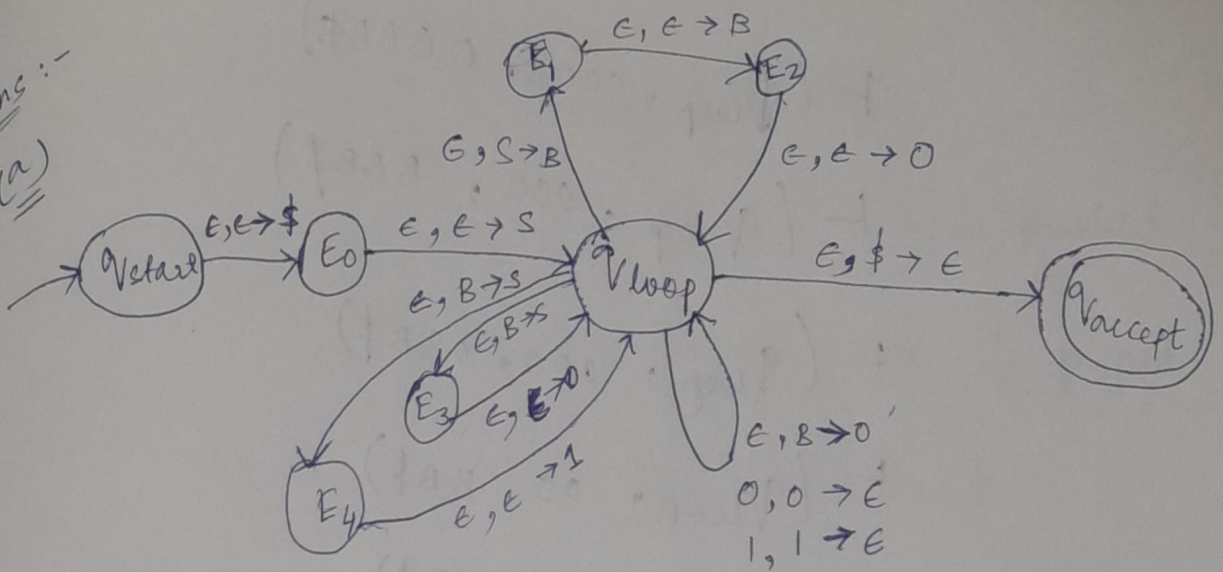
, Process the string 01000.

(b) $S \rightarrow aSb \mid Ab \mid aA \mid a$

$A \rightarrow c \mid \epsilon$

ans :-

(a)



$w = 010000$

$(q_{start}, w, \epsilon) \vdash (q_{start}, 010000, \epsilon)$

$\vdash (E_0, 010000, \$)$

$\vdash (q_{loop}, 010000, S\$)$

$\vdash (E_1, 010000, B\$)$

$\vdash (E_2, 010000, BB\$)$

$\vdash (q_{loop}, 010000, 0BB\$)$

$\vdash (q_{loop}, 10000, BB\$)$

$\vdash (E_4, 10000, SB\$)$

$\vdash (q_{loop}, 10000, 1SB\$)$

$\vdash (q_{loop}, 0000, SB\$)$

$\vdash (E_1, 0000, BB\$)$

$\vdash (E_2, 0000, BBB\$)$

$\vdash (q_{loop}, 0000, 0BBB\$)$

$\vdash (q_{loop}, 000, BBB\$)$

$\vdash (q_{loop}, 000, 0BB\$)$

$\vdash (q_{loop}, 00, BB\$)$

$\vdash (q_{loop}, 00, 0B\$)$

$\vdash (q_{loop}, 0, B\$)$

$\vdash (q_{loop}, 0, 0\$)$

$\vdash (q_{loop}, \epsilon, \$)$

~~$\vdash (q_{loop}$~~

$\vdash (q_{accept}, \epsilon, \epsilon)$

PDA to CFG :-

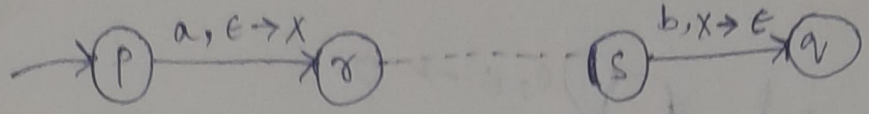
For each pair of state P and Q in the PDA P, the grammar will have a variable $A_{pq} \mid B_q \in Q$.

Simplifying the PDA such that :-

i) It contains only one final state.

ii) The stack should be empty before accepting.

iii) Every transition on a state is either a PUSH or a POP oper^n but not the both.

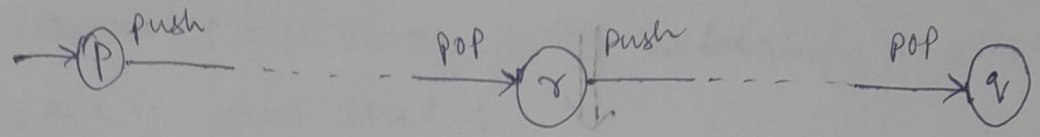
(i) 

$\delta(p, a, \epsilon) = (r, X)$ X is pushed

$\delta(s, b, X) = (q, \epsilon)$ X is poped.

Then add a rule $A_{pq} \longrightarrow a A_{rs} b$

(ii) $A_{pp} \rightarrow \epsilon$

(iii) 

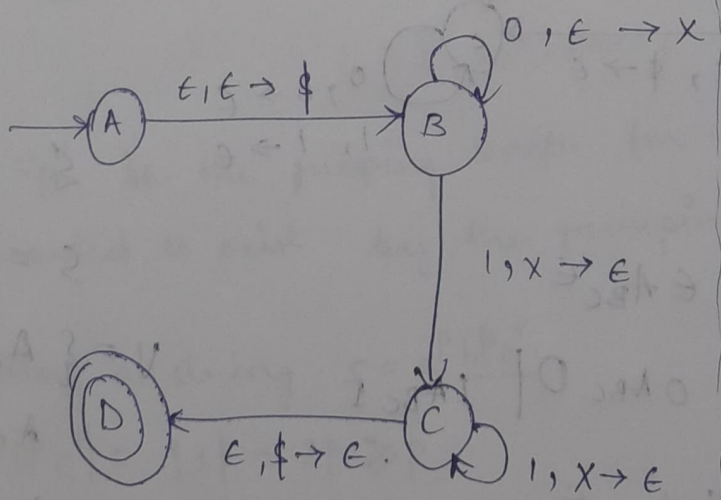
Then add a rule $A_{pq} \longrightarrow A_{pr} A_{rq}$.

Q Construct a CFG for the PDA represented by the language :-

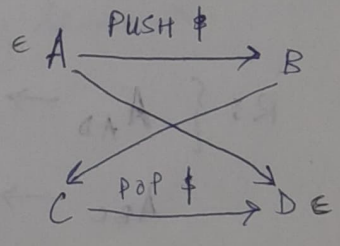
(i) $L = \{ w w^R \mid w \in \{0,1\}^* \}$.

(ii) $N = \{ 0^n 1^n \mid n \geq 1 \}$.

Sol :-

(ii) 

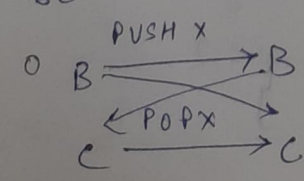
$A_{AD} \longrightarrow \epsilon A_{BC} \epsilon$



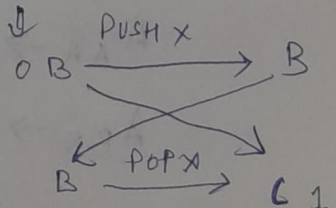
$A_{ABC} \longrightarrow 0 A_{BC} 1$

$A_{AA} \rightarrow \epsilon$ $A_{CC} \rightarrow \epsilon$

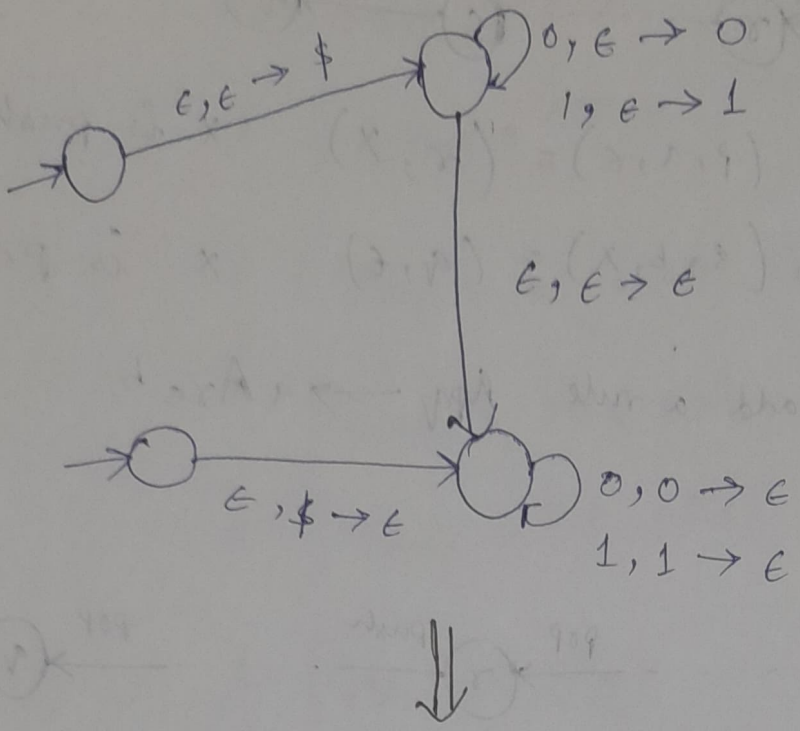
$A_{BB} \rightarrow \epsilon$ $A_{DD} \rightarrow \epsilon$



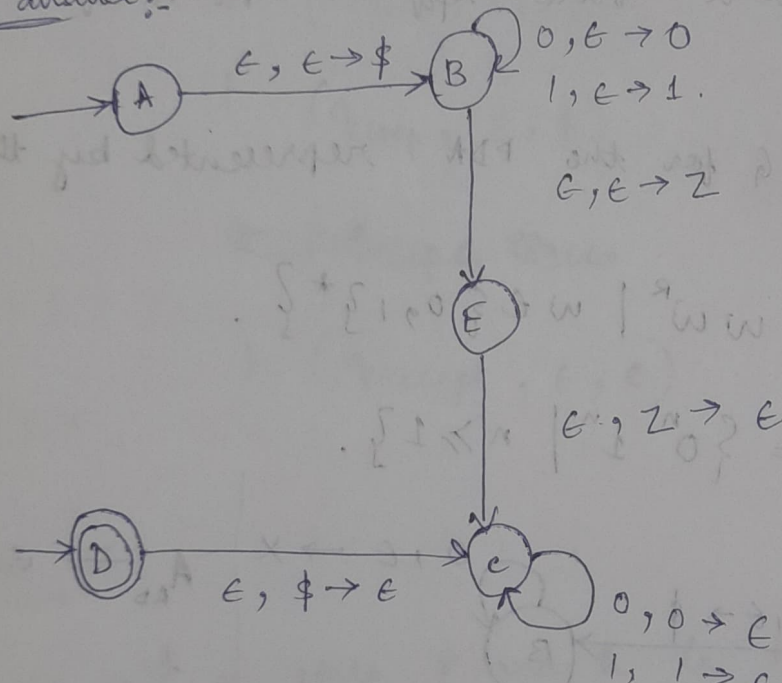
$A_{BC} \longrightarrow 0 A_{BB} 1$.



(i)



Final answer :-



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

$S = A_{AD}$

$V = \{ A_{AA}, A_{BB}, A_{CC}, A_{DD}, A_{EE}, A_{AD}, A_{BC} \}$.

R: $\{ A_{AD} \rightarrow \epsilon A_{BC} \epsilon$

$A_{BC} \rightarrow 0 A_{BC} 0 \mid 1 A_{BC} 1$

$A_{BC} \rightarrow \epsilon A_{EE} \epsilon$

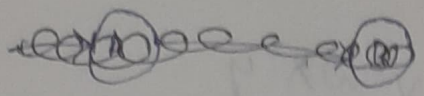
$A_{AA} \rightarrow \epsilon$

$A_{BB} \rightarrow \epsilon$

$A_{CC} \rightarrow \epsilon$

$A_{DD} \rightarrow \epsilon$

$A_{EE} \rightarrow \epsilon$ $\}$



Pumping Lemma for CFL :-

If a language L is contact free then there exists some integer $p \geq 1$ called pumping length such that every string 's' in L that has a length of p or more symbols, i.e., $|s| \geq p$ can be written as $s = uvwxy$ with substrings u, v, w, x, and y such that :

(i) $|vx| \geq 1$

(ii) $|vwx| \leq p$

(iii) $uv^i wx^i y \in L$ for all $i \geq 0$

Eg :- Prove that the language $L = \{a^n b^n c^n \mid n \geq 1\}$ is not CFL.

Ans :- Let's assume that $L = \{a^n b^n c^n \mid n \geq 1\}$ is a CFL.

$L = \{abc, aabbcc, \cdots\}$

- Let 'p' be the pumping length for the language 'L' guaranteed to exist by the pumping lemma.

- considering a string $s = a^p b^p c^p$

$|a^p b^p c^p| = |s| = 3p \geq p$

- The string s can be divided into 5 parts :-

$s = uvwxy$ where → take one element $\forall\ m, n \geq 1$

$u = a^p \quad v = b^m \quad w = b^{p-(m+n)} \quad x = b^n \quad y = c^p$

. Here $|vx| \geq 1$ and $|vwx| \leq p$

for $i=2$ $uv^iwx^iy = uv^2wx^2y$

$= a^p b^{p+m} b^{p-m-n} b^{2n} c^p$

$= a^p b^{p+m+n} c^p \notin L$

∵ No. of b's is more than no. of a's & c's, Hence this is a contradiction.

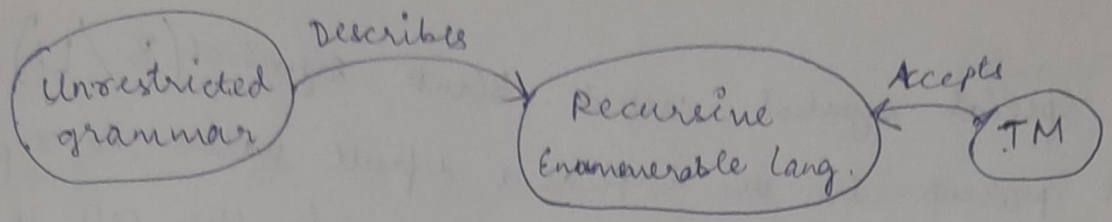
∴ $L = \{a^n b^n c^n / n \geq 1\}$ is not a CFL.

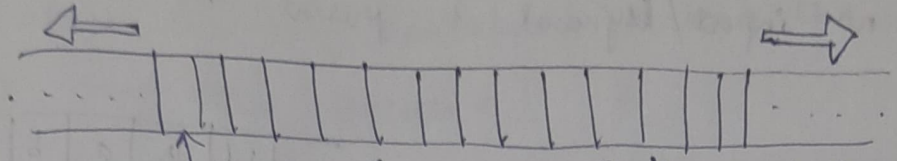
Q Prove that $L = \{ww / w \in \{0,1\}^*\}$ is not a CFL.

Q Prove that $L = \{a^p / p \text{ is prime}\}$ is not a CFL.

11/01/24

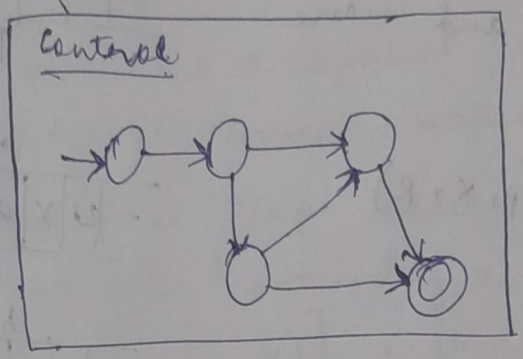
Turing Machine.

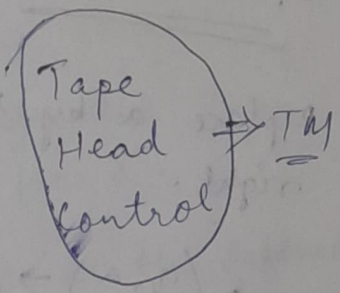




read & write Head

moves in both the direc^ns





→ At each step :-
- reads a symbol
- writes a symbol
- moves left or right or doesn't move.

Formal Def^n of Turing Machine :-

A Turing Machine 'T' is a 7-tuple $(Q, \Sigma, T, \delta, q_0, q_{accept}, q_{reject})$

where :- Q → Finite set of states

Σ → finite set of input symbols/alphabets that does not contain a Blank symbol.

T → A finite set of Tape symbols / Tape alphabets

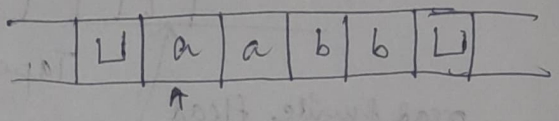
δ → Transition func^n given as $Q \times T \times \{L, R\}$.
(↑ direc^n of movement.)

$q_0 \in Q$ → initial state.

$q_{accept} \in Q \rightarrow$ acceptance state.

$q_{reject} \in Q \rightarrow$ rejectance state

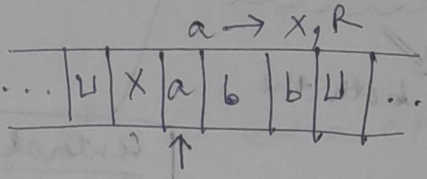
⇒ A special symbol called Blank symbol $(B / \sqcup) \in T$ is present in the tape as a default value to represent that no input/Esymbol is present in the cell of the Tape.



Operations on TM :-

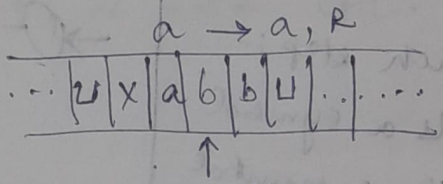
Replace a by X and move right.

$\delta(q, a) \rightarrow (p, X, R)$



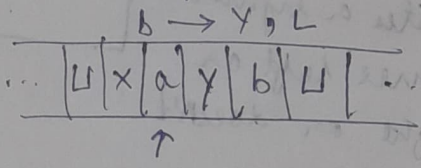
Move right.

$\delta(q, a) \rightarrow (p, a, R)$



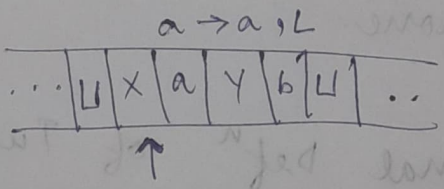
Replace b with Y and move left

$\delta(q, b) \rightarrow (p, Y, L)$



Move Left

$\delta(q, a) \rightarrow (p, a, L)$



05/01/2024

Strings classes of TM :-

Every TM over Σ divides the input string into 3 classes.

→ Accept → Loop → Reject

① Accept (TM) :- It is the set of all strings $w \in \Sigma^*$ if the tape initially contains w and the TM is then run and ends in a ~~halt~~ halt state.

(q accept)

② Loop (TM) :- It is the set of all strings $w \in \Sigma^*$. If the tape initially contains w and the TM runs and then loops forever.

③ Reject (TM) :- It is the set of all strings $w \in \Sigma^*$ if any of the following three cases arise :-

case-1 : A state and a symbol under the Tape head does not have a value for δ :

case-2 : when the Tape head is at the leftmost or rightmost cell, containing a symbol 'x' at a state 'q' $[\delta(q,x)]$, then $\delta(q,x)$ suggest a left or right move respectively. Then the machine rejects as there is no cell in the left and right in the respective cases.

case-3 : In case of an infinite loop of a TM, the string w gets rejected because of the above 2 cases. The TM graces or terminates unsuccessfully.

Language accepted by TM :-

A turing machine 'M' accepts the set of strings w, belongs to Σ^* $[w \in \Sigma^*]$, then the language of the TM :⇒ $L(M) = \{ w / q_0 w \vdash^* x_1 q_A x_2 \}$.

There are 3 types of TM related languages:-

(i) Turing Acceptable language:- A Language 'L' over Σ is said to be Turing Acceptable language if there is a TM $M \ni L = L(M)$.

(ii) Turing Decidable language:- A language 'L' over Σ is said to be Turing Decidable if both the language and its complement are Turing Acceptable.

(iii) Recursive enumerable langnag:- A language 'L' is said to be recursive enumerable if it is accepted by TM.

(Recursive Enumerable lang.)

NOTE :- A Language is called Turing Recognisable, if some TM recognizes it, i.e., the language is accepted or it is rejected or loops forever.

(Recursive lang)

- A Language is called Decidable if some TM decides it, i.e., the language is accepted or rejected.

[(*) Diff bet recursive & recursive enumerable].

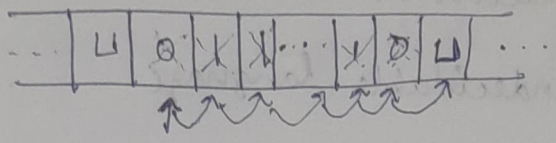
Types of TM :-

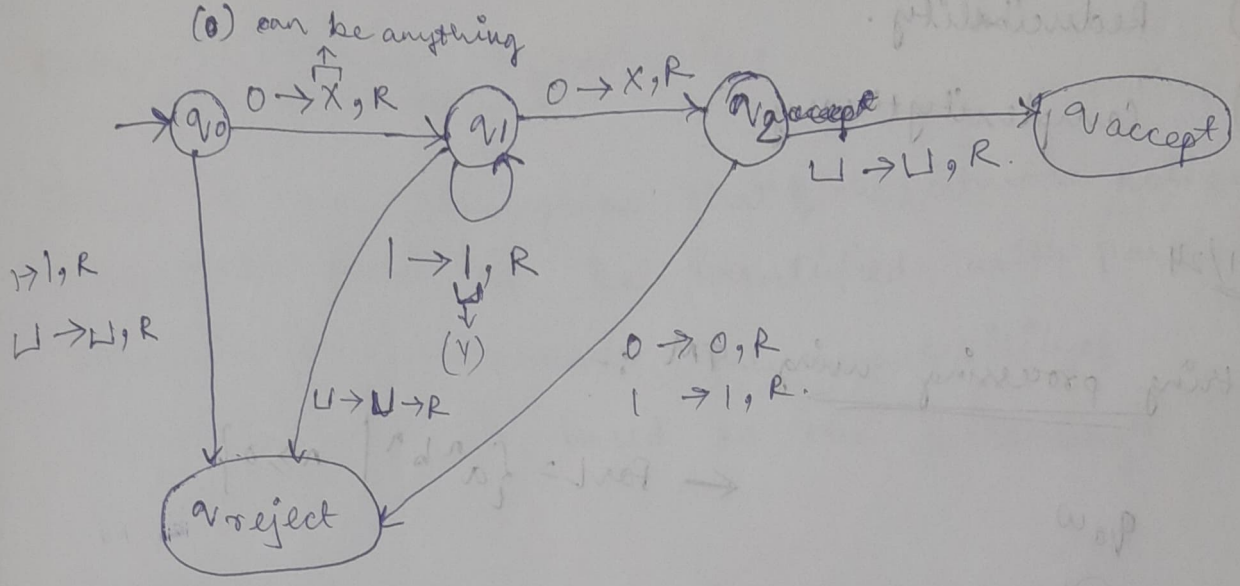
(i) 2 way infinite tape TM
(ii) Multi-tape TM
(iii) Multi-head TM
(iv) Multi-stack TM
(v) Multi-dimensional TM
(vi) Non deterministic TM.
(vii) offline TM
(viii) Universal TM

designing a TM :-

Q// Design a TM for the language represented by the RL 01^*0.

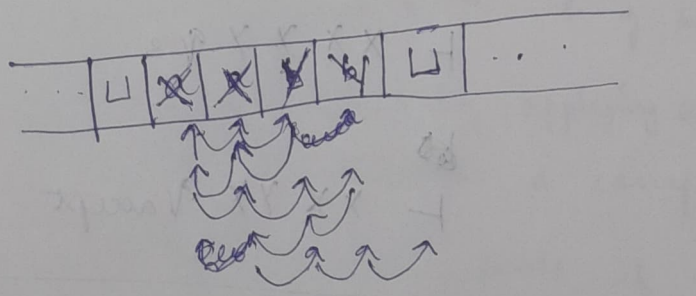
ans :-

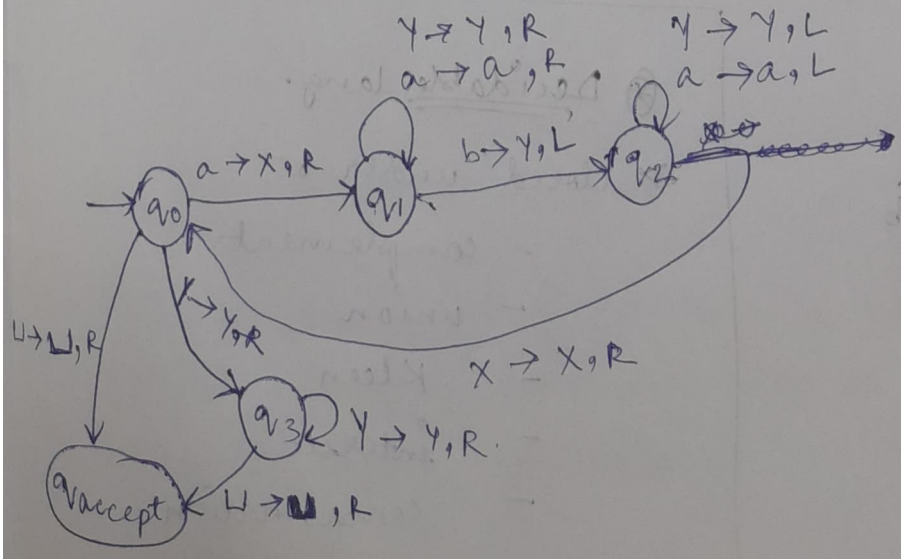




Q// Design a TM for the language $L = \{a^n b^n \mid n \geq 0\}$

ans :-





① String processing using TM.

② Halting problem

③ Church's Turing Hypothesis

④ Recursive vs Recursive enumerable language.

⑤ Decidable vs Undecidable language

⑥ Reducibility.

⑦ Complexity theory.

06/01/24

String processing using TM :-

← For $L = \{a^n b^n \mid n \geq 0\}$.

$q_0 w$

$\vdash q_0 a a b b$

$\vdash X q_1 a b b$

$\vdash X a q_1 b b$

$\vdash X a q_2 Y b$

$\vdash X q_2 a Y b$

$\vdash X q_0 a Y b$

$\vdash X X q_1 Y b$

$\vdash X X Y q_1 b$

$\vdash X X Y q_2 Y$

$\vdash X X q_2 Y Y$

$\vdash X X q_0 Y Y$

$\vdash X X Y q_3 Y$

$\vdash X X Y Y q_3$

$\vdash X X Y Y q_{accept}$

---

⑧ Decidable lang.

⇒ closed under :-

- complement
- union
- Kleen
- intersec^n
- concatenation

(*) The problems that can be computed are called decidable problems.

(*) The problems which cannot be computed are called computationally undecidable problems.

Church's Turing Hypothesis :-

Church's Hypothesis has the

There is an assumption that the intuitive notion of computable func^n can be identified with partial recursive func^{n}s. However the computability of recursive func^n is based on the following:

→ Each elementary func^n is computable.

→ Let f be a computable func^n & 'g' be another func^n which can be obtained by applying an elementary func^n to f then g becomes a computable func^n.

→ Any func^n becomes computable if it is obtained by the above 2 cond^{n}s.

Halting Problem :-

- Halting problem is determining whether a computer program will eventually stop or run forever.
- It is always undecidable.

The Complexity classes:-

→ P :- def^n & eg.

→ NP.

→ NP - complete :- Knapsack, 3 - SAT

→ NP - hard.

⇒ Diff bet^n (i) P & NP

(ii) NP hard & NP complete.

⇒ Eg of complexity classes.