02/09/2020 Automata Language and Computation

Introduction:-

* Computation & Computational devices
  Finite Automata → Computational device
* Automaton is plural of Automata
* Computational devices are also referred as Machines or Recognizers or language Acceptors.
* Chomsky Hierarchy:-

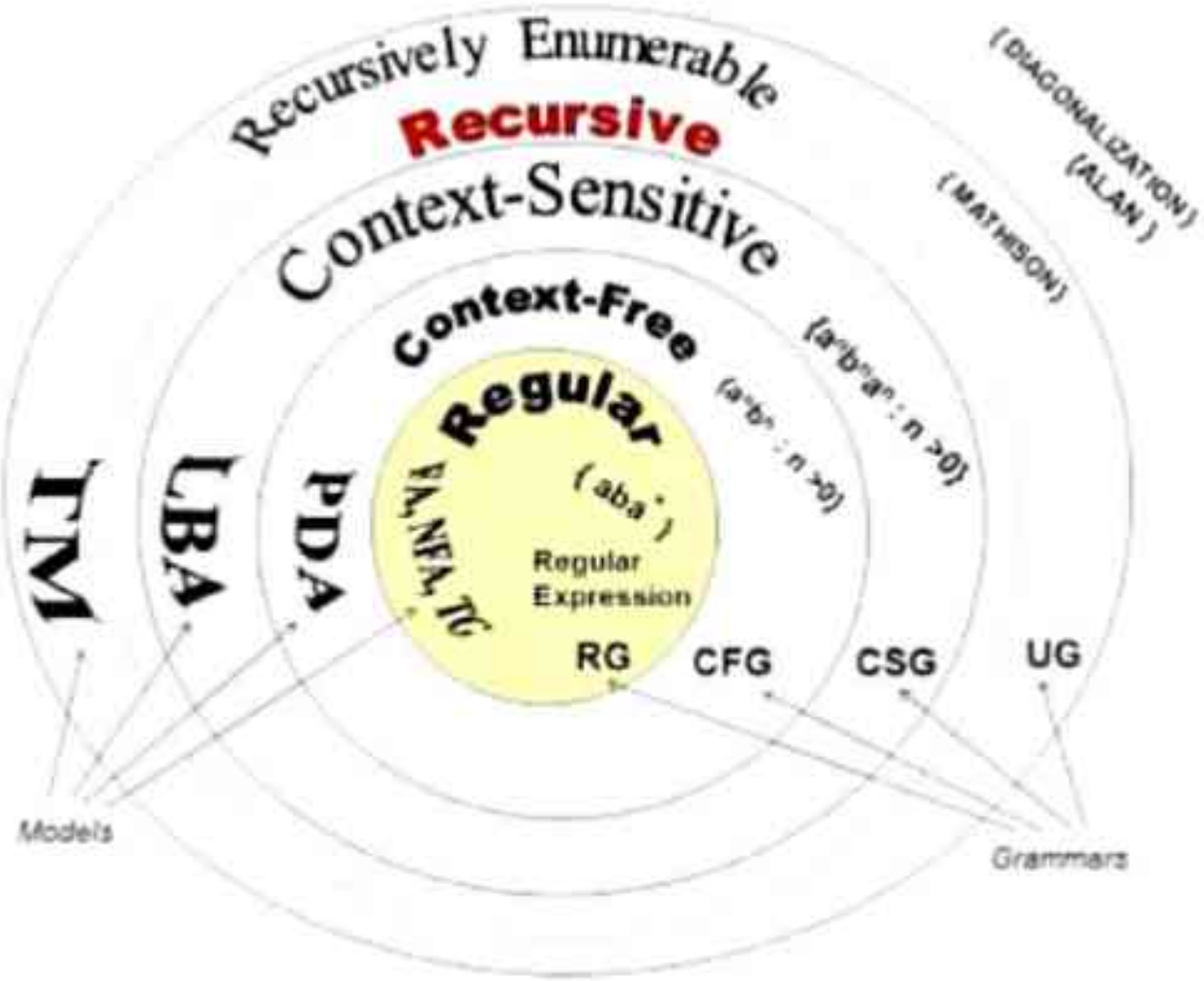
	Grammar	Language	Acceptor
1.	Unrestricted	Recursively enumerable language	Turing machine
2.	Context-Sensitive	context-sensitive language	Linearbounded
3	Context-free	Context-free language	pushdown
4	Regular grammar	Regular language	finite

3/9/2020

-> Finite Automata accepts only regular grammar while push-down automata accepts context-free as well as regular grammar.

-> linear bounded automata accepts Context-Sensitive grammar as well as Context-free grammar and regular grammar.

-> Turing machine accepts recursively enumerable & all other grammar. Hence it is Universal language Acceptor.



4/9/2020 UNIT:- 01

* Introduction:-

→ Symbols :- { a, b, c ..... 0, 1, ... }

→ Alphabets :- $\Sigma = \{a, b\}$

→ Strings :- Sequence of Symbols

→ length of a string is defined as |w|

→ Null String :- String whose length is zero i.e |w| = 0.

→ Consider a string w, then the reverse of a string is denoted as w^R

E.g :- w = aabb

w^R = bbaa

→ Language :- language is defined as a set of all strings (that follow rule) over the alphabet.

consider a string, $w = \{a, b\}$

then language

$L = \{ \epsilon, aa, ab, ba, bb, ... \}$

→ Σ^n → Here n denotes the power of Alphabets.

Eg:- Σ^0 → power is zero → $\{\in\}$ length is zero.

Σ^1 → power is 1 → $\{a, b\}$ i.e length 1

Σ^2 → power is 2 → $\{aa, ab, ba, bb\}$ i.e length is 2

Q) Language for equal no. of a's & Equal no. of b's.

Sol $L = \{ab, aabb, aaabbb \ldots\}$

$L = \{ab, a^2b^2, a^3b^3 \ldots\}$

$L = \{ a^n b^n \mid n >= 0 \}$

* Set form representation of language

→ Used to describe the language in set format

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

$L = \{ a^n b^n \mid n >= 0 \}$

Note:- $a^* b^*$ → Any no. of a's followed by any no. of b's

* Kleene closure:- (Σ^*)

→ Consider $\Sigma = \{a\}$

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \ldots\ldots$$

$\Sigma^0 = \{\epsilon\}$ $\Sigma^2 = \{aa\}$

$\Sigma^1 = \{a\}$ $\Sigma^3 = \{aaa\}$

$\Sigma^* = \{\epsilon, a, aa, aaa \ldots\ldots\}$

$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$$

* Positive closure :- (Σ^+)

→ $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots$

$$\boxed{\Sigma^+ = \Sigma^* - \{\in\}}$$

if $\Sigma = \{a, b\}$

$\Sigma^0 = \{\in\}$

$\Sigma^1 = \{a, b\}$

$\Sigma^2 = \{aa, ab, ba, bb\}$

$$\boxed{\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i}$$

$$\boxed{\Sigma^* = \Sigma^+ \cup \{\in\}}$$

Note :- a^* → any no. of a's including zero

a^+ → any no. of a's excluding zero

i.e one or more no. of a's

7/9/2020

$(a+b)^*$ → any no. of a's & any no. of b's.

* Regular Expression :-

→ The regular Expression for a set $\{\in\}$ is denoted as ϕ

→ If r is regular Expression & s is regular Expression

then $r+s$, r/s, $r.s$, r^* are also regular Expressions.

Q) Write the regular Expression for the language

i) Accepts all combinations of a over the set $\Sigma = \{a\}$

Ans :- $L = \{\in, a, aa, aaa, aaaa \ldots\}$

$\Rightarrow R.E = a^*$

ii) All combinations of a Except null string

Ans :- $L = \{a, aa, aaa, aaaa \ldots\}$

$\Rightarrow R.E = a^+$

3) Containing all strings of a's & b's.
any no. of a's & any no. of b's
$\Sigma = \{a, b\}$

Ans $L = \{ \epsilon, a, b, aa, bb, abb, aabb \ldots \}$

$\Rightarrow R.E \rightarrow (a+b)^*$

4) Accepting all strings which ends with aa over the $\Sigma = \{a, b\}$

Ans $L = \{ aaa, baa, abaa \ldots \}$

$\Rightarrow R.E \rightarrow (a+b)^* aa$

5) Accepting all strings that ends with b and starts with a

Ans $L = \{ ab, aabb, aaabbb \ldots \}$

$\Rightarrow R.E \rightarrow a(a+b)^* b$

6) Containing all strings whose length = 2

Ans $L = \{ ab, aa, bb, ba \}$

$\Rightarrow R.E = (a+b)(a+b)$

7) Accepting all strings whose length is divisible by 2

$|\omega| \bmod 2 = 0$

Ans $R.E = [(a+b)(a+b)]^*$

8) Accepting all strings whose length = 3

Ans $R.E = (a+b)(a+b)(a+b)$

9) Accepting all strings whose length is divisible by 3 $|\omega| \bmod 3 = 0$

Ans $R.E = [(a+b)(a+b)(a+b)]^*$

10) Accepting all strings over $\{a,b\}$ in which total no. of a's is divisible by 3

Ans $L = \{ \omega \in (a,b)^*, n_a(\omega) \bmod 3 = 0 \}$

$R.E = [b^* a b^* a b^* a b^*]^*$

11) Accepting strings over $\{a,b\}$ in which total no. of a's is 3.

Ans $R.E = b^* a b^* a b^* a b^*$

12) Regular expression for even length over {a, b}

Sol $L = \{aa, aaaa, aaaaaa, \ldots\}$

R.E → $(aa)^*$

13) Regular Expression for odd length over {a, b}

Sol $L = \{a, aaa, aaaaa, \ldots\}$

R.E → $a.(aa)^*$

14) Even no. of a's followed by odd no. of b's

Ans R.E → $[(aa)^* b (bb)^*]$

15) Any no. of a's followed any no. b's followed by any no. of c's

Ans R.E = $a^* b^* c^*$

16) Atleast one a followed by b & c

Ans R.E → $a^+ b^+ c^+$

17) Each string is such that the third character from right side is a

Ans R.E = $(a+b)^* a (a+b)(a+b)$

Note :- $a^*b^* \neq a^n b^n$

a^*b^* → any no. of a's followed by any no. of b's

$a^n b^n$ → equal no. of a's followed by equal no. of b's

9/sep/2020

Palindrome :-

$w = w^R$

$L = \{ w \mid w = w^R \}$

18) Each string such that the 10th symbol from right is a

Ans :- R.E = $(a+b)^* a (a+b)^9$

19) Containg atleast 2 b's

Ans R.E = $(a+b)^* b (a+b)^* b (a+b)^*$

20) Exactly 2 b's

Ans R.E = $a^* b a^* b a^*$

Finite Automata — with o/p (Mealy, Moore)
— without o/p (DFA, NFA, FA)

Deterministic Finite Automata :—

DFA :- Deterministic Finite Automata

→ Deterministic refers to Uniqueness of the Computation.

→ The Finite Automata are called deterministic FA if the machine is read an input string one symbol at a time

→ In DFA, there is only one path for specified input from current state to next state

→ DFA does not accept the null move (ε) i.e DFA cannot change state without any input character.

→ It can contain multiple final states

→ DFA is represented using 5 tuples

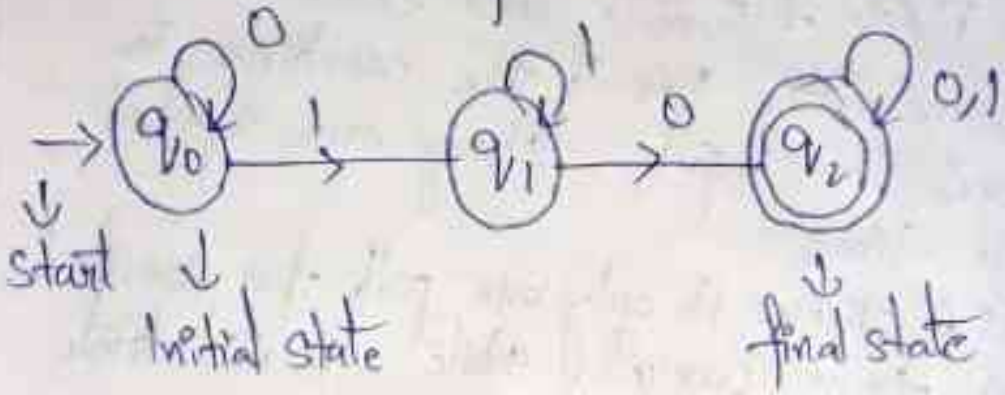
$M = \{Q, \Sigma, \delta, q_0, F\}$ → final state

- Q → finite set of states
- Σ → Input symbols / Alphabets
- δ → Transition function
- q_0 → Initial state

Transition function δ can be defined as

$$\delta : Q \times \Sigma \rightarrow Q$$

→ DFA can be represented by digraphs called state diagram/ Transition diagram.



Initial state is marked with arrow
final state is denoted by double circle/ two concentric circles.

Transition table:- (above dfa)

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

Q/Σ	0	1
→ q_0	q_0	q_1
q_1	q_2	q_1
* q_2	q_2	q_2

* Non-Deterministic Finite Automata

-> The Finite Automata are called NFA when there exist many paths for specific input from current state to next state

-> Every NFA is not DFA, but each NFA can be translated into DFA

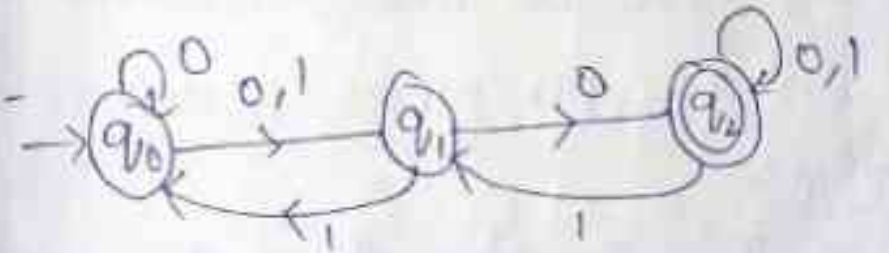
-> NFA is defined in same way as DFA but with following two expectation.

1) It contains multiple next state
2) It contains ϵ transition

-> NFA : $M = (Q, \Sigma, \delta, q_0, F)$

$\delta : Q \times \Sigma \rightarrow 2^Q$

Eg:-



Transition table :-

$Q \mid \Sigma$	0	1
-> q_0	q_0, q_1	q_1
q_1	q_2	q_0
q_2	q_2	q_1, q_2

we can see that on input 0 q_0 is going to q_0 and q_1, similar on input 1 q_2 is going to multiple states i.e q_2 & q_1.

* Applications of finite Automata:-

1) For designing of lexical analysis of a Compiler
2) For Recognizing the pattern using regular Expression
3) For the designing of Combination and Sequential Circuits using mealy and moore machine
4) Used in text Editor
5) For Implementation of Spell checkers.

* Identities related to Regular Expression:-

→ $\phi^* = \epsilon$

→ $\epsilon^* = \epsilon$

→ $RR^* = R^*R$

→ $R^*R^* = R^*$

→ $(R^*)^* = R^*$

→ $(PQ)^*P = P(QP)^*$

→ $R + \phi = \phi + R = R$

→ $R.\epsilon = \epsilon.R = R$

→ $R + R = R$ (Idempotent)

→ $L(M+N) = LM + LN$ (left distributive)
$(M+N)L = ML + NL$ (Right distributive)

* Finite Automata :-

$M = \{ Q, \Sigma, \delta, q_0, F \}$

$Q \rightarrow$ No. of states

$\Sigma \rightarrow$ Input symbols

$\delta \rightarrow$ transition function

$q_0 \rightarrow$ Initial state

$F \rightarrow$ Final state.

$Q \rightarrow$ No. of states

Example of electric switch

Q = No. of states = 2

$Q = \{ ON, OFF \}$

$\Sigma = \{ P \}$ (∵ push operation)

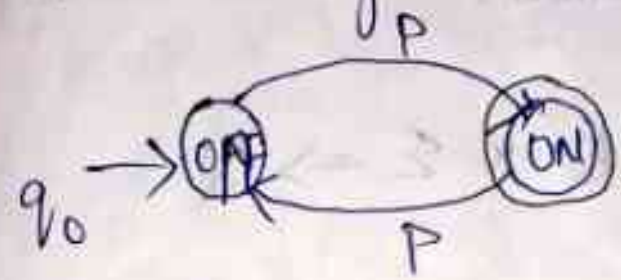
$\delta \rightarrow$ transition function

$Q \times \Sigma \rightarrow$ another state

transition table

	P
OFF	ON
ON	OFF

Transition diagram



q_0 = OFF

F = ON

$F \subseteq Q$

$\delta(OFF, P) = ON$

$\delta(ON, P) = OFF$

Finite Automata

- FA with output
  - Mealy M/C
  - Moore M/C
- F.A without o/p
  - Deterministic FA
  - Non-Deterministic FA
    - with ε
    - without ε
  - FA

DFA → Deterministic Finite Automata
→ on an input it goes to only one state

NDFA → Non-Deterministic finite Automata
→ on an input it goes to more than one state.

Transition function :-

→ DFA

↳ $Q \times \Sigma \Rightarrow Q$

→ NDFA

↳ $Q \times \Sigma \rightarrow 2^{Q}$

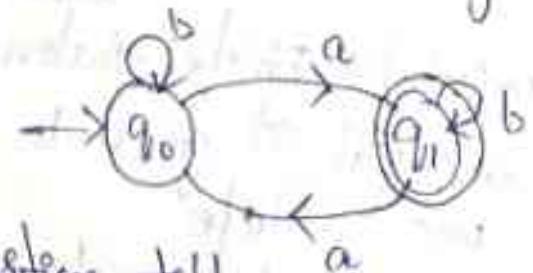
* DFA :-

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

$Q = \{q_0, q_1\}$

$\Sigma = \{a, b\}$

δ → transition diagram



q_1 → final state

q_0 - initial state

transition table

Q/Σ	a	b
q_0	q_1	q_0
q_1	q_0	q_1

$\delta(q_0, a) = q_1$

$\delta(q_1, a) = q_0$

$\delta(q_0, b) = q_0$

$\delta(q_1, b) = q_1$

6) Check whether the string is accepted by above DFA with $w = abaa$

Sol) Given :- $w = abaa$

$\Rightarrow \delta(q_0, abaa)$

$\vdash (q_1, baa)$

$\vdash (q_1, aa)$

$\vdash (q_0, a)$

$\vdash q_1$

After reading a string if we reached final state then the string is Accepted.

$\therefore w = abaa$ is Accepted by above DFA

similarly for $w = abab$

$\delta(q_0, abab)$

$\vdash \delta(q_1, bab)$

$\vdash \delta(q_1, ab)$

$\vdash \delta(q_0, b)$

$\vdash q_0$

$\therefore$ abab is Not Accepted.

Q) Design a DFA for the language accepts the strings such that the length of the string is exactly two over the alphabet

$\Sigma = \{a, b\}$

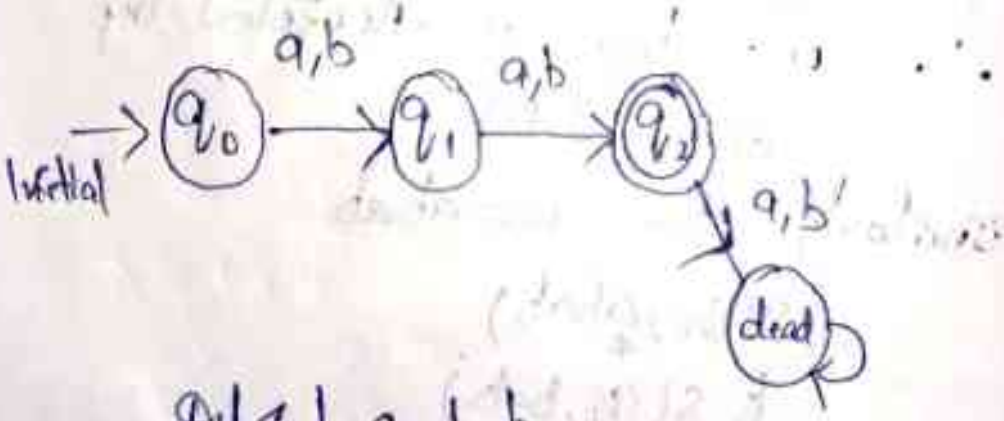
Sol i) $|w| = 2$

$L = \{aa, ab, ba, bb\}$

$q_0 \rightarrow$ length of string with zero length

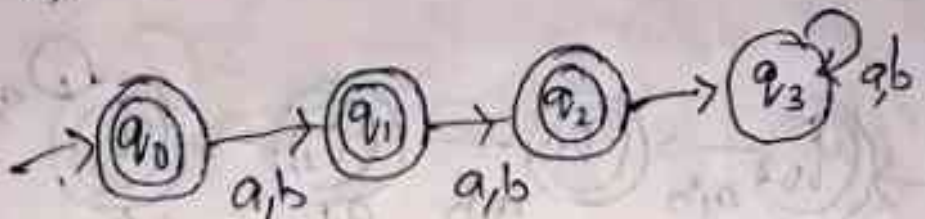
$q_1 \rightarrow$ length 1

$q_2 \rightarrow$ string with length 2

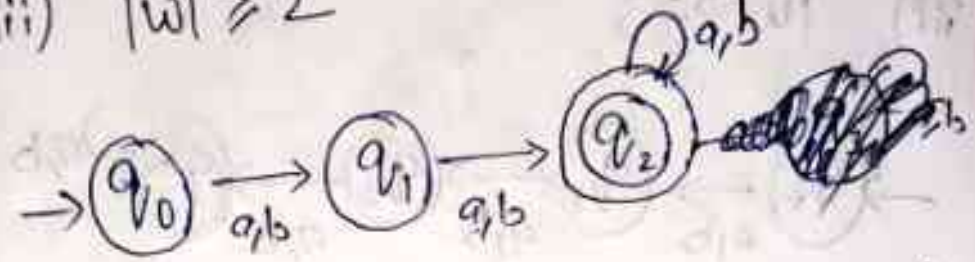


δ / Σ	a	b
q_0	q_1	q_1
q_1	q_2	q_2
q_2	q_3	q_3 ← dead

ii) $|w| \leq 2$



iii) $|w| \geq 2$



Q) Design a DFA for the language accepts the string such that the length of the string $\Sigma = \{a, b\}$

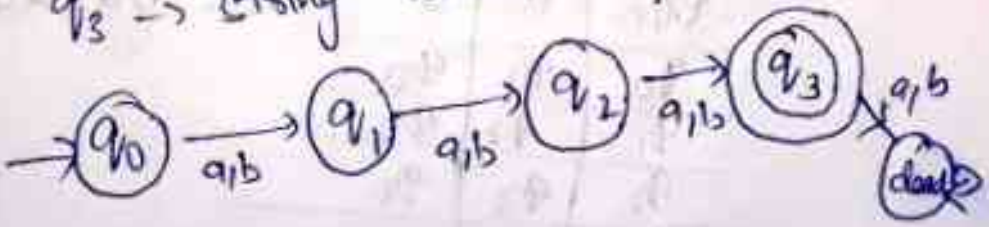
i) $|w| = 3$ ii) $|w| \leq 3$ iii) $|w| \geq 3$

Sol

i) $|w| = 3$

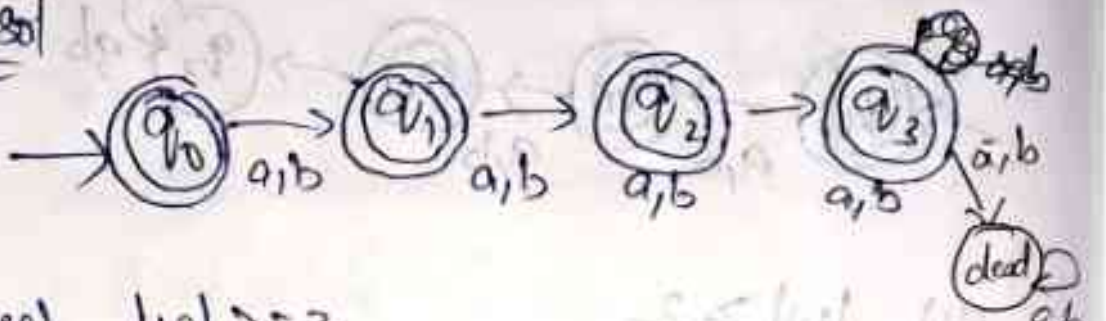
$L = \{aaa, bbb, abb, bba \ldots\}$

q_0 → length of string zero
q_1 → string with length one
q_2 → string with length two
q_3 → string with length three

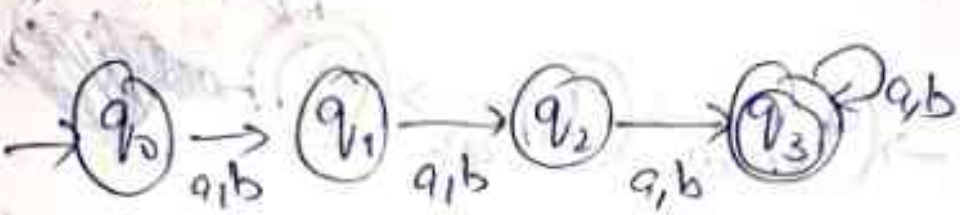


ii) $|w| <= 3$

Sol



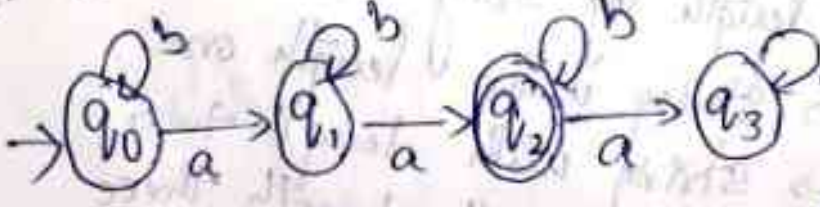
iii) $|w| >= 3$



Q) Design a D.F.A

1) $n_a(w) = 2$ i.e no. of a's = 2
2) $n_a(w) >= 2$ i.e no. of a's >= 2
3) $n_a(w) <= 2$ i.e no. of a's <= 2

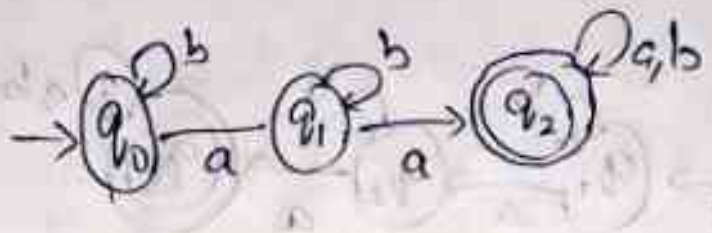
Sol i) $n_a(w) = 2$



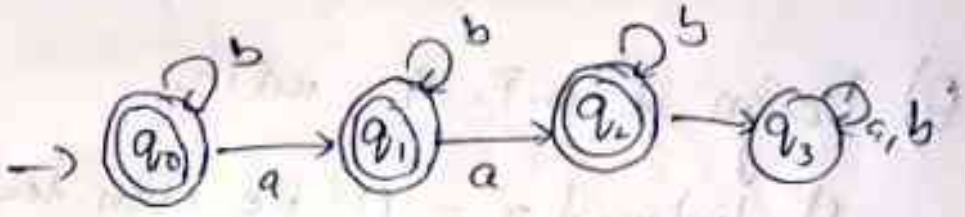
δ / Σ	a	b
q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
q_3	-	-

R.E = $b^* a b^* a b^*$

2) $n_a(w) >= 2$



3) $n_a(w) <= 2$



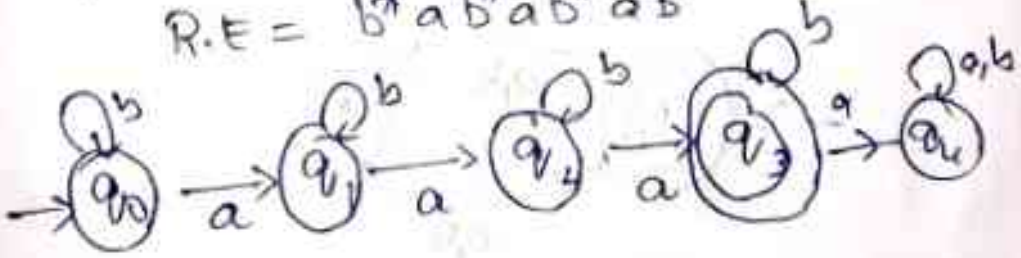
Q) Design a D.F.A

i) $n_a(w) = 3$ ii) $n_a(w) <= 3$

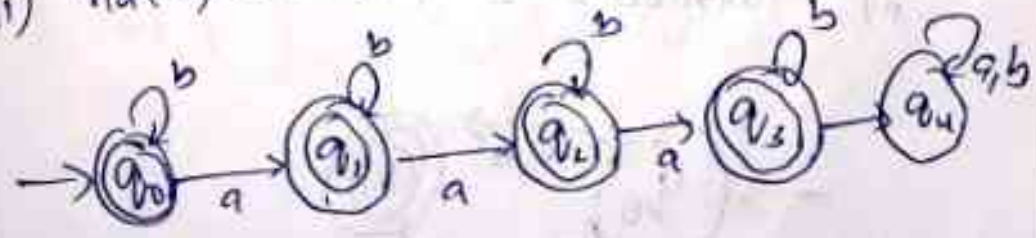
iii) $n_a(w) >= 3$

Sol i) $n_a(w) = 3$

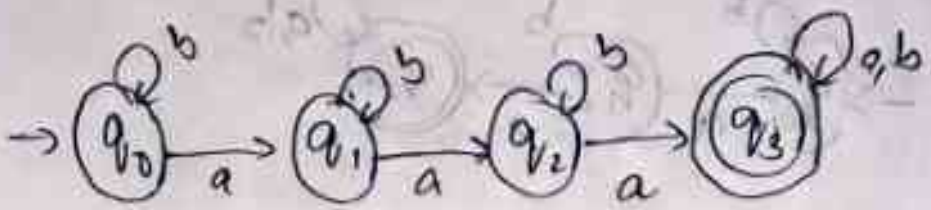
R.E = $b^* a b^* a b^* a b^*$



ii) $n_a(w) <= 3$



iii) $n_a(w) >= 3$



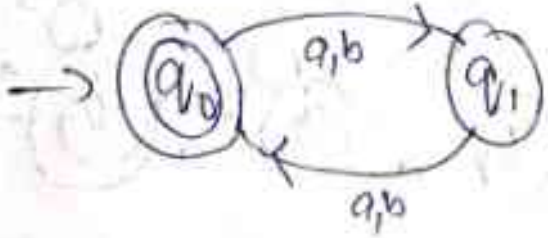
Q) Design a D.F.A, with.

i) $|w| \bmod 2 = 0$ i.e even length string

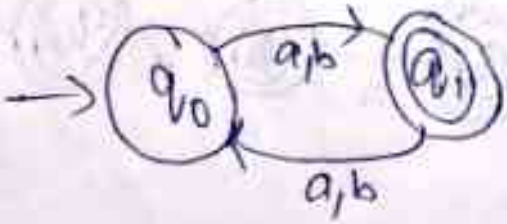
ii) $|w| \bmod 2 = 1$ i.e odd length string

Sol i) $|w| \bmod 2 = 0$

R.E → $((a+b)(a+b))^*$



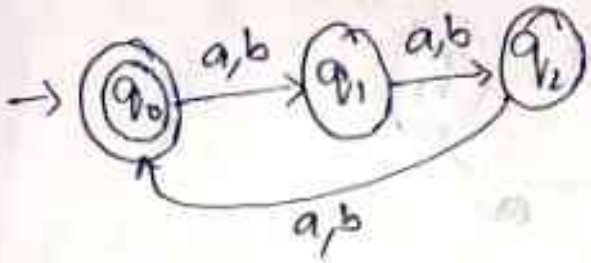
ii) $|w| \bmod 2 = 1$



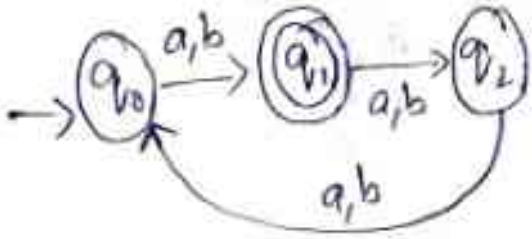
Q) Design a D.F.A

1) $|w| \bmod 3 = 0$
2) $|w| \bmod 3 = 1$
3) $|w| \bmod 3 = 2$

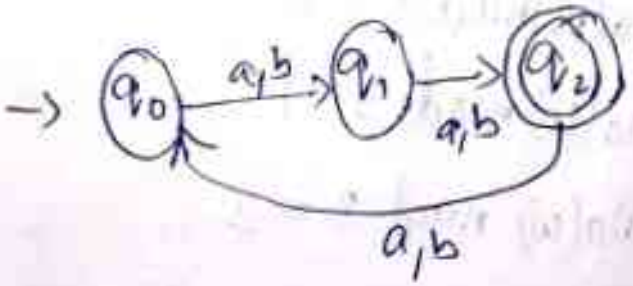
Sol) 1) $|w| \bmod 3 = 0$



2) $|w| \bmod 3 = 1$



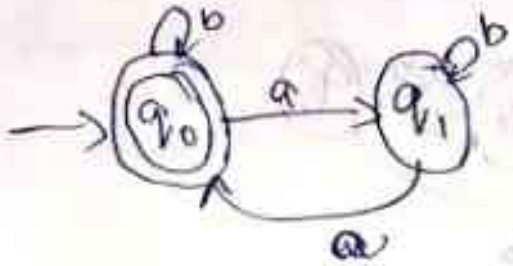
3) $|w| \bmod 3 = 2$



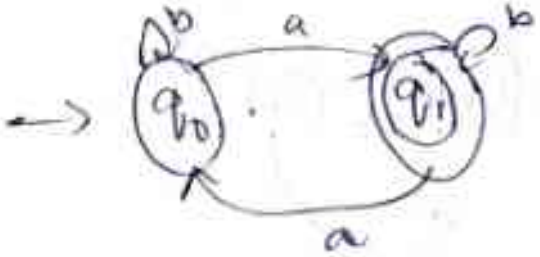
Q) Design a D.F.A

1) $n_a(w) \bmod 2 = 0$
2) $n_a(w) \bmod 2 = 1$

Sol 1) $n_a(w) \bmod 2 = 0$



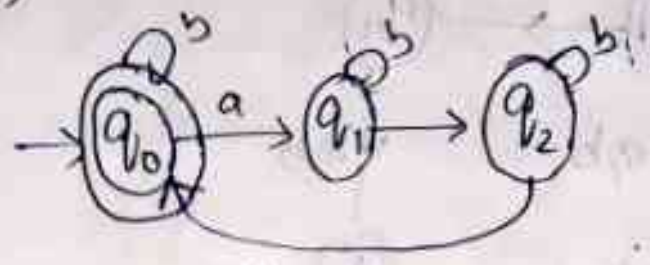
2) $n_a(w) \bmod 2 = 1$



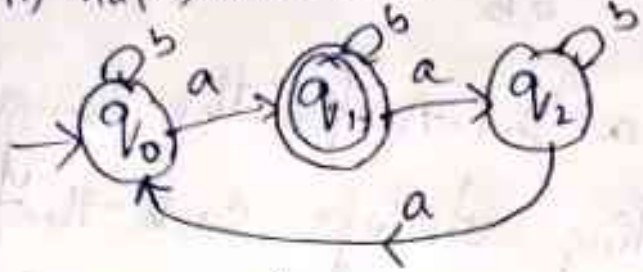
Q) Design a D.F.A

i) $n_a(w) \bmod 3 = 0$
ii) $n_a(w) \bmod 3 = 1$
iii) $n_a(w) \bmod 3 = 2$

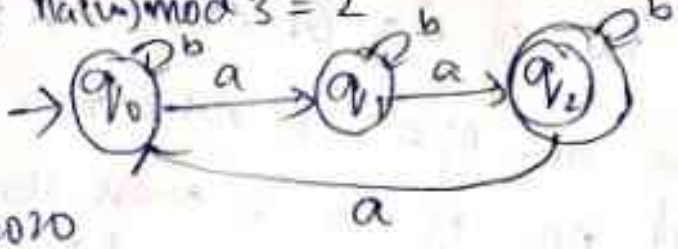
Sol i) $n_a(w) \bmod 3 = 0$



ii) $n_a(w) \bmod 3 = 1$



iii) $n_a(w) \bmod 3 = 2$



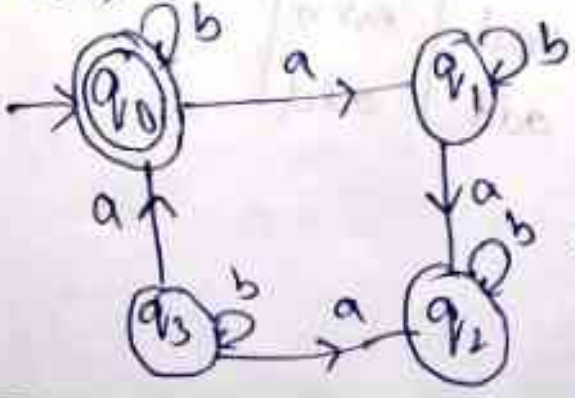
16/sep/2020

Q) Design a DFA for the language containing strings such that

i) $L = \{ w / w \in (a,b)^*, n_a(w) \bmod 4 = 0 \}$

ii) $L = \{ w / w \in (a,b)^*, |w| = 4 \}$

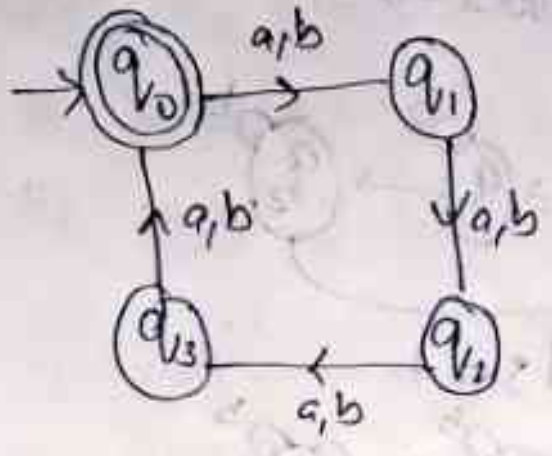
Sol i) $n_a(w) \bmod 4 = 0$



$q_0 \rightarrow$ final state

if $(n_a(w) \bmod 4 == 1)$ then $q_1 \rightarrow$ final state

ii) $|w| = 4$



Q) Design a D.F.A for the language accepting strings such that

i) Even no. of a's & even no. of b's

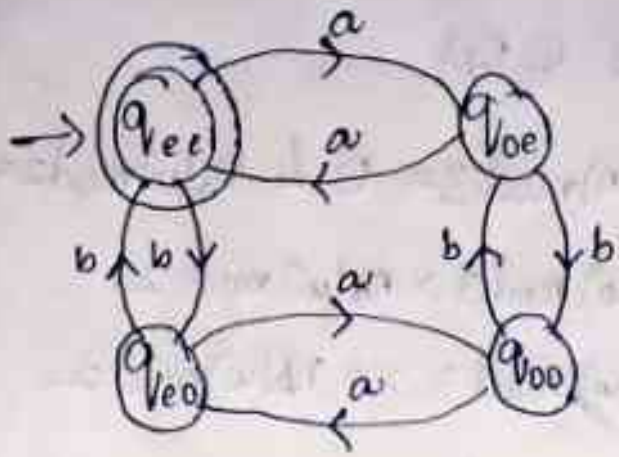
ii) Even no. of a's & odd no. of b's

iii) Odd no. of a's & even no. of b's

iv) odd no. of a's & odd no. of b's

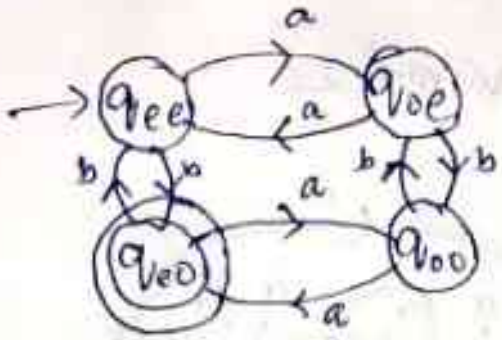
Sol i) $n_a(w) \rightarrow$ even $n_b(w) \rightarrow$ even

	$n_a(a)$	$n_a(b)$
q_{ee} →	even	even
q_{eo}	even	odd
q_{oe}	odd	even
q_{oo}	odd	odd



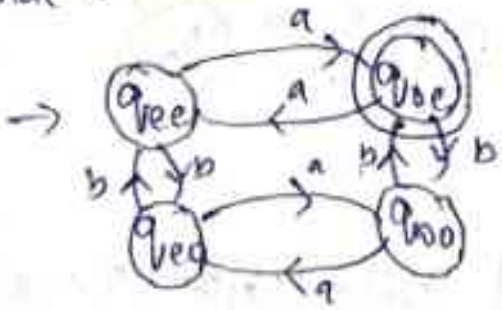
q_{ee} → final state

ii) $n_a(w)$ → even & $n_b(w)$ → odd



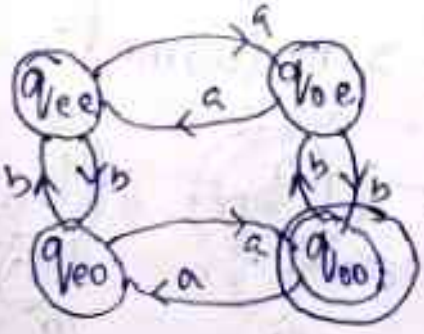
q_{eo} → final state

iii) $n_a(w)$ → odd & $n_b(w)$ → even



q_{oe} → final state

iv) $n_a(w)$ → odd & $n_b(w)$ → odd



q_{oo} → final state

Q) Design a DFA

i) $n_a(w) \bmod 3 = 0$ & $n_b(w) \bmod 2 = 0$

ii) $n_a(w) \bmod 3 > n_b(w) \bmod 2$

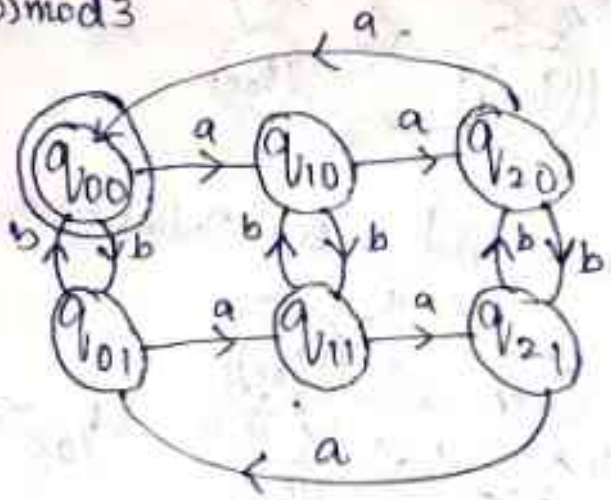
iii) $n_a(w) \bmod 3 >= n_b(w) \bmod 2$

i) $n_a(w) \bmod 3 = 0$ & $n_b(w) \bmod 2 = 0$

q_0 0 → $n_b(w) \bmod 2$

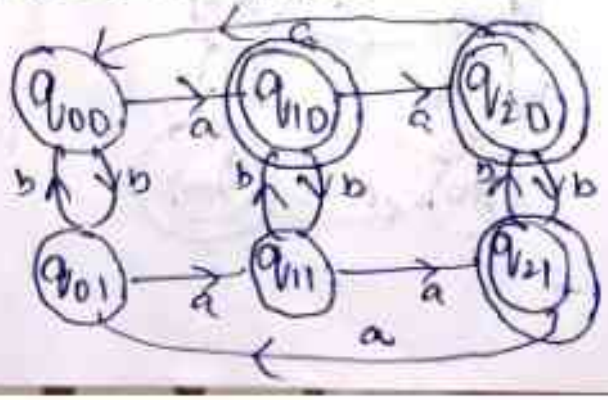
↓

$n_a(w) \bmod 3$



q_{00} → Final state ($\because n_a \bmod 3 = 0$ & $n_b(w) \bmod 2 = 0$)

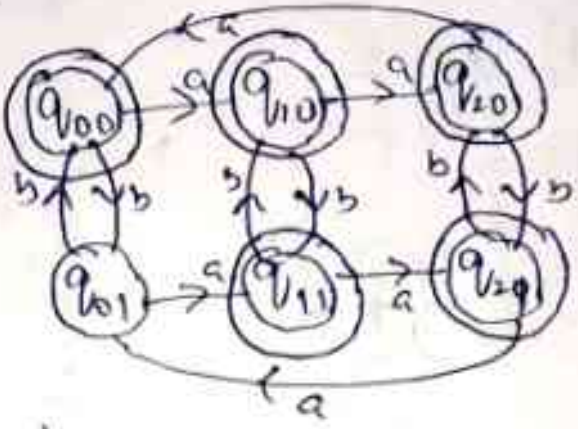
ii) $n_a(w) \bmod 3 > n_b(w) \bmod 2$



Final states are q_{10}, q_{20} & q_{21}

($\because$ $n_a(\omega) \bmod 3 > n_b(\omega) \bmod 2$)

(ii) $n_a(\omega) \bmod 3 >= n_b(\omega) \bmod 2$

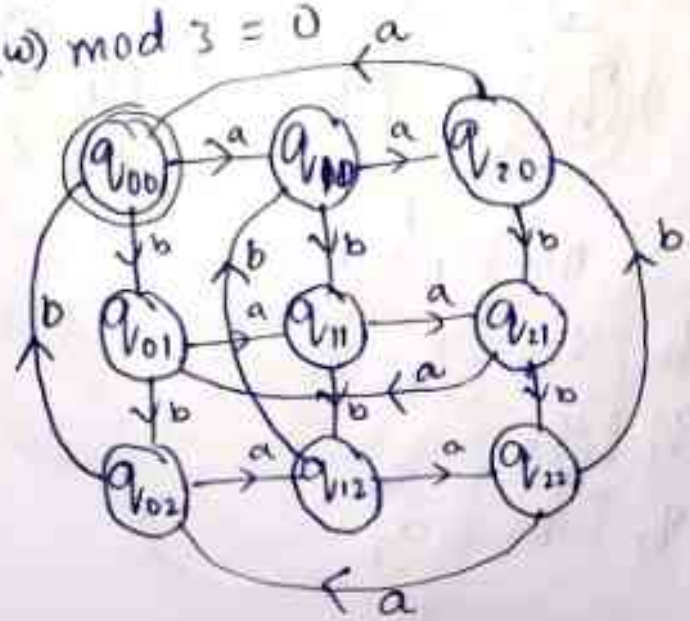


final states → q_{00}, q_{10}, q_{20}, q_{11}, q_{21}

($\because$ $n_a(\omega) \bmod 3 >= n_b(\omega) \bmod 2$)

Q) Design a D.F.A. $n_a(\omega) \bmod 3 = 0$ & $n_b(\omega) \bmod 3 = 0$

Sol $n_a(\omega) \bmod 3 = 0$ & $n_b(\omega) \bmod 3 = 0$



9) Design a DFA for the language L which each binary string w is interpreted as decimal number over $\Sigma = \{0,1\}$

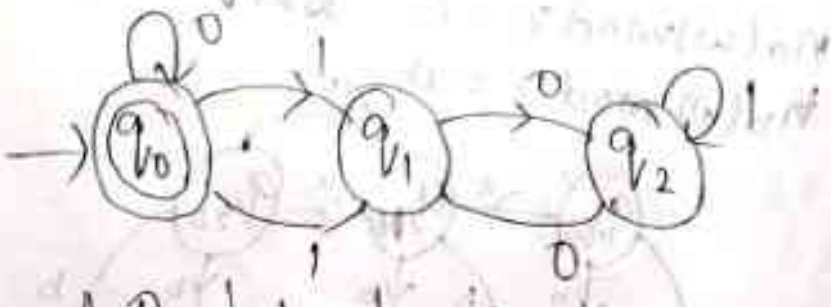
i) Divisible by 2

Sol

	0	1
q_0	q_0	q_1
q_1	q_0	q_1

Decimal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
.	.
.	.
.	.

ii) Divisible by 3

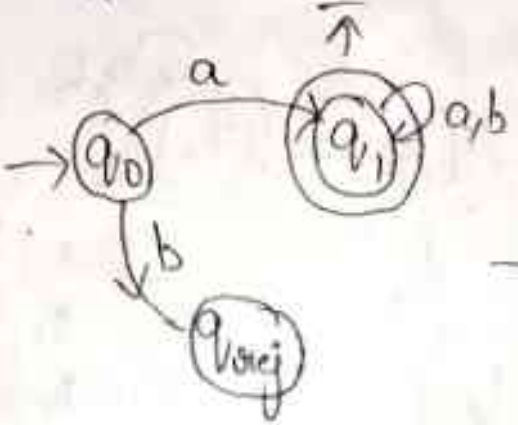


	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

Q) Design a DFA for the language over the Alphabet $\Sigma = \{a, b\}$ in which string

i) Starting with a

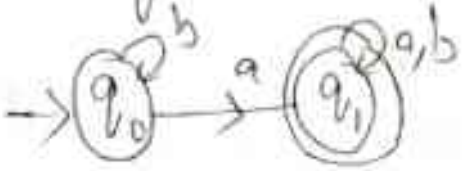
Sol: R.E = $a(a+b)^*$



	a	b
q_0	q_1	q_{rej}
q_1	q_1	q_1

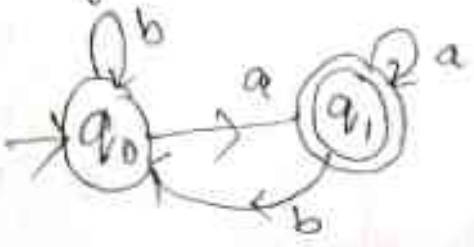
ii) containing a

Sol:



	a	b
q_0	q_1	q_0
q_1	q_1	q_1

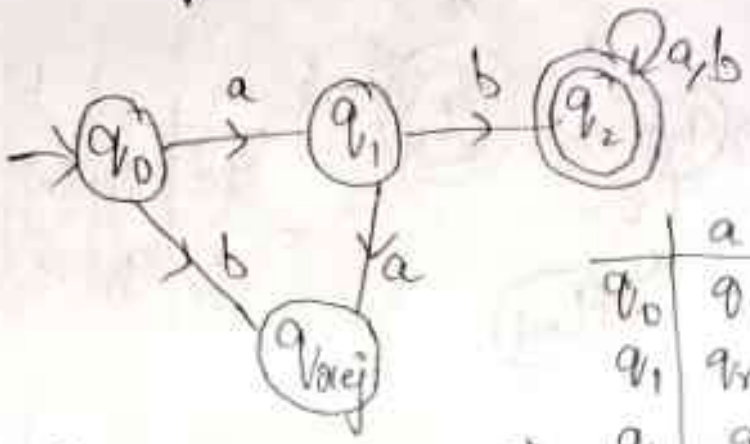
iii) ending with a



R.E = $(a+b)^* a$

Q) Design a DFA for the language containing strings in which each string

i) starting with ab

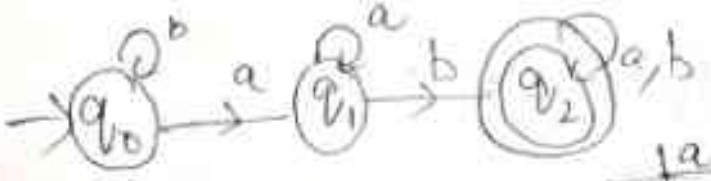


	a	b
q_0	q_1	q_{rej}
q_1	q_{rej}	q_2
q_2	q_2	q_2

R.E → ab(a+b)*

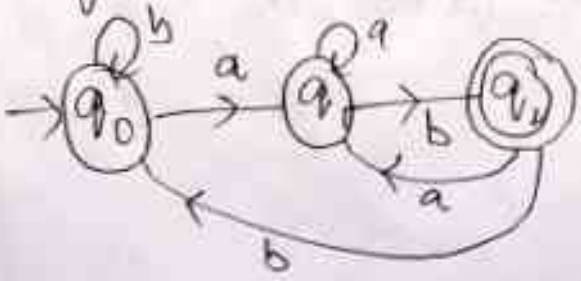
ii) containing ab

R.E → (a+b)*ab(a+b)*



	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_2	q_2

iii) ending with ab

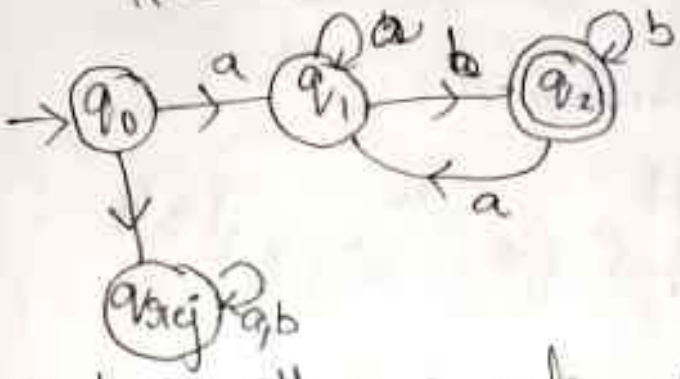


Q) Design a DFA for the language over $\Sigma = \{a, b\}$ in which each string ω

i) begins with a & ends with b

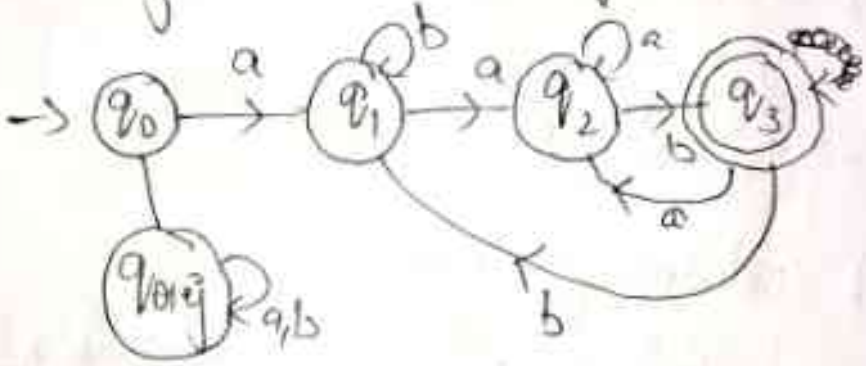
Sol

R.E → $a(a+b)^* b$



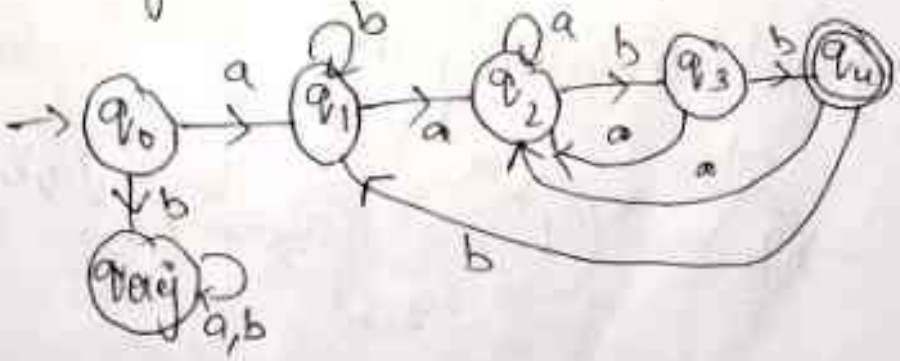
ii) Starting with a & ending with ab

Sol



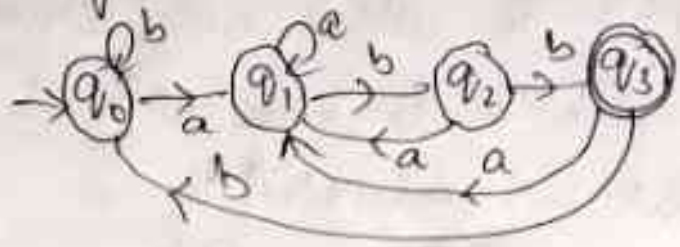
iii) Starting with a and ending with abb

Sol



iv) Ending with abb

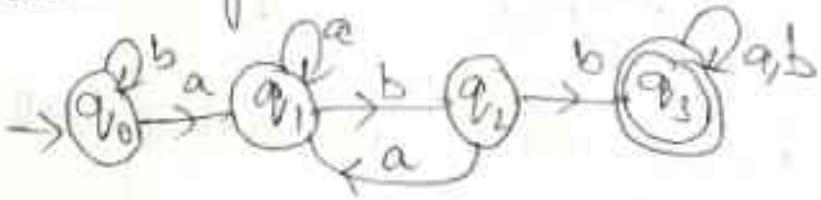
Sol



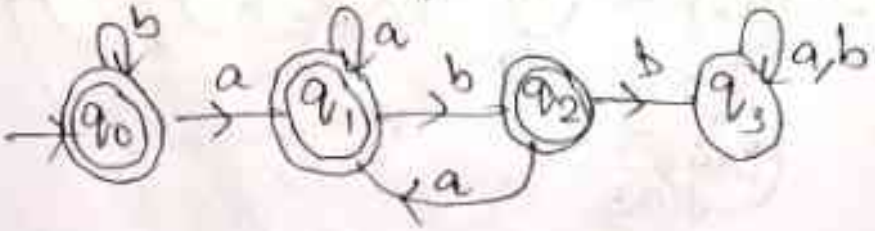
v) not containing ab

vi) containing ab

vii) containing abb



viii) not containing abb



c) Design a DFA that begins & ends with different symbols.

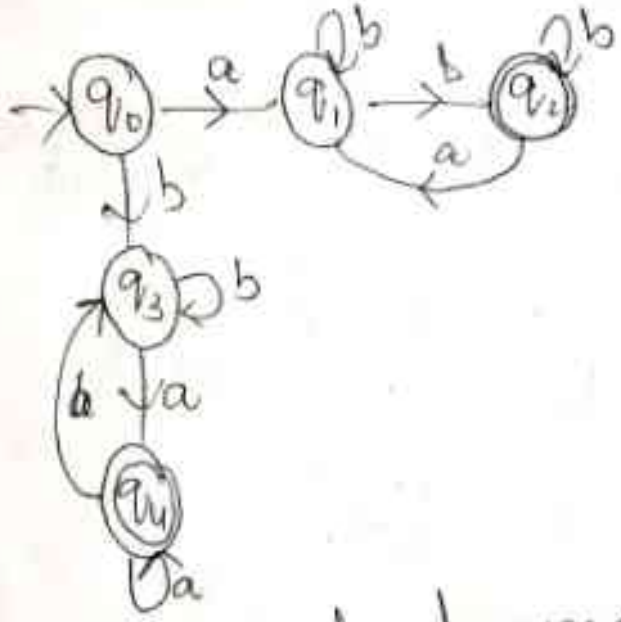
Sol

$L_1 \rightarrow a\underline{(a+b)^*}b$

$L_2 \rightarrow b\underline{(a+b)^*}a$

$R.E \rightarrow L_1 \cup L_2$

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

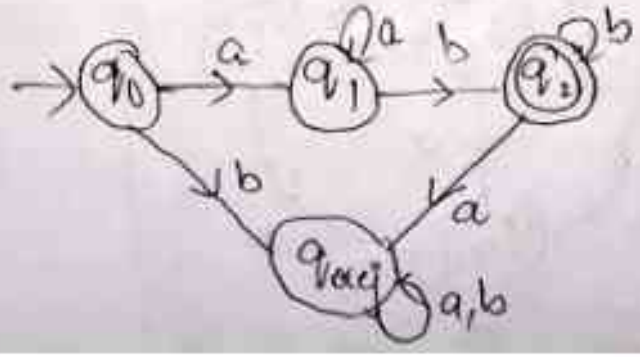


d) Design DFA for language

1) $L = \{a^n b^m \mid n, m >= 1\}$

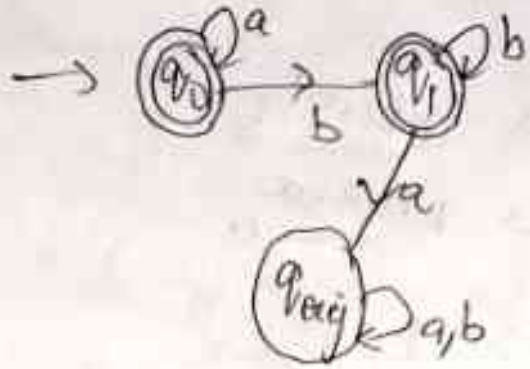
Sol

$L = \{ab, abb, abbb, aab, aaab \ldots\}$



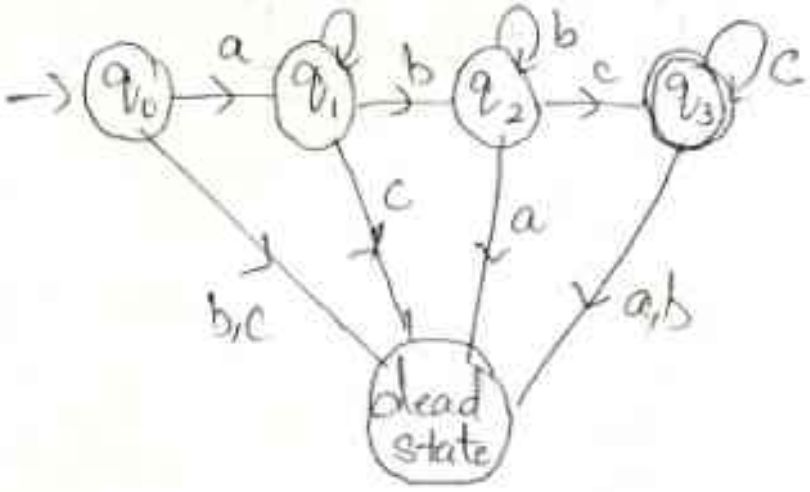
2) $L = \{ a^n b^m \mid n, m >= 0 \}$

Sol



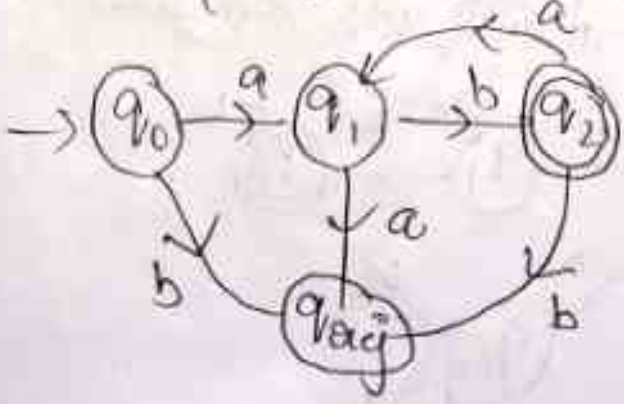
3) $L = \{ a^n b^m c^z \mid n, m, z >= 1 \}$

Sol



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

$L = \{ ab, abab, ababab, abababab, \cdots \}$



28/Sep/2020

NFA:- Non-Deterministic Finite Automata

→ NFA:-

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

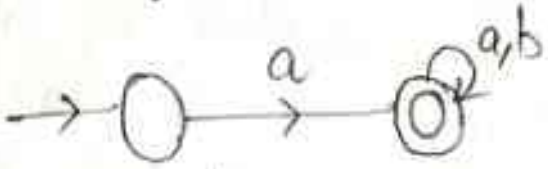
$\delta \rightarrow$ transition function

$Q \times \Sigma = 2^Q$

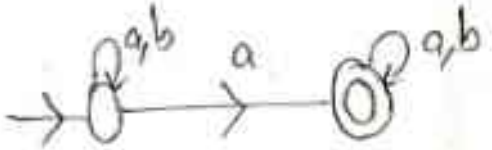
Q1) Design a NFA for the language

i) starting with a

Sol



ii) containing a



Q2) Check whether the string $w = aaaba$ is accepted by below NFA or not

Sol NFA transition table

	a	b
q_0	q_0, q_1	q_0
q_1	q_2	ϕ
q_2	ϕ	q_3
q_3	q_4	ϕ
q_4	ϕ	ϕ

initial state → q_0

q_4 → final state

$w = aaaba$

$\delta(q_0, aaaba)$

$\vdash ((q_0, q_1), aaba)$

$\vdash ((q_0, q_1, q_2), aba)$

$\vdash ((q_0, q_1, q_2), ba)$

$\vdash ((q_0, q_3), a)$

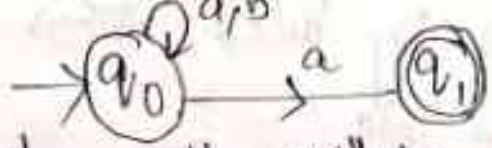
$\vdash (q_0, q_1, q_4)$

$w = aaaba$ is accepted by above NFA $\because q_0, q_1, q_4$ is reached while reading a string & it contains final state q_4

$\therefore$ String is Accepted.

* Convert NFA to DFA

i)

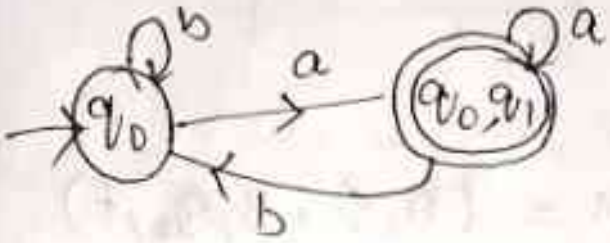


Transition table will be

	a	b
q_0	q_0, q_1	q_0
q_1	ϕ	ϕ

DFA

	a	b
q_0	q_0, q_1	q_0
q_0, q_1	q_0, q_1	q_0



DFA

a)ii)

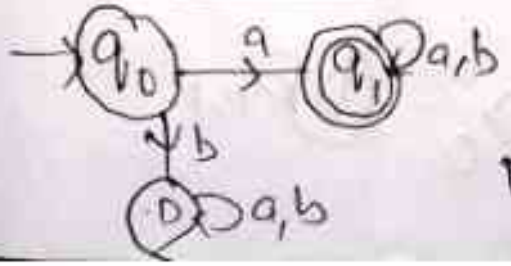
NFA

	a	b
q_0	q_1	ϕ
q_1	q_1	q_1

DFA

	a	b
q_0	q_1	D
D	D	D
q_1	q_1	q_1

D → Dead state



DFA

iii) NFA

	0	1
q_0	$\{q_0, q_1\}$	q_1
q_1	ϕ	$\{q_0, q_1\}$

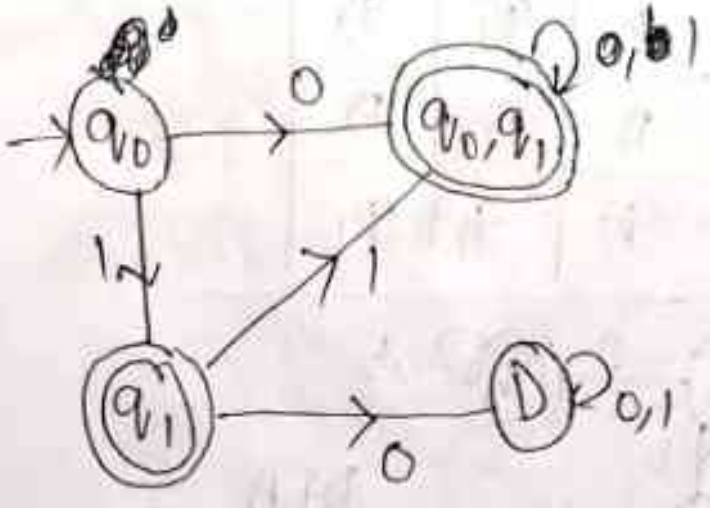
Sol

Given:- NFA

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

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

Q \ Σ	0	1
q_0	$\{q_0, q_1\}$	q_1
q_1	D	$\{q_0, q_1\}$
q_0, q_1	$\{q_0, q_1\}$	$\{q_0, q_1\}$
D	D	D

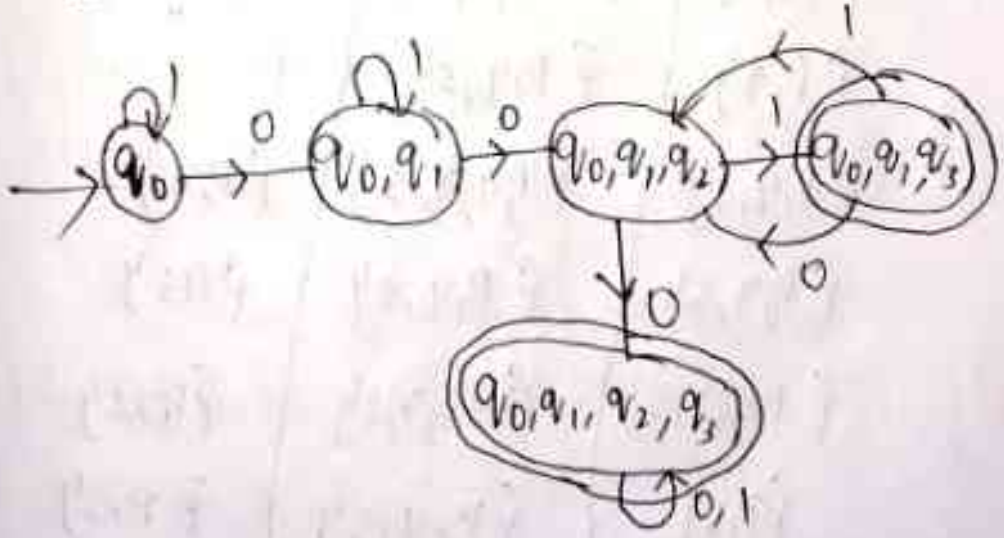


iv) NFA

	0	1
q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	q_1
q_2	q_3	q_3
q_3	ϕ	q_2

DFA $M' = (Q', \Sigma, \delta', q'_0, F')$

δ' / Σ	0	1
q_0	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_3\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_1, q_3, q_2\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$



v) NFA $M = (Q, \Sigma, \delta, q_0, F)$

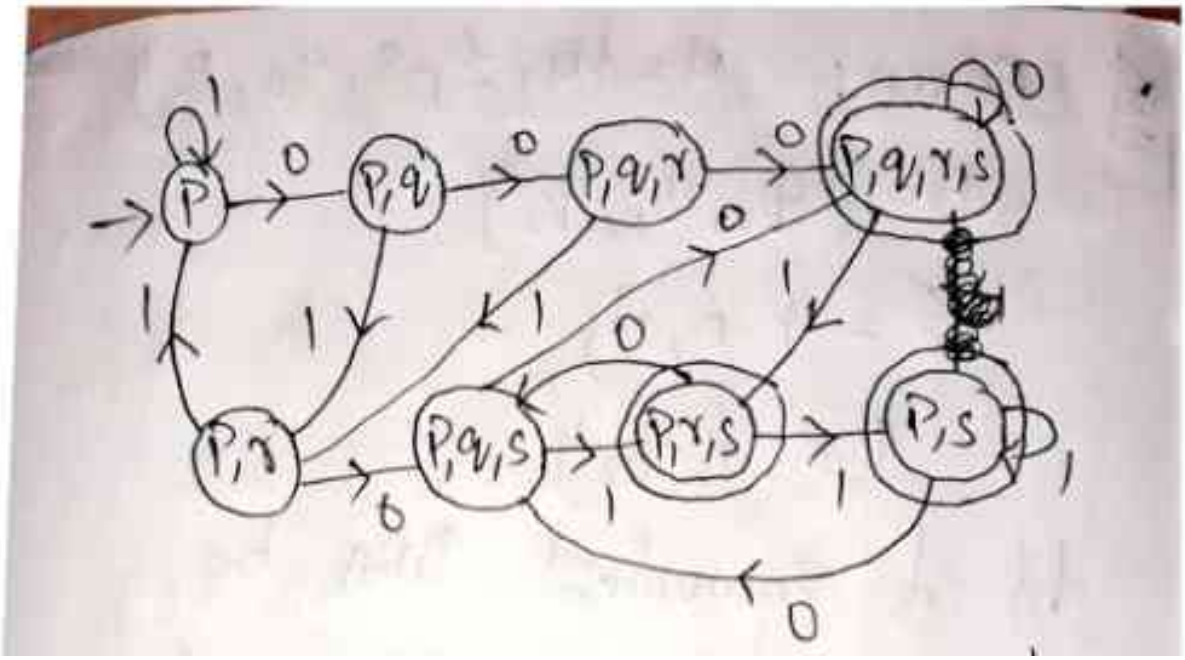
	0	1
p	{p,q}	p
q	r	r
r	s	—
s	s	s

$Q = \{p, q, r, s\}$ $\Sigma = \{0, 1\}$

$q_0 \to \{p\}$ $F \to \{s\}$

DFA

Q/Σ	0	1
p	{p,q}	p
{p,q}	{p,q,r}	{p,r}
{p,q,r}	{p,q,r,s}	{p,r,s}
{p,r}	{p,q,s}	p
{p,q,s}	{p,q,r,s}	{p,r,s}
{p,r,s}	{p,q,s}	{p,s}
{p,q,r,s}	{p,q,r,s}	{p,r,s}
{p,s}	{p,q,s}	{p,s}



* NFA to Equivalent DFA :-
(Subset construction method)

Statement :-

Let $M = \{Q, \Sigma, \delta, q_0, F\}$ be an NFA which accepts the language $L(M)$ then there should be Equivalent DFA denoted by $M' = \{Q', \Sigma, \delta', q_0', F'\}$ such that $L(M) = L(M')$.

Proof :- Let $M = \{Q, \Sigma, \delta, q_0, F\}$ be an NFA for language L

then we define

$M' = \{Q', \Sigma, \delta', q_0', F'\}$

The states of M' are all subset of M

F' states → final states of M'

The elements in Q' $[q_1, q_2 \cdots q_i]$

The elements in Q $[q_1, q_2, \cdots q_i]$

The element $[q_1, q_2, \cdots q_i]$ will be assumed as one state in DFA.

If q_0 is initial state, in DFA
if is denoted as $q_0' = [q_0]$

We define

$$\delta'([q_1 \ldots q_i], a) = [P_1 \ldots P_j]$$

if and only if

$$\delta(\{q_1 \ldots q_i\}, a) = [P_1, P_2 \ldots P_j]$$

The theorem can be proved with induction method by assuming length of string as x

$$\delta'(q_0', x) = [q_1, q_2 \ldots q_i]$$

if and only if

$$\delta(q_0, x) = \{q_1, q_2 \ldots q_i\}$$

Case i) if string $|x|$ is 0

$|x| = 0$, x is ϵ

then $q_0' = [q_0]$ —— (1)

If we assume that the hypothesis is true for input string of length n or less than n then

If xa is a string of length $n+1$, then δ' should be written as

$$\delta'(q_0', xa) = \delta'(\delta'(q_0, x), a)$$

By Induction hypothesis

$$\delta'(q_0', x) = [P_1, P_2, P_3 \ldots P_j]$$

if and only if

$$\delta(q_0, x) = \{P_1, P_2 \ldots P_j\}$$

By definition of δ'

$$\delta'([P_1, P_2 \ldots P_j], a) = [r_1, r_2 \ldots r_j]$$

if and only if

$$\delta(\{P_1, P_2 \ldots P_j\}, a) = \{r_1, r_2 \ldots r_j\}$$

Thus $\delta'(q_0', xa) = [r_1, r_2 \ldots r_k]$

if and only if

$$\delta(q_0, xa) = \{r_1, r_2 \ldots r_k\}$$

is shown by Induction

Thus, $L(M) = L(M')$ // Hence proved

Procedure Subset Construction

1) The start state of NFA will be the start state of DFA. Hence q_0 of NFA is added to Q' then find the transition from start state

2) For each state $[q_1 \dots q_i]$ in Q the transition for each input symbol Σ can be obtained as

i) $\delta'([q_1 \dots q_i], a)$

$= \delta(q_1, a) \cup \delta(q_2, a) \cup \dots$

$\dots \cup \delta(q_i, a)$

$= [q_1 \dots q_k]$ → may be some state

ii) Add the state $[q_1 \dots q_k]$ to DFA if it is not already in Q'.

iii) Find the transition for every input symbol for Σ of string $[q_1 \dots q_k]$, if we get some state $[q_1 \dots q_n]$ which is not in Q' then add to Q'

iv) If there is no new State then Stop the process.

3) For State $[q_1 \ldots q_n] \in Q'$ of DFA If any one state, q is the final State of NFA then all the State in Q' Containg q will became final States of DFA (added to F').

Theorem :- For every NFA there exists an Equivalent DFA

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

DFA $M' = (Q', \Sigma, \delta', q_0', F')$

Q) Convert NFA to DFA by Subset Constn method by showing neccesary intermediate steps in detail

	0	1
q_0	q_0	$\{q_0, q_1\}$
q_1	q_2	q_2
q_2	ϕ	ϕ

8] Given :- $M = \{Q, \Sigma, \delta, q_0, F\}$

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

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

$F = \{q_2\}$

Let its equivalent DFA be

$M' = \{Q', \Sigma, \delta', q_0', F'\}$

$\delta'(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0)$

$= \{q_0\} \cup \{q_2\}$

$= \{q_0, q_2\}$

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

$= \{q_0, q_1\} \cup \{q_2\}$

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

$\delta'(\{q_0, q_1, q_2\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)$

$= \{q_0, q_2\}$

$\delta'(\{q_0, q_1, q_2\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)$

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

$\delta'(\{q_0, q_2\}, 0) = \delta(q_0, 0) \cup (\delta(q_2, 0))$

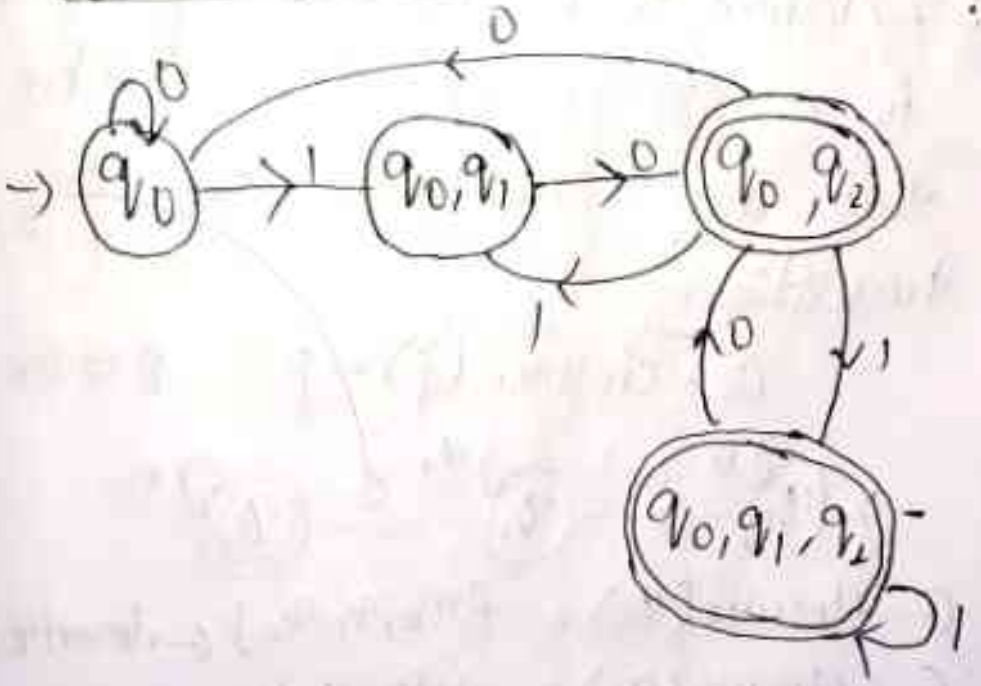
$= \{q_0\} \cup \{\phi\} = \{q_0\}$

$\delta(\{q_0, q_2\}, 1) = \delta(q_0, 1) \cup \delta(q_2, 1)$

$= \{q_0, q_1\} \cup \{\phi\}$

$= \{q_0, q_1\}$

q/ε	0	1
q_0	q_0	$\{q_0, q_1\}$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	q_0	$\{q_0, q_1\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$



* NFA with ε moves:-

The NFA may change the state without reading any input symbol such NFA changes from one state to another state on reading ε (Epsilon) such NFA is called NFA with ε moves

Transition function:-

$$\delta : \{ Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q \}$$

* NFA with ε to NFA without ε moves

ε-closure :- ε-closure of a state is a set of all the states that are reachable on ε-transition

$$\epsilon\text{-closure}(p) = p \quad p \in Q$$

ε-closure(q_0) = $\{q_0, q_1, q_2\}$ ε-closure(q_2) = q_2

ε-closure(q_1) = $\{q_1, q_2\}$

→ For every NFA with ∈ moves there exists a NFA without ∈ moves

$$\delta'(q,a) = \in\text{-closure}(\delta(\hat{\delta}(q,\in),a))$$

where $\hat{\delta}(q,\in) = \in\text{-closure}(q)$

δ → with epsilon moves

δ' → without ∈ moves

Q1) Convert the given NFA with ∈ moves into NFA without ∈ moves.

Sol

$\in$-closure $(q_0) = \{q_0, q_1, q_2\}$

$\in$-closure $(q_1) = \{q_1, q_2\}$

$\in$-closure $(q_2) = \{q_2\}$

Let the Equivalent NFA without ∈ moves be M'

Given:-

	0	1	2
q_0	q_0	ϕ	ϕ
q_1	ϕ	q_1	ϕ
q_2	ϕ	ϕ	q_2

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

$$\delta'(q_0,1) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0,\epsilon),1))$$
$$= \epsilon\text{-closure}(\delta((q_0,q_1,q_2),1))$$
$$= \epsilon\text{-closure}(\delta(q_0,1) \cup \delta(q_1,1) \cup \delta(q_2,1))$$
$$= \epsilon\text{-closure}(q_1)$$
$$= \{q_1,q_2\}$$

$$\delta'(q_0,2) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0,\epsilon),2))$$
$$= \epsilon\text{-closure}(\delta((q_0,q_1,q_2),2))$$
$$= \epsilon\text{-closure}(\delta(q_0,2) \cup \delta(q_1,2) \cup \delta(q_2,2))$$
$$= \epsilon\text{-closure}(q_2)$$
$$= \{q_2\}$$

$\delta'(q_1,0) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_1,\epsilon),0))$

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

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

$= \epsilon\text{-closure}\{\phi\}$

$= \phi$

$\delta'(q_1,1) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_1,\epsilon),1))$

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

$= \epsilon\text{-closure}(\delta(\delta(q_1,1) \cup \delta(q_2,1))$

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

$= \{q_1,q_2\}$

$\delta'(q_1,2) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_1,\epsilon),2))$

$= \epsilon\text{-closure}(\delta(\epsilon(q_1,q_2),2))$

$= \epsilon\text{-closure}(\delta(q_1,2) \cup \delta(q_2,2))$

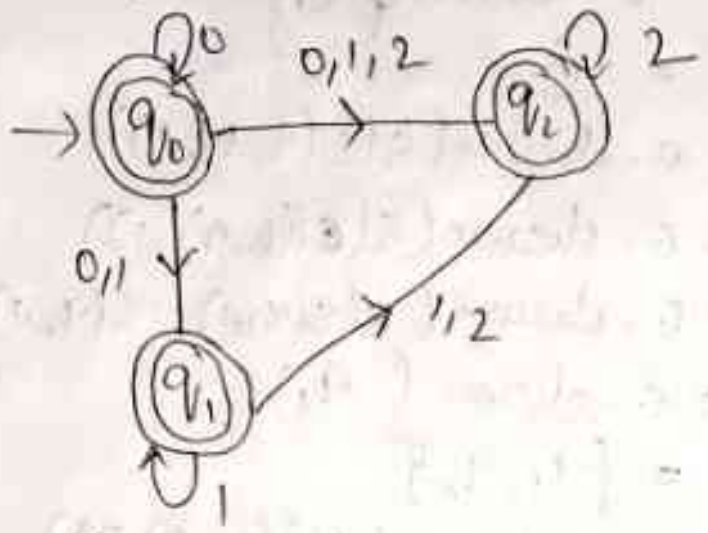
$= \epsilon\text{-closure}(q_2)$

$= q_2$

	0	1	2
q_0	q_0,q_1,q_2	q_1,q_2	q_2
q_1	ϕ	q_1,q_2	q_2
q_2	ϕ	ϕ	q_2

The final state q_2 is present in ϵ-closure of q_0, q_1 & q_2

$\therefore$ q_0, q_1, q_2 are final states



Q2) Conversion of NFA with ϵ → NFA without ϵ

.	a .	b
q_0	q_1	ϕ
q_1	ϕ	ϕ
q_2	ϕ	q_2

Sol

ϵ-closure $(q_0) = \{q_0\}$

ϵ-closure $(q_1) = \{q_1, q_2\}$

ϵ-closure $(q_2) = \{q_2\}$

$\delta'(q_0, a) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, \epsilon), a))$

$= \epsilon\text{-closure}(\delta(q_0, a))$

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

$= \{q_1, q_2\}$

$\delta'(q_0, b) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, \epsilon), b))$

$= \epsilon\text{-closure}(\delta(q_0, b))$

$= \epsilon\text{-closure}(\phi)$

$= \phi$

$\delta'(q_1, a) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_1, \epsilon), a))$

$= \epsilon\text{-closure}(\delta((q_1, q_2), a))$

$= \epsilon\text{-closure}(\delta(q_1, a) \cup \delta(q_2, a))$

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

$= q_1, q_2$

$\delta'(q_1, b) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_1, \epsilon), b))$

$= \epsilon\text{-closure}(\delta((q_1, q_2), b))$

$= \epsilon\text{-closure}(\delta(q_1, b) \cup \delta(q_2, b))$

$= \epsilon\text{-closure}(q_2)$

$= q_2$

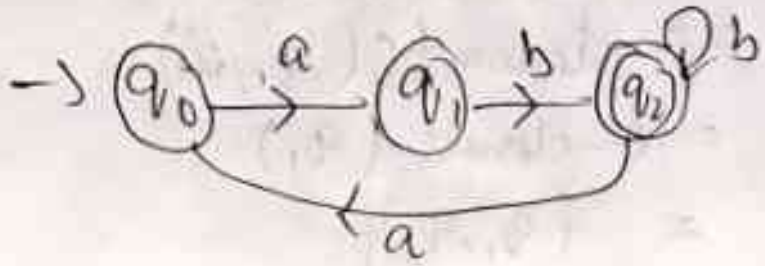
$\delta'(q_2, a) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_2, \epsilon), a))$

$= \epsilon\text{-closure}(\delta(q_2, a))$

$= \epsilon\text{-closure}(\phi)$

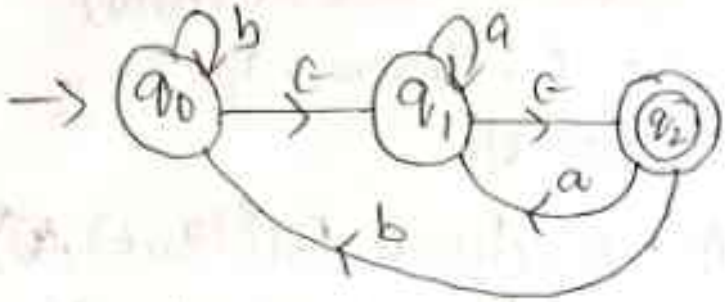
$= \phi$

$\delta'(q_2, b) = q_2$



Q) NFA with ϵ moves to DFA

i)



Sol ϵ-closure $(q_0) = \{q_0, q_1, q_2\}$

ϵ-closure $(q_1) = \{q_1, q_2\}$

ϵ-closure $(q_2) = \{q_2\}$

	a	b
q_0	ϕ	q_0
q_1	q_1	ϕ
q_2	q_1	q_0

ϵ-closure $(q_0) = \{q_0, q_1, q_2\}$

$\{q_0, q_1, q_2\}$ be 'A'

$\delta'(A,a) = \epsilon\text{-closure}(\delta(A,a))$

$= \epsilon\text{-closure}(\delta(\{q_0,q_1,q_2\},a))$

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

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

$= \{q_1,q_2\}$

$\{q_1,q_2\}$ be 'B'

$\delta'(A,b) = \epsilon\text{-closure}(\delta(A,b))$

$= \epsilon\text{-closure}(\delta((q_0,q_1,q_2),b))$

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

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

$= \{q_0,q_1,q_2\} \rightarrow A$

$\delta'(B,a) = \epsilon\text{-closure}(\delta(B,a))$

$= \epsilon\text{-closure}(\delta(\{q_1,q_2\},a))$

$= \epsilon\text{-closure}(\delta(q_1,a) \cup \delta(q_2,a))$

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

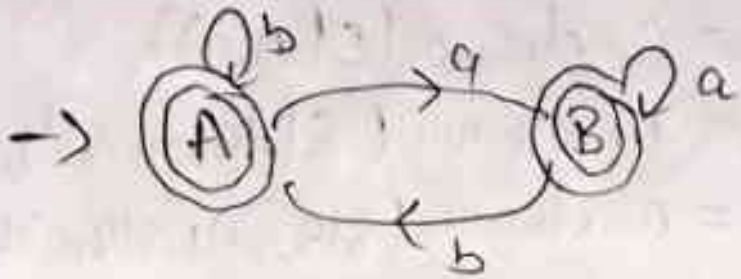
$= q_1,q_2 \rightarrow B$

$\delta'(B,b) = q_0$

q_2 was final state in NFA

$\therefore$ A & B are final state

$\because$ q_2 is present in both states.



8)

	0	1	2
q_0	q_0	ϕ	ϕ
q_1	ϕ	q_1	ϕ
q_2	ϕ	ϕ	q_2

Sol

ε-closure$(q_0) = q_0, q_1, q_2$

ε-closure$(q_1) = q_1, q_2$

ε-closure$(q_2) = q_2$

$\{q_0, q_1, q_2\} \rightarrow A$

$\delta'(A, 0) = \varepsilon$-closure$(\delta(A, 0))$

$= \varepsilon$-closure$(\delta(\{q_0, q_1, q_2\}, 0))$

$= \varepsilon$-closure$(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))$

$= \varepsilon$-closure(q_0)

$= A \rightarrow \{q_0, q_1, q_2\}$

$\delta'(A,1) = \epsilon\text{-closure}(\delta(A,1))$

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

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

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

$= \{q_1,q_2\} \rightarrow B$

$\delta'(A,2) = \epsilon\text{-closure}(\delta(A,2))$

$= \epsilon\text{-closure}(\delta(\{q_0,q_1,q_2\},2))$

$= \epsilon\text{-closure}(\delta(q_0,2) \cup \delta(q_1,2) \cup \delta(q_2,2))$

$= \epsilon\text{-closure}(q_2)$

$= \{q_2\} \rightarrow C$

$\delta'(B,0) = \epsilon\text{-closure}(\delta(B,0))$

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

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

$= \epsilon\text{-closure}(\{\emptyset\})$

$= \emptyset$

$\delta'(B,1) = \epsilon\text{-closure}(\delta(B,1))$

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

$= \epsilon\text{-closure}(\delta(q_1,1) \cup \delta(q_2,1))$

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

$= q_1,q_2 \rightarrow B$

$\delta'(B,2) = \epsilon\text{-closure}(\delta(B,2))$

$= \epsilon\text{-closure}(\delta(\{q_1,q_2\},2))$

$= \epsilon\text{-closure}(\delta(q_1,2) \cup \delta(q_2,2))$

$= \epsilon\text{-closure}(q_2)$

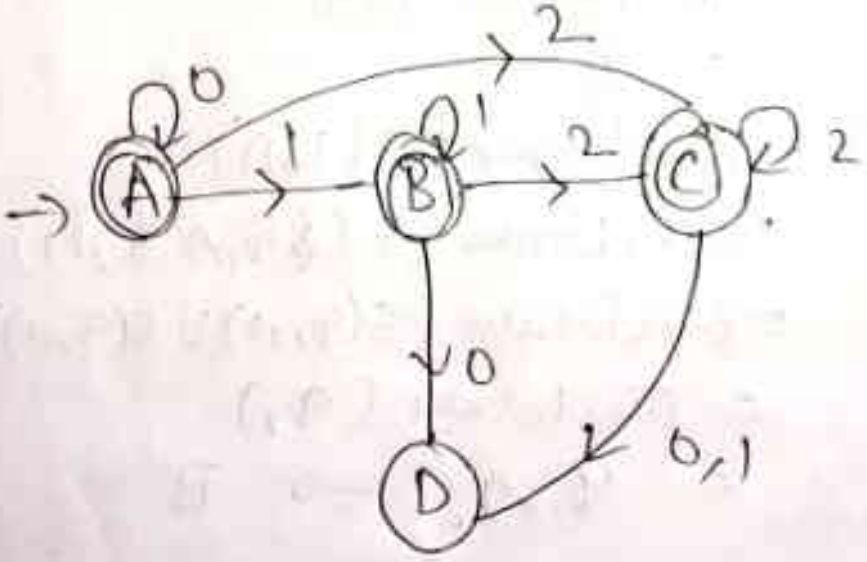
$= q_2 \rightarrow C$

$\delta'(C,0)$

DFA

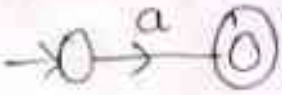
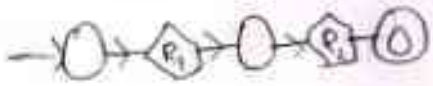
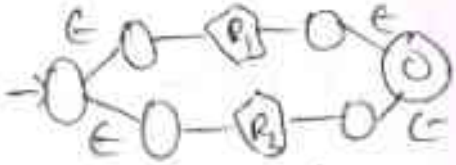
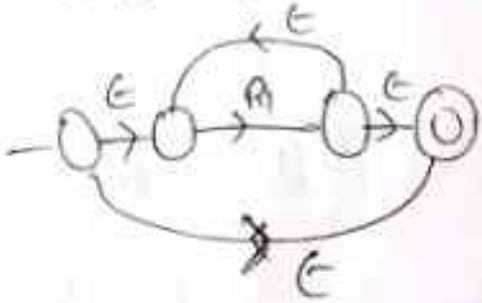
	0	1	2
A	A	B	C
B	D	B	C
C	D	D	C

D → Dead State



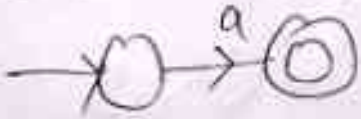
5/oct/2020

* Regular Expression to NFA with ∈ moves conversion

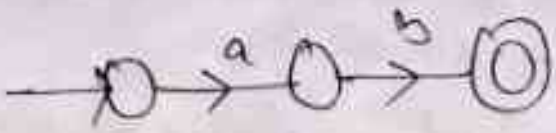
R.E	Regular set	NFA (∈ moves)
i) ∈	{∈}	
ii) ϕ	{}	
iii) a	{a}	
iv) $R_1 . R_2$ ↓ concat		
v) $R_1 + R_2$ ↓ Union		
vi) R_1^* (Kleene closure)		

* Give the Finite Automata (NFA with ∈ moves) for the following...

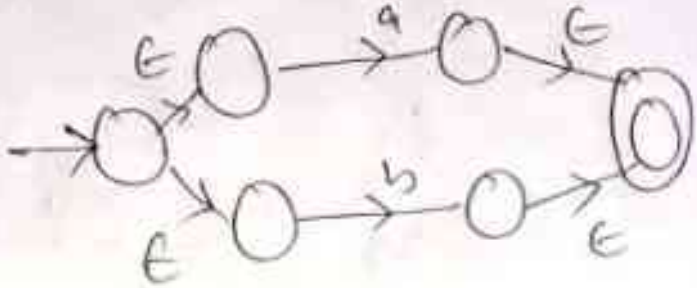
i) R.E = a



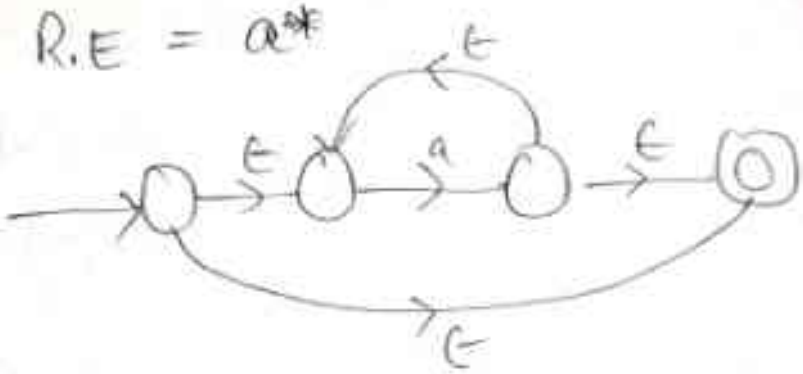
ii) R.E = a.b



iii) R.E = a+b



iv) R.E = a*



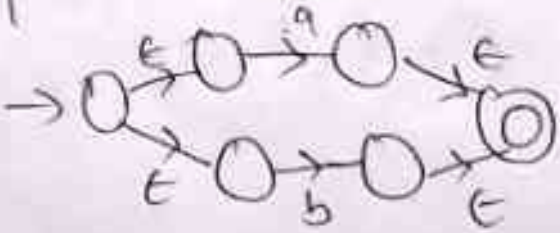
v) R.E = abb

vi) R.E = (a+b)*

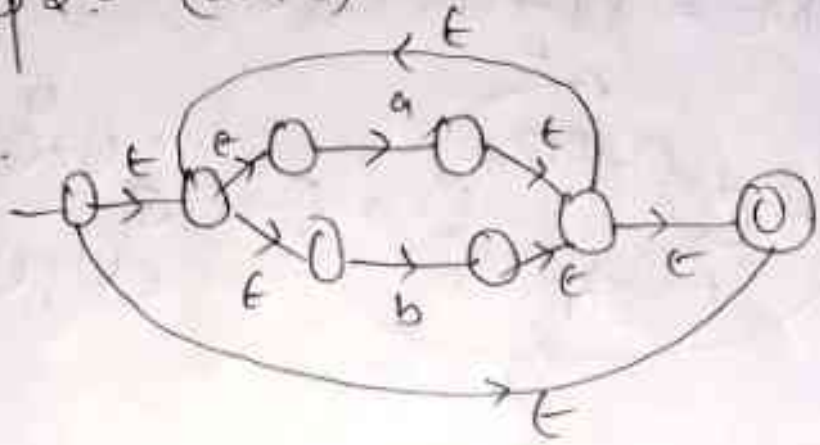
imp

Sol

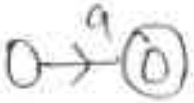
Step 1 :- a+b

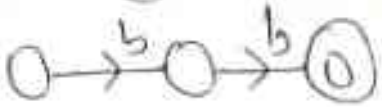


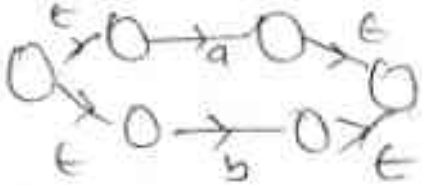
Step 2 :- $(a+b)^*$



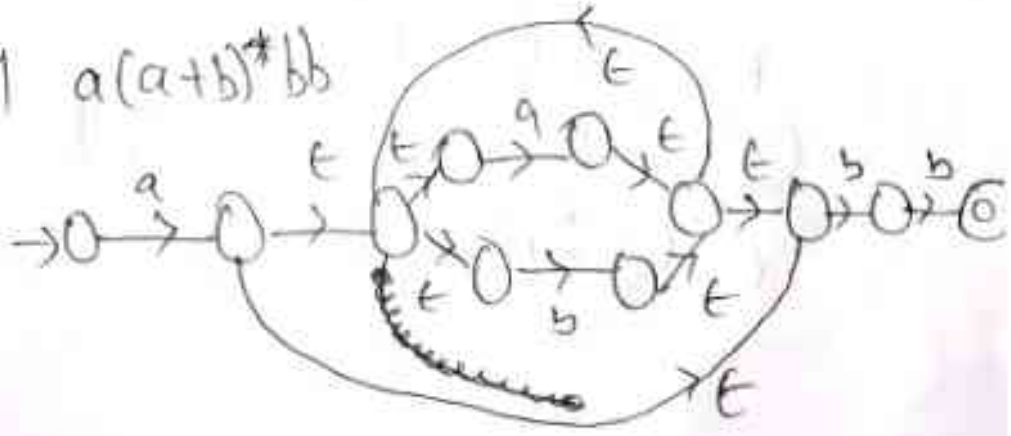
vii) R.E = $a(a+b)^*bb$

$a \rightarrow$ 

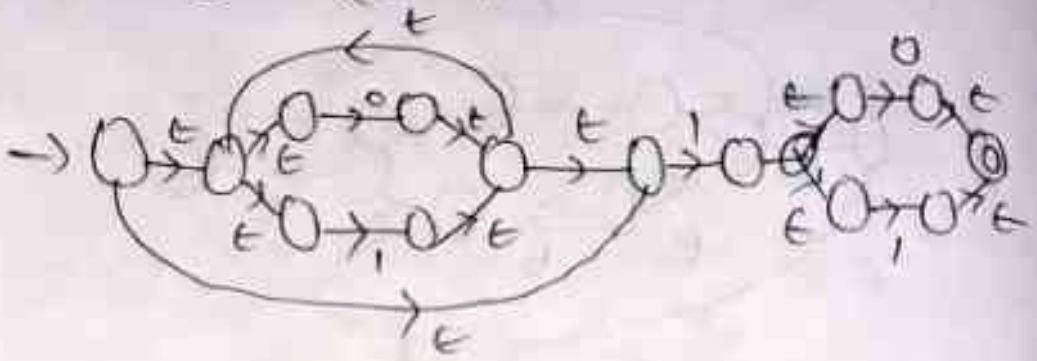
$bb \rightarrow$ 

$(a+b)^* \rightarrow$ 

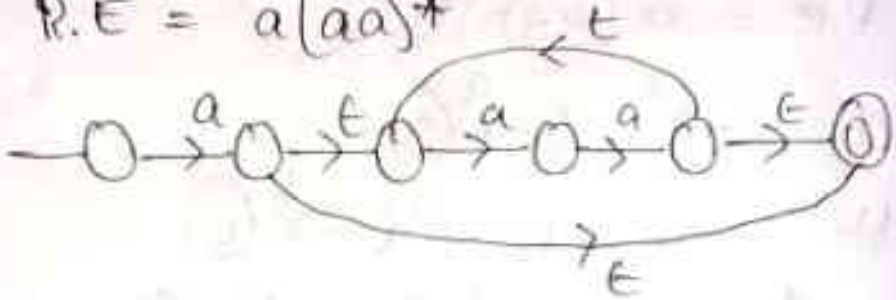
Ans $a(a+b)^*bb$



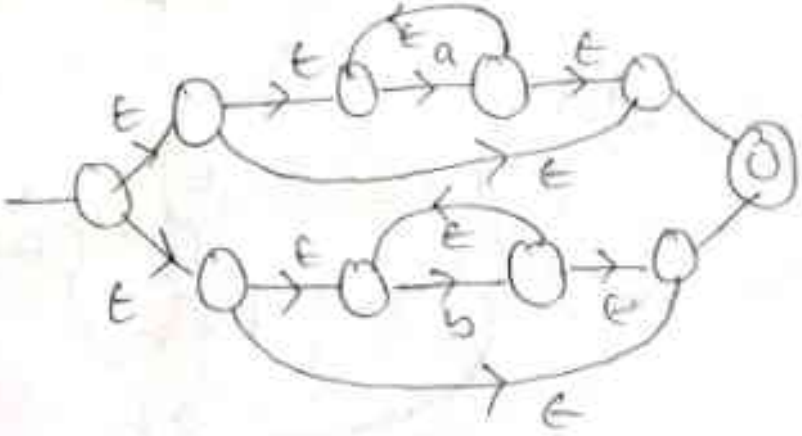
viii) R.E = $(0+1)^*.1.(0+1)$



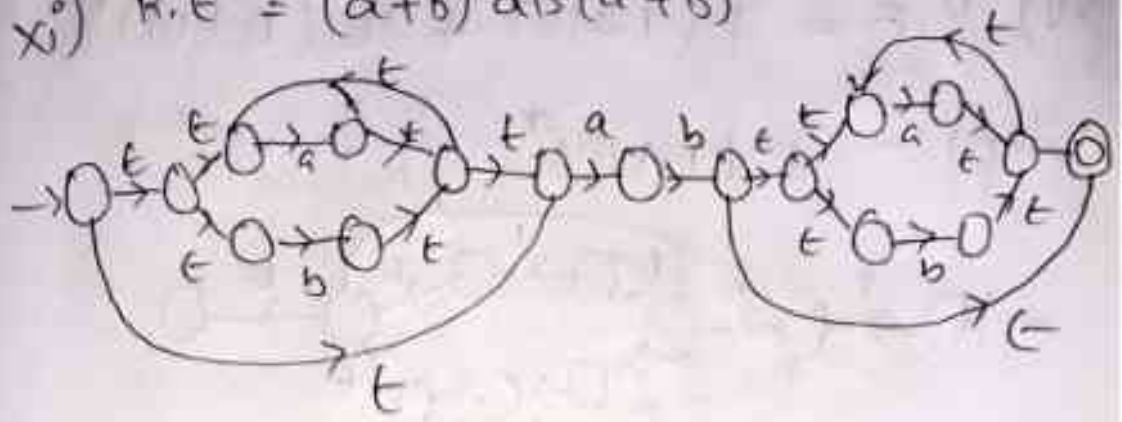
ix) R.E = $a(aa)^*$



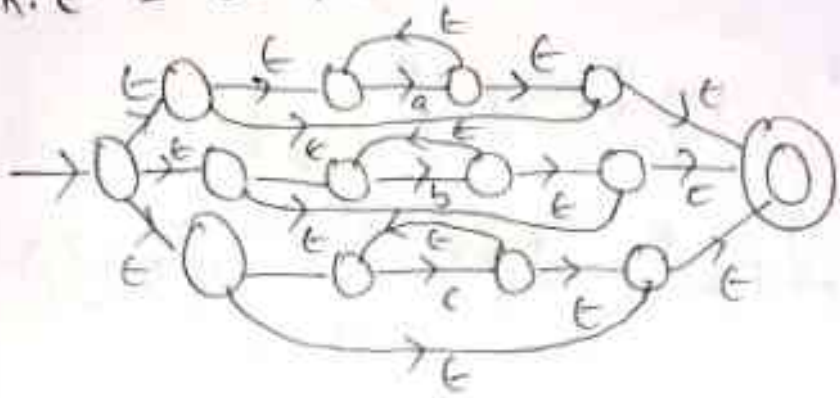
x) R.E = $a^* + b^*$



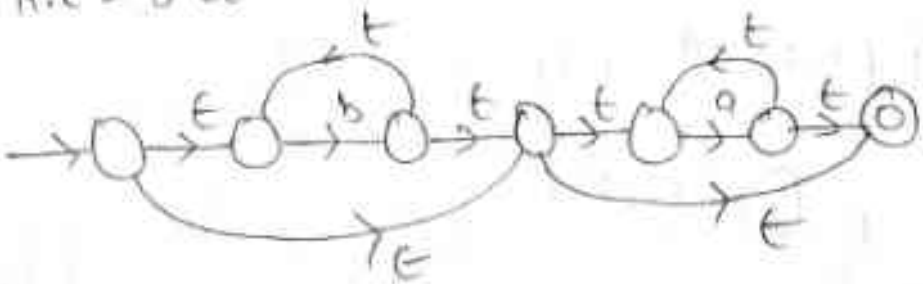
xi) R.E = $(a+b)^*ab(a+b)^*$



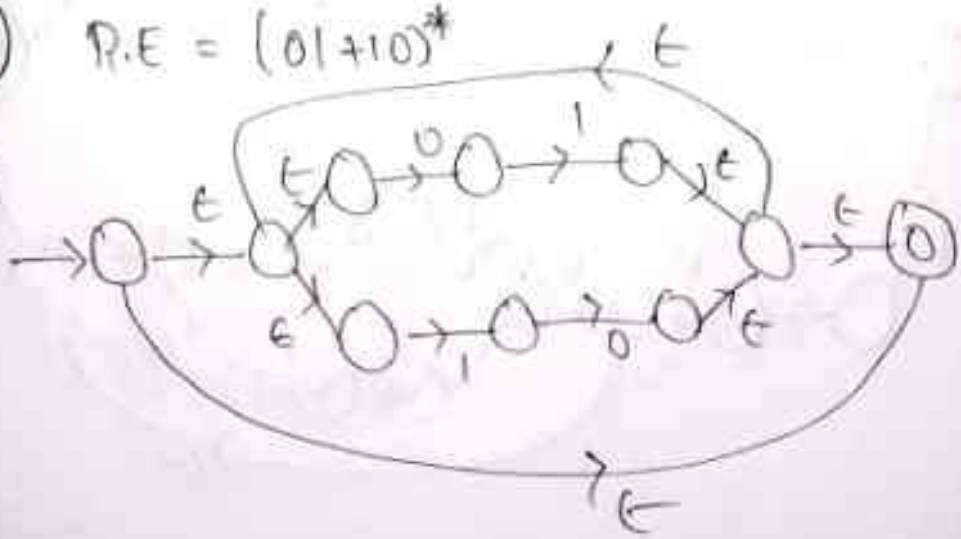
xii) R.E = $a^* + b^* + c^*$



xiii) R.E = b^*a^*

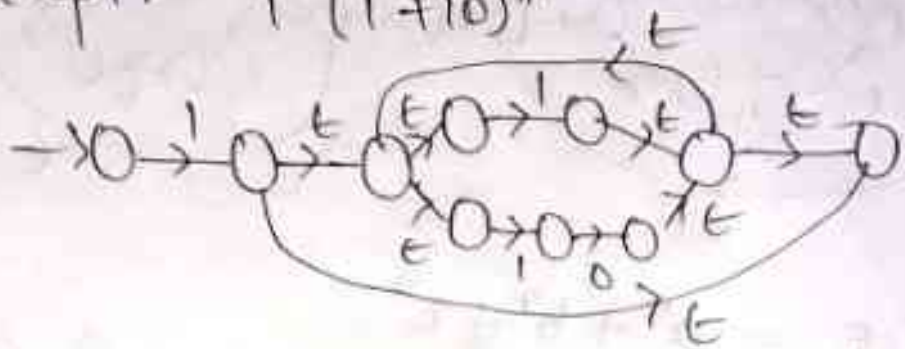


xiv) R.E = $(01+10)^*$

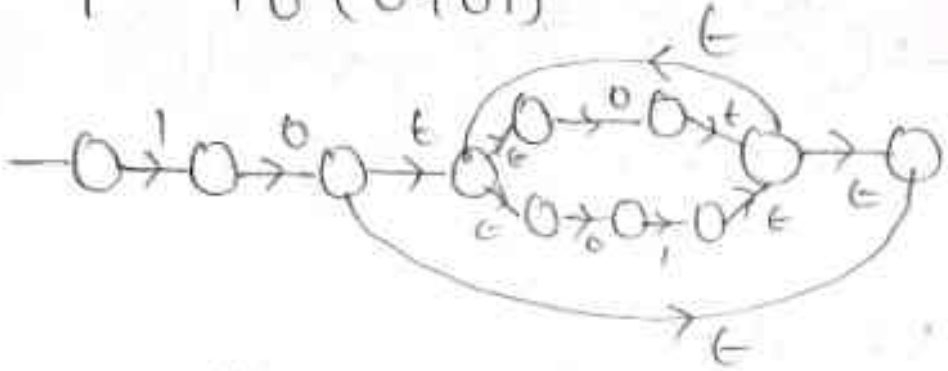


xv) R.E = $1(1+10)^* + 10(0+01)^*$

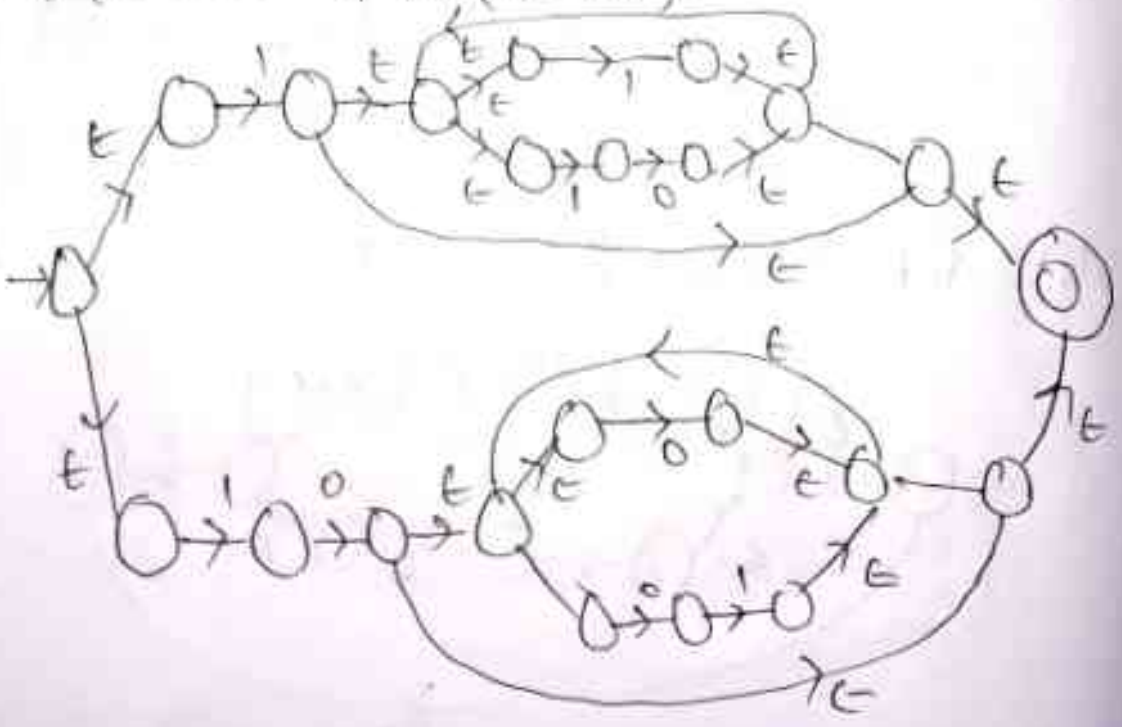
Step 1:- $1(1+10)^*$



Step 2: $10(0+01)^*$



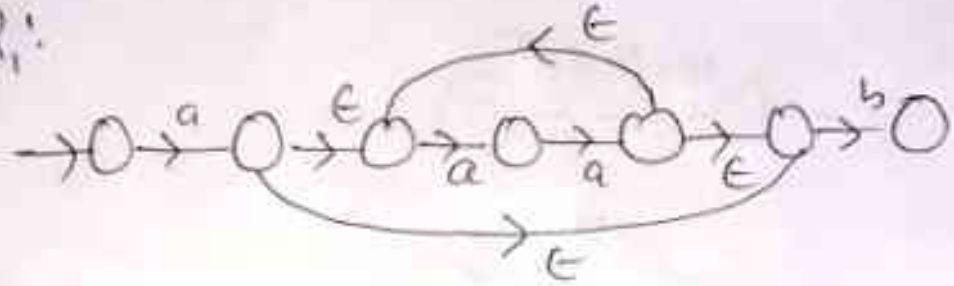
$1(1+10)^* + 10(0+01)^*$



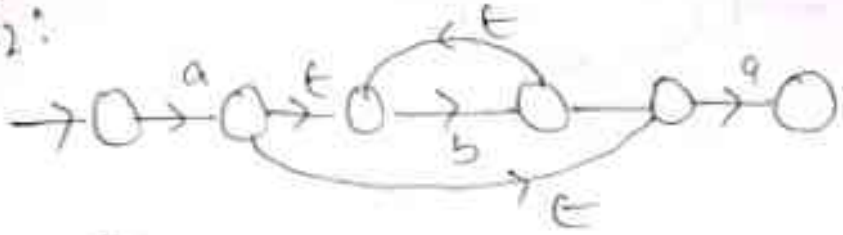
xvi) R.E = $[a(aa)^*b + ab^*a]^*$

let $R_1 = a(aa)^*b$ $R_2 = ab^*a$

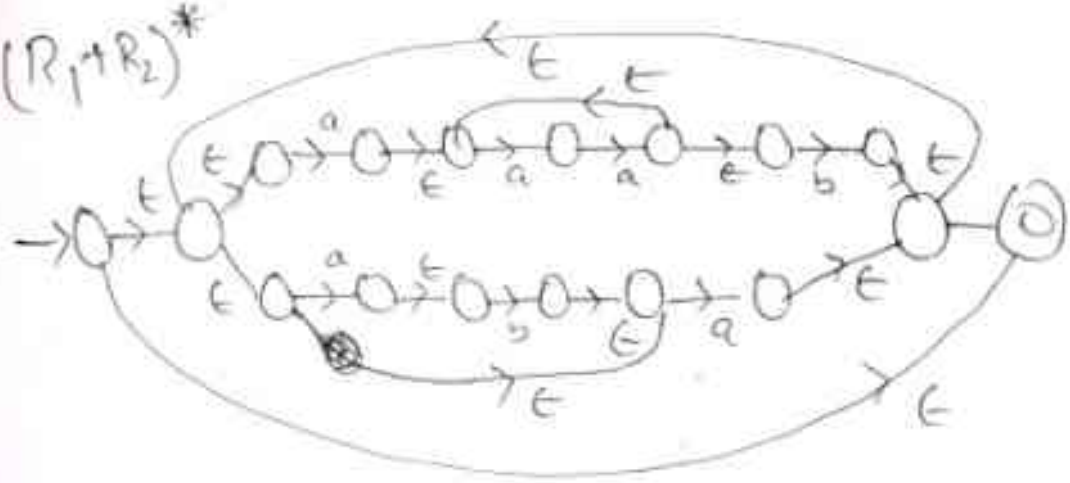
R_1:



R_2:



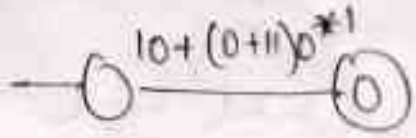
$(R_1+R_2)^*$

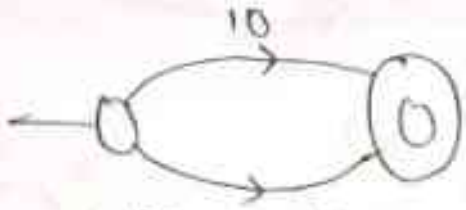


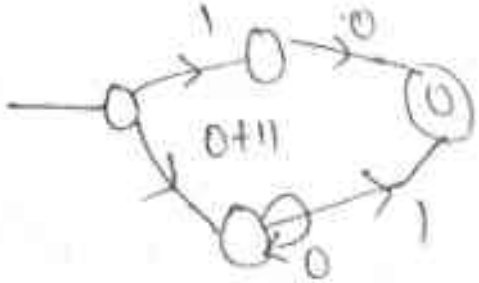
Design F.A for R.E = 10 + (0+11)0*1

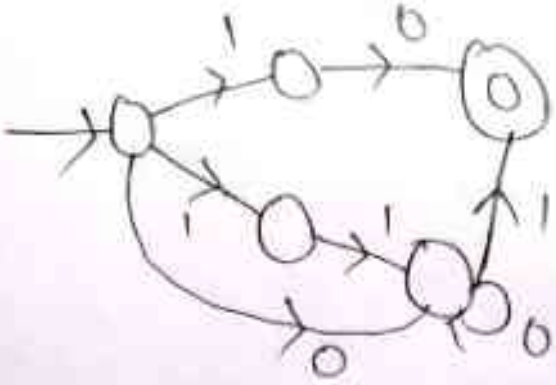
Sol Given:-

R.E = 10 + (0+11)0*1









R.E = 10 + (0+11)0*1

* Arden's theorem:-

Statement:-

If P and Q are two regular Expression over Σ, and if P does not contain ϵ, then the equation in R given by $R = Q + RP$ has a Unique Solution, i.e $R = QP^*$

Proof:- $R = Q + RP$ ——— (1)

Substitute $R = QP^*$

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

$R = Q(\epsilon + P^*P)$

$\because [\epsilon + P^*P = P^*]$

$\therefore R = \theta_1 p^*$

Hence proved.

Similarly :-

$R = \theta_1 + RP$

Sub $R = \theta_1 + RP$

$R = \theta_1 + (\theta_1 + RP)P$

$= \theta_1 + (\theta_1 P + RP^2)$

$= \theta_1 + \theta_1 P + RP^2$

Sub $R = \theta_1 + RP$

$= \theta_1 + \theta_1 P + (\theta_1 + RP)P^2$

$= \theta_1 + \theta_1 P + \theta_1 P^2 + RP^3$

. . . . . .

n times

$R = \theta_1 + \theta_1 P + \theta_1 P^2 + \ldots + \theta_1 P^n + RP^{n+1}$

Sub $R = \theta_1 P^*$

$R = \theta_1 + \theta_1 P + \theta_1 P^2 + \ldots + \theta_1 P^n + \theta_1 P^* P^{n+1}$

$R = \theta_1 (e + \underbrace{P + P^2 + \ldots + P^n + P^* P^{n+1}}_{P^*})$

$[R = \theta_1 P^*]$ ↑ Hence Proved

Note:- $RR^* = R^*R = R^+$

$R.E + RR^* = R^*$

* **Arden's theorem:-**

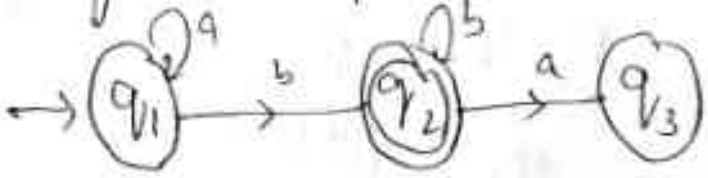
Finite Automata to Regular Expression

⇒ If P, Q be Regular Expression, the Regular Expression of the form $R = Q + RP$ has a Unique Solution

$R = QP^*$

* Convert the following finite Automata to Regular Expression

i)



Sol

$q_1 = q_1 \cdot a + \epsilon$ — (1)

$q_2 = q_1 \cdot b + q_2 \cdot b$ — (2)

$q_3 = q_2 \cdot a$ — (3)

Consider eq ①

$q_1 = q_1 \cdot a + \epsilon$ its in form

$R = R \cdot P + Q$

$R = q_1 \quad Q = \epsilon$

$P = a$

then $R = Qp^*$

i.e $q_1 = \epsilon a^*$

$$\boxed{q_1 = a^*}$$

Substitute q_1 in eq ②

$q_2 = a^* \cdot b + q_2 \cdot b$ its in form

$q_2 \; R = Q + R \cdot P$

$R = Qp^*$

$R = q_2 \quad P = b \quad Q = a^* b$

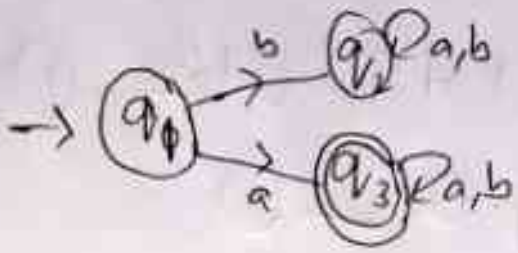
$q_2 = a^* b b^*$

$$\boxed{\text{Note :-} \quad bb^* = b^+}$$

$\{q_2 = a^* b^+ y_{//}$

Its our final state

ii)



Sol

$q_1 = \epsilon$ — (1)

$q_2 = q_2(a+b) + q_1 \cdot b$ — (2)

$q_3 = q_3(a+b) + q_1 \cdot a$ — (3)

Sub q_1 in eq (3)

$q_3 = q_3(a+b) + \epsilon \cdot a$

It in the form

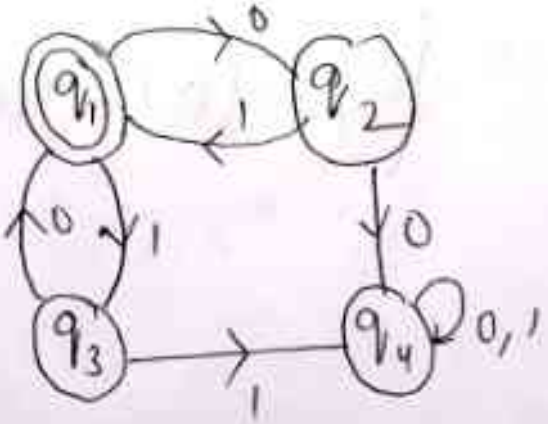
$R = RP + Q$

$R = q_3 \quad P = (a+b) \quad Q = \epsilon \cdot a$

$\therefore R = QP^*$

$\boxed{q_3 = a \cdot (a+b)^*}$

iii)



8)

$$q_1 = q_2 \cdot 1 + q_3 \cdot 0 + \epsilon \quad \text{— (1)}$$

$$q_2 = q_1 \cdot 0 \quad \text{— (2)}$$

$$q_3 = q_1 \cdot 1 \quad \text{— (3)}$$

$$q_4 = q_2 \cdot 0 + q_3 \cdot 1 + q_4(0+1) \quad \text{— (4)}$$

Sub eq (2) & (3) in eq (1)

$$q_1 = q_1 \cdot 0 \cdot 1 + q_1 \cdot 1 \cdot 0 + \epsilon$$

$$q_1 = q_1(01+10) + \epsilon$$

Its in the form

$$R = RP + Q$$

$R = q_1 \quad P = 01+10 \quad Q = \epsilon$

$$R = QP^*$$

$$\Rightarrow q_1 = \epsilon(01+10)^*$$

$$\Rightarrow \boxed{q_1 = (01+10)^*}$$

iv)

Sol:

$$q_1 = q_1 \cdot 0 + \epsilon \quad \text{— (1)}$$

$$q_2 = q_1 \cdot 1 + q_2 \cdot 1 \quad \text{— (2)}$$

$$q_3 = q_2 \cdot 0 + q_3 (0+1) \quad \text{— (3)}$$

$q_1 = q_1 \cdot 0 + \epsilon$ is in the form

$R = R \cdot P + Q_1$, $R = q_1$, $P = 0$, $Q = \epsilon$

$R = Q P^*$

$q_1 = \epsilon 0^*$ $\boxed{q_1 = 0^*}$ — (4)

$q_2 = q_1 \cdot 1 + q_2 \cdot 1$

$q_2 = 0^* \cdot 1 + q_2 \cdot 1$

$R = R \cdot P + Q$ $\quad Q = 0^* \cdot 1$

$P = 1$

$R = q_2$

$R = Q P^*$

$q_2 = 0^* \cdot 1 \cdot 1^*$

$\boxed{q_2 = 0^* 1^+}$ $\quad \boxed{1 \cdot 1^* = 1^+}$

$\therefore R.E = q_1 + q_2$

$= 0^* + 0^* 1^+$ //

* Regular Expression:- ϕ → Empty language
ϵ → Empty string

→ The language accepted by FA can be easily described by simple Expressions called Regular Expressions. It is the most effective way to represent any language

→ The language accepted by some Regular Expression are referred as Regular languages

→ A Regular Expression can also be described as a sequence of pattern that defines a string.

→ a^* → zero or more occurrance of a

→ a^+ → one or more occurrance of a

Operation:-

1. If r_1 is a Regular Expression and S is a Regular Expression then $r_1 + S$, $r_1 . S$, r_1^* are also Regular Expressions.

2. Kleene closure :-

If L is a regular language then its Kleen closure L^* is also a regular language

3. positive closure :-

If L is a regular language then its positive closure L^+ is also a regular language.

eg :- 1. $L = \{ \epsilon, a, aa, aaa \ldots \ldots \}$

Regular expression $\rightarrow a^*$

2. language containing any no. of a's & any no. of b's

Regular Expression $\rightarrow (a+b)^*$

* Properties of Regular set :-

-> Any set that represents the value of regular Expression is called a regular set

Properties:-

Closure properties

Closure properties on Regular languages are defined as certain operations on regular language which guarnteed to Produce regular language

Decision properties

A decision property for a class of language is an algorithm that takes a formal description of a language (Eg DFA) and tells wheather or not some property holds.

* Closure properties

1) Union:- The Union of two regular languages is regular

If L_1 is regular & L_2 is regular then $L_1 \cup L_2$ is also regular.

Proof:- let $RE_1 = a(aa)^*$

$RE_2 = (aa)^*$

so $L_1 = \{a, aaa, aaaaa \ldots\}$

$L_2 = \{\epsilon, aa, aaaa, \ldots\}$

$L_1 \cup L_2 = \{\epsilon, a, aa, aaa \ldots\}$

(string of all possible length)

$RE(L_1 \cup L_2) = a^*$ which is regular expression itself.

2) Intersection:- If L_1 is regular set & L_2 is regular set then $L_1 \cap L_2$ is also regular.

Proof:- $RE_1 = a(aa)^*$ $RE_2 = (aa)^*$

$L_1 = \{a, aa, aaa, aaaa \ldots\}$ (∵ all possible length)

$L_2 = \{\epsilon, aa, aaaa, \ldots\}$ (∵ even length with ϵ)

$L_1 \cap L_2 = \{aa, aaaa, \ldots\}$

(∵ even length)

$RE(L_1 \cap L_2) = aa(aa)^*$ which is regular expression itself

③ Complement:-
The Complement of regular expression is regular.

Proof:- RE = (aa)*

L = {∈, aa, aaaa, ...}
(even length including ∈)

Complement of L = all string not in L

L' = {a, aaa, aaaaa.....}
(odd length).

RE (L') = a(aa)* which is regular expression itself.

④ Difference:-
The Difference of two regular set is regular.

Proof:- RE_1 = a(a*) RE_2 = (aa)*

L_1 = {a, aa, aaa, ...} L_2 = {∈, aa, aaaa.....}
(all length) (even with ∈)

$L_1 - L_2$ = {a, aaa, ...} (odd length)

RE($L_1 - L_2$) = a(aa)* which is regular

⑤ Reversal:-

The reversal of a regular set is regular.

Proof:- If L is regular L^R is also regular.

Let, $L = \{01, 10, 11, 10\}$.

$RE(L) = 01 + 10 + 11 + 10$

$L^R = \{10, 01, 11, 01\}$

$RE(L^R) = 10 + 01 + 11 + 01$

which is regular itself.

⑥ Kleen closure:-

The closure of regular set, is regular.

If $L = \{a, aaa, aaaaa \ldots\}$ (odd length)

$RE(L) = a(aa)^*$

$L^* = \{a, aa, aaa, \ldots\}$

(all lengths)

$RE(L^*) = a(a^*)$

which is regular.

⑦ Concatenation :-

The Concatenation of two regular set is regular.

Proof - $RE_1 = (0+1)^*0$

$RE_2 = 01(0+1)^*$

$L_1 = \{0, 00, 10, 000, 010 \ldots\}$

(ending with 0)

$L_2 = \{01, 010, 011, \ldots\}$

(beginning with 01)

$L_1 . L_2 = \{001, 00010, 10011 \ldots\}$

(set of string Containg 001)

$RE(L_1 . L_2) = (0+1)^* 001 (0+1)^*$

which is regular Expression itself

⑧ Homomorphism :-

→ A String homomorphism is a function on string that works by substituting a particular string for each symbol

Suppose Σ & Σ' are Alphabets then function

$$h: \Sigma \rightarrow \Sigma'$$

It is called homomorphism.

→ In other words homomorphism is a substitution in which each letter is replaced with a string.

$$w = x_1 . x_2 \ldots . x_n$$

$$h(w) = h(x_1)\, h(x_2) \ldots . h(x_n)$$

If L is a language then its homomorphic image is defined as

$$h(L) = \{ h(w) : w \in L \}$$

Ex: a) $\Sigma = \{0,1\}$ $\Sigma' = \{0,1,2\}$ defined by

$h(0) = 01$ $h(1) = 112$

find $h(010)$ & homomorphic image of $L = \{00, 010\}$

Sol $h(010) = h(0)\, h(1)\, h(0)$

$= 01\ 112\ 01$

$h(010) = 01112\,01$

$L = \{00, 010\}$

$h(L) = \{h(00), h(010)\}$

$h(L) = \{0101, 01112201\}$

↳ homomorphic image

(9) Inverse homomorphism:—

It is represented as $h^{-1}(L)$

Q) $h(0) = a$, $h(1) = b$, $h(2) = ab$

$\Sigma = \{0, 1, 2\}$ $\Gamma = \{a, b\}$

$L = \{ababa\}$

↳ It is homomorphism form

$L = \{abaab\}$

$h^{-1}(L) = \{0101, 22, 201, 012\}$

(we get multiple possibilities)

* Decision properties of Regular language

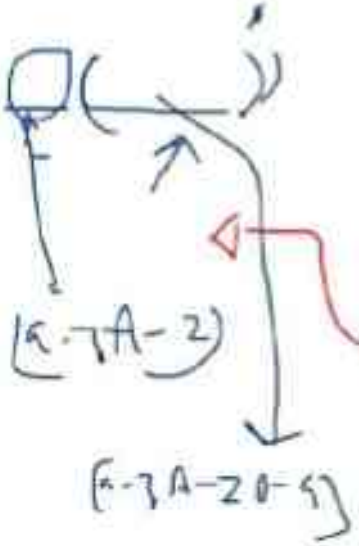
1) Converting among representation.
   (∈ - NFA to DFA &
   DFA to NFA)

2) Testing Emptiness of regular language
   (Testing if a regular generated by automaton is empty)

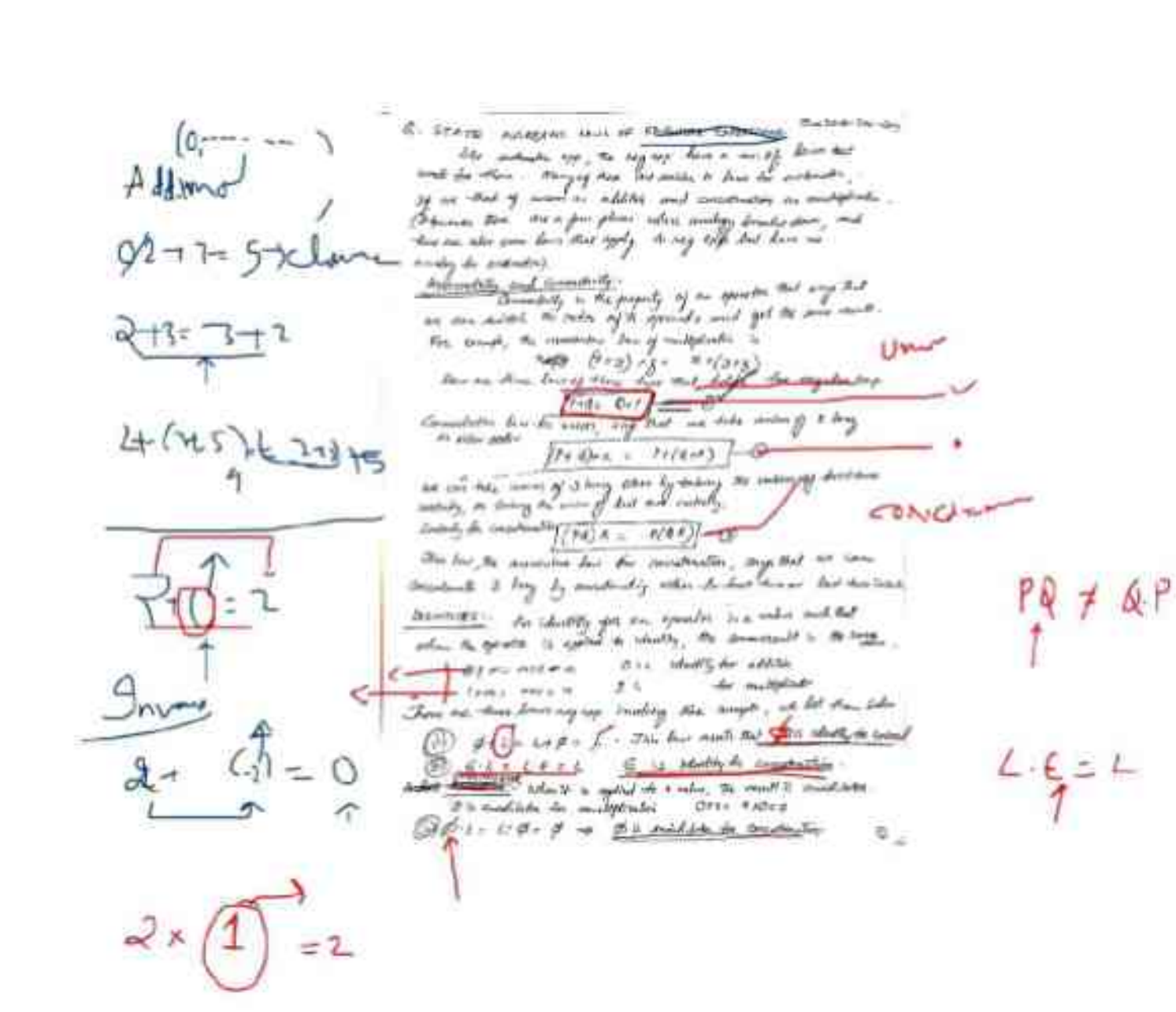
3) Testing membership of regular language
   (processing input symbols to check its member or not)

Reg ex

steps



Scanned with CamScanner



Scanned with CamScanner

DISTRIBUTIVE LAWS — A distributed law involves two operators and asserts that one operator can be pushed to be applied to each argument of the operator individually.

$x*(y+z) = (x*y)+(x*z)$ → Distributive law of multiplication over addition.

However there is an analogous law for reg expression, that we must state in 2 forms

Since concatenation is not commutative, these laws are

$P(Q+R) = PQ+PR$ → This law is the Right distributed law of concatenation over union.

$(Q+R)P = QP+RP$ → This law is the Right distributed law of concatenation over union.

IDEMPOTENT LAW:—

[We write idempotent and closure laws of reg lang on Dec-2015]

An operator is said to be idempotent if the result of applying it to two of the same values as argument is that value.

(The common arithmetic operators are not idempotent $x+x \neq x$, $x*x \neq x$)

However union and intersection are common examples of idempotent operators.

Therefore reg exp, we assert the following laws:

$L+L = L$ → Idempotent law for union, states if we take union of identical exp, we can replace them by one copy of exp.

Laws of closure:- There are various operators involving closure operators.

1. $(L^*)^* = L^*$. This law says that closing an expression that is already closed doesn't change the lang.

2. $\emptyset^* = \epsilon$ The closure of $\emptyset$ contains only the string ϵ.

3. $\epsilon^* = \epsilon$ It is easy to check that only string that can be formed by concatenating any no: of copies of empty string is the empty string itself.

4. $L^+ = L \cdot L^* = L^* \cdot L$ (Can be proved) $L^* = \{\epsilon, L, LL, LLL, \cdots\}$ $L^+ = \{L, LL, LLL, \cdots\}$

5. $L^* = L^+ + \epsilon$. The proof is easy, since the expansion of L^* includes every term in the expansion of L^+ except ϵ.

6. $L? = \epsilon + L$ → This rule is really the definition of the operator?

$L^+ = L \cdot L^*$

* Pumping lemma

Statement:-

If L is a language then there exists some positive integer n (pumping length) such that any string $S \in L$ has length greater than or equal to n i.e $|S| \geq n$ then S can be divided into 3 parts $S = xyz$ that satisfy

i) $i >= 0$, $xy^iz \in L$

ii) $|y| > 0$ &

iii) $|xy| <= n$

Proof:- let $M = \{Q, \Sigma, \delta, q_0, F\}$ be a finite automata let L is the language accepted by some DFA 'm' and is regular.

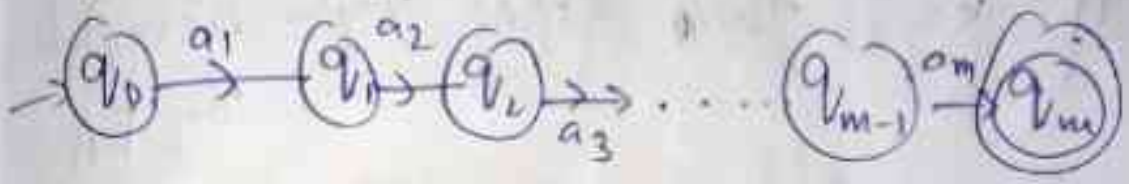
let w be the string of length n or more

i.e $w = a_1 a_2 a_3 \ldots a_m$, $m >= n$

we have m inputs, so we will have m+1 states $(q_0, q_1, q_2 \ldots q_m)$

$q_0 \rightarrow$ Initial state

$q_m \rightarrow$ final state



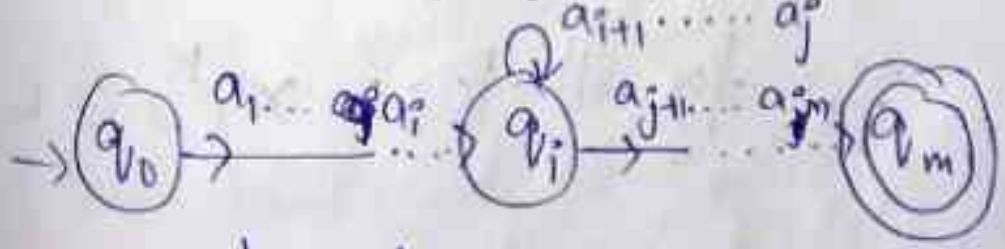
$\therefore |w| \geq n$ by piegeon hole principle it is not possible to have distinct transition. one of the state will have loop

let w is divided into 3 substring

$x = a_1 a_2 \ldots a_i$

$y = a_{i+1} a_{i+2} \ldots a_j$

$z = a_{j+1} a_{j+2} \ldots a_m$



x takes from q_0 to q_i

y takes from q_i to itself

z takes from q_i to q_m

Then minimum string that can be accepted by above FA is xz i.e for $k=0$ $(xy^kz \in L)$

If $k=1$ then string xyz accepted by FA

If $k=2$ then string xy^2z accepted by FA

i.e for all $k>0$, the FA goes from q_0 to q_i on input x, loops from q_i to q_i on Input y and then goes to accepting state on input z after accepting the complete string w.

$\therefore$ for all $k \geq p$ $xy^kz \in L$

→ Pumping lemma is used as a proof for irregularity of a language

→ If there exists atleast one string made from pumping which is not in L surely L is not regular.

* Show that the language $L = \{a^n b^n \mid n >= 1\}$ is not regular.

Sol: Given:- $L = \{a^n b^n \mid n >= 1\}$

The language has equal no. of a's followed by equal no. of b's

$L = \{ab, aabb, aaabbb, a^4b^4, a^5b^5 \ldots\}$

Let Assume the given language is regular

Let the string be $a^P b^P$ (pumping length $P = 3$)

$S = aaaaabbbbb \in L$

Case i) $\underbrace{aaaaa}_{x}\underbrace{bbbb}_{y}\underbrace{b}_{z}$ $x = aa$, $y = aaa$, $z = bbbbb$

$xy^iz, i \geq 0, xy^2z$

$aa\ aaaaaa\ bbbbbb \notin L$

No. of a's > No. of b's $\times$

Case ii) $\underbrace{aaaaa}_{x}\underbrace{bbb}_{y}\underbrace{bb}_{z}$

$x = aaaaa$ $y = bbb$ $z = bb$

xy^2z

$aaaaa\ bbbbbbbb \notin L$ $\times$

Case iii) $\underbrace{aaa}_{x}\underbrace{aabb}_{y}\underbrace{bbb}_{z}$

$x = aaa \quad y = aabb \quad z = bbb$

xy^2z

aaa aabbaabbbbb, $\notin L$

$|xy| \not\leq p$ ✗ ✗

∴ our assumption was wrong.

$L = \{a^n b^n \mid n \geq 1\}$ is not a regular language

* Minimization of DFA :-

Minimization of DFA

- Table-filling Algorithm (Myhill-Nerode Algorithm)
- Partitioning Algorithm

Distinguishable pair

Equivalent pair

* Minimization of DFA
It refers to those states whose presence or absence does not affect the language accepted.
The states that can be eliminated from the automata without affecting the language accepted are
1) Unreachable states (inaccessible states)
2) Dead states
3) Equivalent states (indistinguishable states)

* Dead state: A dead state is a non final state which has no outgoing transitions — ∅
* Unreachable state: Unreachable states are those states that does not reach the initial state on any input

* Distinguishable States: Two states P, Q are called distinguishable if $\delta(P,w) \in F$, $\delta(Q,w) \notin F$ or vice versa for all $w \in \Sigma^*$
i.e distinguishable pair is that atleast one string w such that on P $\delta(P,w)$ & $\delta(Q,w)$ is an accepting state / final state and the other is non final state

* Equivalent States: P & Q are said to be equivalent if for all input string w $\delta(P,w)$ is an accepting state if and only if $\delta(Q,w)$ is an accepting state.

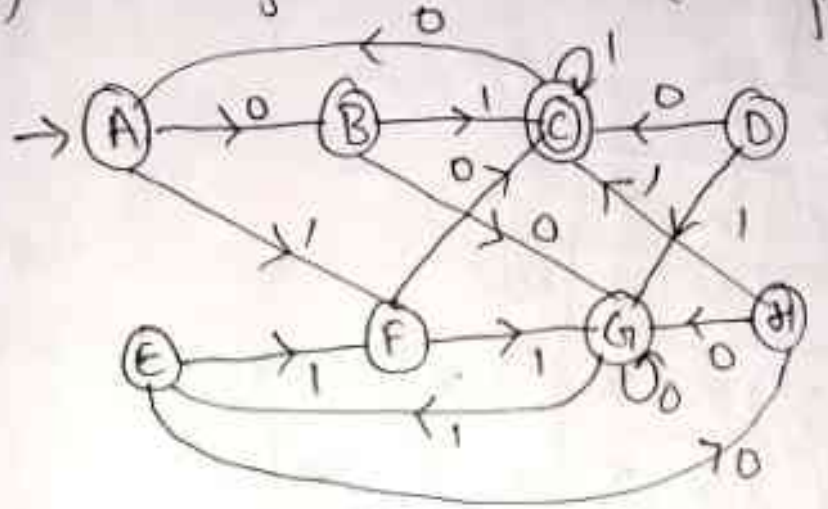
Minimization of DFA

Table Filling Algorithm (Myhill-Nerode Algorithm) | Partitioning Algorithm

Minimization by Table-Filling method
* Before applying this procedure or algorithm remove all the unreachable states from the DFA
* Only DFA's are minimized (not NFA's)

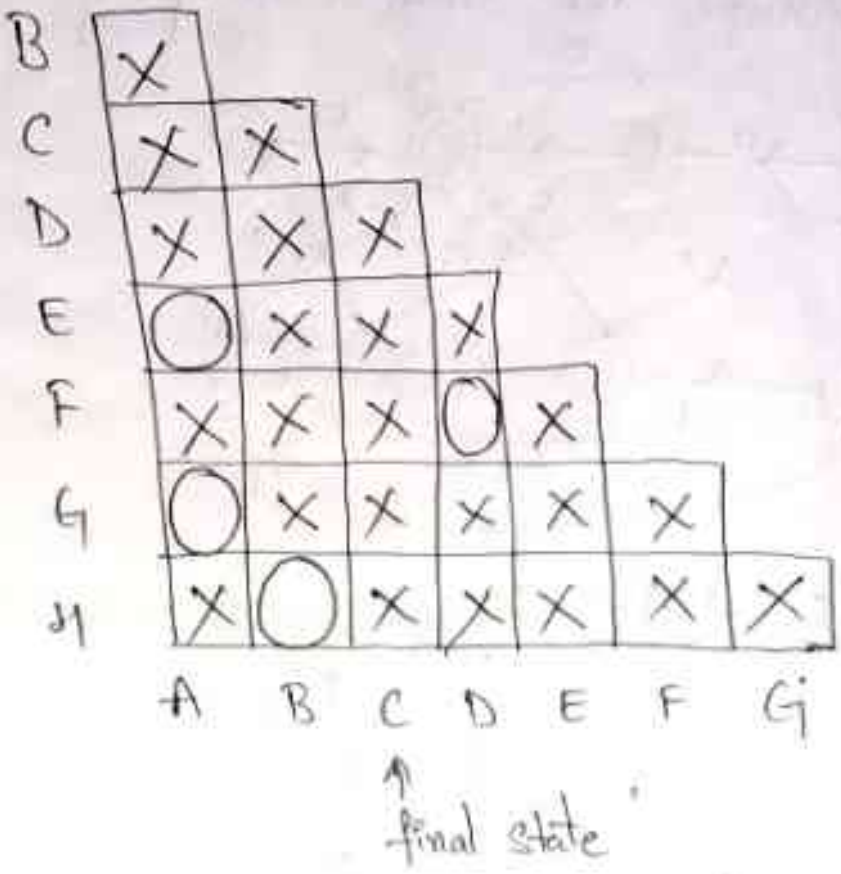
Procedure
The procedure of table filling algorithm is as follows:
1) For each pair (P,Q) $P \in F$ & $Q \notin F$ then the pair (P,Q) is distinguishable and mark it as X
2) Let (P,Q) be a pair of states such that for input symbol 'a' $\delta(P,a) = R$ & $\delta(Q,a) = S$ if (R,S) is already marked as distinguishable then the pair (P,Q) is also distinguishable & mark it as X
3) Repeat step 2 for each pair (P,Q) till the pair P,Q is not marked or atleast one

Q1) Minimize the DFA. (Table-filling method)



Transition table:-

	0	1
A	B	F
B	G	C
C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C



final state with all non-final states are distinguishable so marked X

	0		1	
A,B	B,G	-	F,C	X
A,D	B,C	X	(F,G)	
A,E	B,H	-	F,F	-
A,F	B,C	X	G,F	
A,G	B,G	-	F,E	-
A,H	B,G	-	F,C	X

Can't say

	0		1	
B,D	G,C	×	C,G	
B,E	G,H		C,F	×
B,F	G,C	×	C,G	
B,G	G,G		C,E	×
B,H	G,G	–	C,C	–

	0		1	
D,E	C,H	×	G,F	–
D,F	C,C	–	G,G	–
D,G	C,G	×	G,E	
D,H	C,G	×	G,C	

	0		1	
E,F	H,C	×	F,G	
E,G	H,G		F,E	×
E,H	H,G		F,C	×

	0		1	
F,G	C,G	×	G,E	
F,H	C,G	×	G,C	

	0		1	
G,H	G,G	–	E,C	×

(A,E) (A,G) (B,H) (D,F) Are remaining pairs

	0	1	
A,E	B,H	F,F	→ Equivalent pair
A,G	B,G	F,E	→ X
B,H	G,G	C,C	→ Equivalent pair
D,F	C,C	G,G	→ Equivalent pair

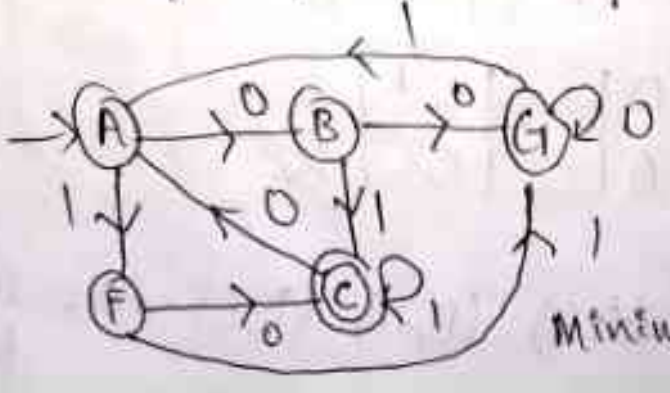
∵ B,H is Equivalent pair

A,E is also Equivalent pair

D,F is Equivalent pair

Transition table

		0	1
A	(A,E)	B	F
B	(B,H)	G	C
	C	A	C
F	(D,F)	C	G
	G	G	A

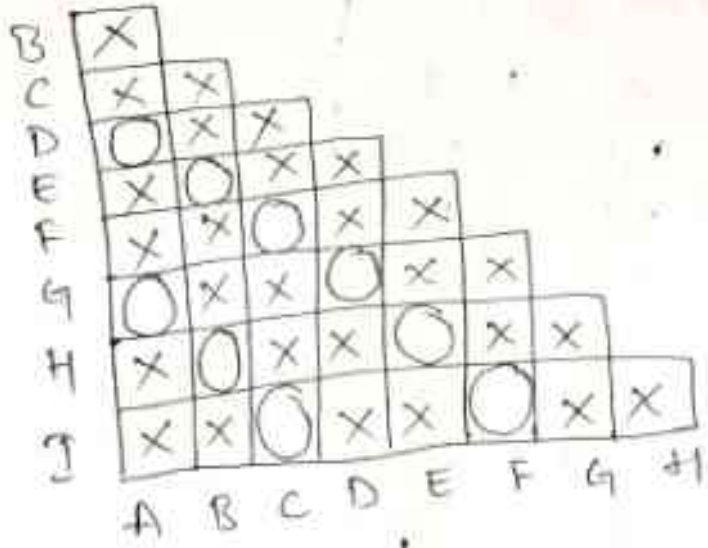


Minimized DFA

Q2)

	0	1
A	B	E
B	C	F
→ C	D	H
D	E	H
E	F	I
→ F	G	B
G	H	B
H	I	C
→ I	A	E

C, F, I are final states



	0		1	
A,B	B,C	X	E,F	X
A,D	B,E	-	E,H	-
A,E	B,F	X	E,I	X
A,G	B,H	-	E,B	-
A,H	B,I	X	E,C	X
B,D	C,E	X	F,H	X
B,E	C,F	-	F,I	-
B,G	C,H	X	F,B	X
B,H	C,I	-	F,C	-

	0		1	
D,E	E,F	X	H,I	X
D,G	E,H	-	H,B	-
D,H	E,I	X	H,C	X
E,H	F,I	-	I,C	-
E,G	F,H	X	I,B	X
G,H	H,I	X	B,C	X

(A,D), (A,G), (B,E), (B,H), (C,F), (C,I), (D,G)
(E,H) , (F,I)

	0	1
A,D	B,E	E,H
A,G	B,H	E,B
B,E	C,F	F,I
B,H	C,I	F,C
C,F	D,G	H,B
C,I	D,A	H,E
D,G	E,H	H,B
E,H	F,I	I,C
F,I	G,A	B,E

* the above 9 pairs can't be proved for Equivalent & distinguishable

* ∴ we can't prove that they are distinguishable at any no. of rounds
So we arrive at a conclusion that all 9 pairs are Equivalent.

(A,D) (D,G) → (A,G) (B,E) (E,H) → (B,H)

(A,D,G) (B,E,H)

(C,F) (C,I) → (F,I)

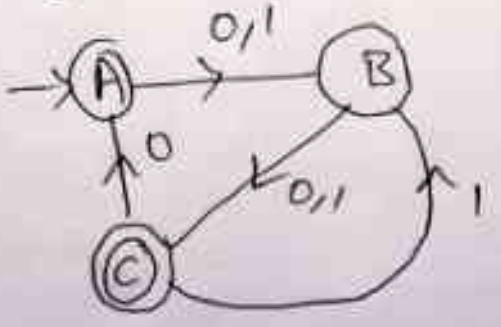
(C,F,I)

The above pairs are Equivalent pairs

		0	1
A	(A,D,G)	B	B
B	(B,E,H)	C	C
C	(C,F,I)	A	B

∵ A,D,C on zero gives B,E,H so B and so on.

Minimized DFA is



19/10/2020 UNIT:-02

Context Free Grammar & language

* Grammar:-

->A Grammar is described using 4-tuples $G = (V, T, S, P)$

V = set of variables or non-terminals

T = set of terminals

S = start symbol

P = Production Rule for terminal & non-terminals.

* Variables are denoted by words Upper case letters.
Variables can be Reduced
Eg:- S, A, B

* Terminals are lower case letters Operators & special symbols are also terminals
Terminals cannot be reduced
Eg:- a, b, +, *, - etc

* Based on production rules grammar are classified into four.

Grammar

Unrestricted grammar	Context-sensitive grammar	Context-free grammar	Regular grammar
$x \to y$	$x \to y$	$x \to v$	$x \to v$
$x, y \in (V \cup T)^*$	$x, y \in (V \cup T)^*$	$y \to (V \cup T)^*$	$y \to$
	$\lvert x \rvert \leq \lvert y \rvert$		almost 1 variable

* Context-free grammar [Accepted by PushDown Automata]

→ The programming language construct is generally given by context-free grammar

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

Eg:- $G = (\{E\}, \{+, *, id\}, E, P)$

$P = \{E \to E+E, E \to E*E, E \to id\}$

$V = \{E\}$

$T = \{+, *, id\}$

$S = E$

* Derivation :-

→ The Process of obtaining a string from the start symbol by replacing variables is called Derivation.

Derivation
- leftmost Derivation
- Rightmost Derivation

Left-most Derivation :- In each step left-most variable is replaced

Right-most Derivation :- In each step right-most variable is replaced

Example :- $E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$

Left-most Derivation

$E \rightarrow E + E$
$\downarrow$
$E \rightarrow E * E + E$
$\downarrow$
$E \rightarrow id * E + E$
$\rightarrow id * id + E$
$\rightarrow id * id + id$

Right-most Derivation

$E \rightarrow E * E$

$\rightarrow E * E + E$

$\rightarrow E * E + id$

$\rightarrow E * id + id$

$\rightarrow id * id + id.$

* Derivation tree or Parse tree

-> The representation of the derivation (either leftmost or right most) in the form of a tree is a parse tree or derivation tree.

Root vertex -> Start symbol

Vertex -> non-terminal

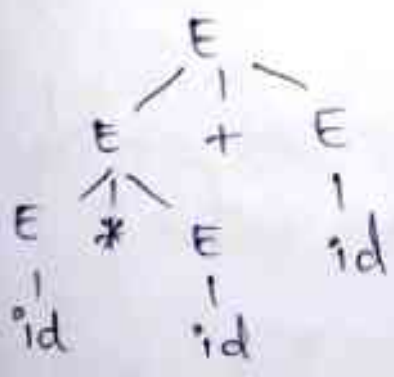
leaves -> terminals

Left-most Derivation , Right-most derivat

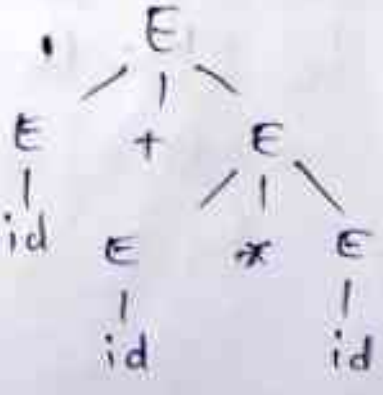
$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow id$



id * id + id



id + id * id

Eg:-

$S \rightarrow aAS \mid aSS \mid \epsilon$,

$A \rightarrow SbA \mid ba$

$V = \{A, S\}$ $T = \{a, b\}$

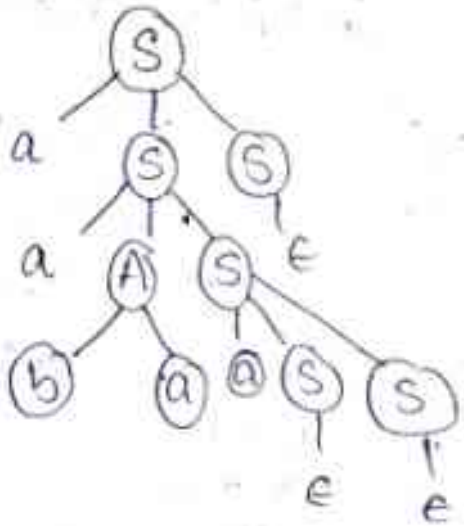
$S = \{S\}$

$P = \{S \rightarrow aAS \mid aSS \mid \epsilon, A \rightarrow SbA \mid ba\}$

generate a string $w = aabaa$

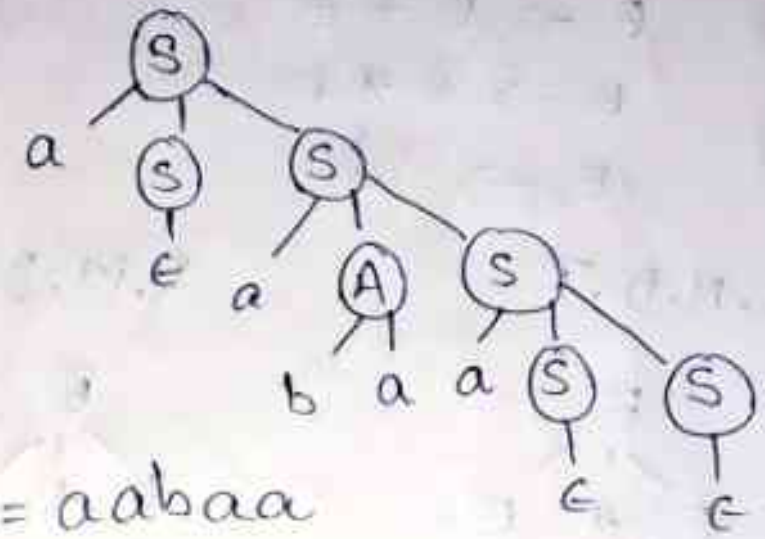
$w = aabaa$

left-most derivation tree



$w = a\ a\ b\ a\ a$

Right derivation tree



$w = aabaa$

* **Ambiguous Grammar**

→ A Grammar is said to be ambiguous if there exists two or more derivation trees or parse trees for a string w.

(or) Two or more left most derivation
(or) Two or more right most derivation also.
(or) If the same variable appears twice on R.H.S.

Ambiguous grammar:-

→ A CFG is said to be ambiguous if there exists more than one derivation tree for given input string.

→ That is more than one LMDT or RMDT

eg:- $E \rightarrow E+E \mid id \mid E*E$

Root cause of Ambiguity:-

↳ precedence of operators are considered

Inherent Ambiguity:-

→ A language is said to be inherently ambiguous if all of its grammar are ambiguous.

→ Ambiguity is a property of grammar not languages

→ Ambiguity of a grammar is undecidable.

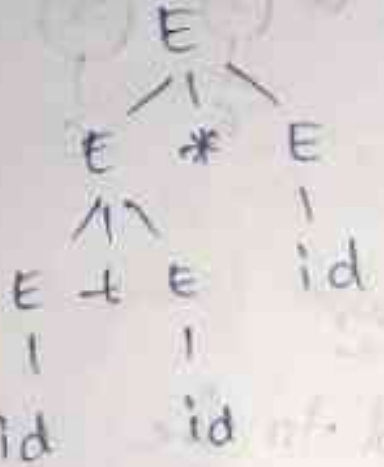
Example :- 1

$E \rightarrow E + E$

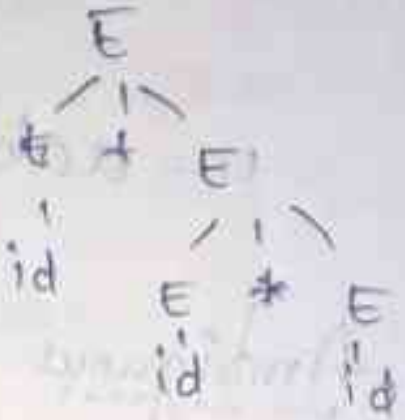
$E \rightarrow E * E$

$E \rightarrow id$

L.M.D.T



L.M.D.T



id + id * id ; id + id * id.

∴ Two parse trees.

The above grammar is ambigous.

Example :- 2

$G = (\{S\}, \{a+b, +, *\}, P, S)$

$P = \{ S \rightarrow S+S \mid S*S \mid a \mid b \}$

String $\omega \rightarrow a + a * b$

Left-most derivation

$S \rightarrow S + S$

$\rightarrow a + S$

$\rightarrow a + S * S$

$\rightarrow a + a * S$

$\rightarrow a + a * b$

Right-most derivation.

$S \rightarrow S * S$

$\rightarrow S + S * S$

$\rightarrow a + S * S$

$\rightarrow a + b * S$

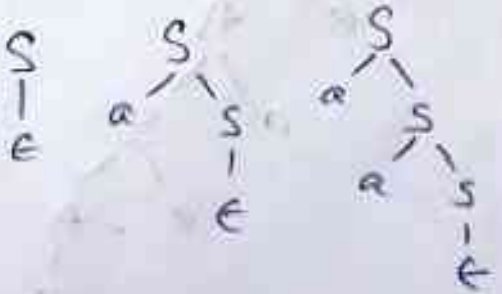
$\rightarrow a + a * b$

Thus, the grammar is ambiguous (∵ multiple derivation).

* Design a Context-free grammar for the languages.

1) $L = \{a^n \mid n \geq 0\}$ $L = \{\epsilon, a, a^2, a^3, \ldots\}$

Sol $S \rightarrow aS/\epsilon$

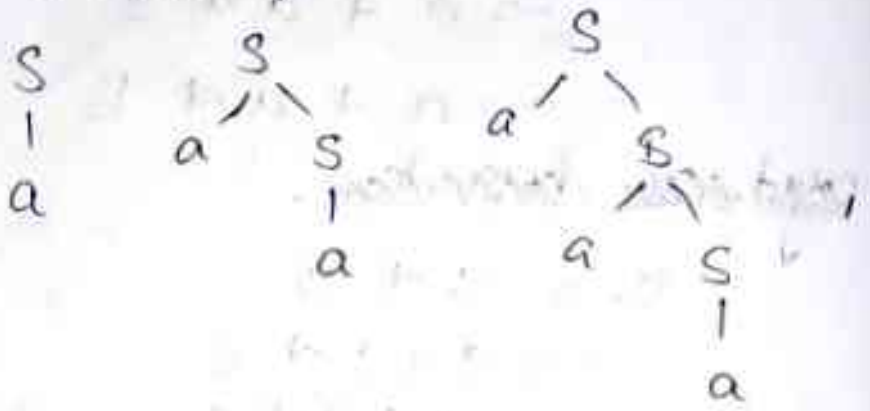


2) $L = \{ a^n \mid n \geq 1 \}$

$L = \{ a, a^2, a^3, a^4 \ldots \}$

Sol

$S \rightarrow aS \mid a$



3) $L = \{ a^{2n} \mid n \geq 1 \}$

Sol

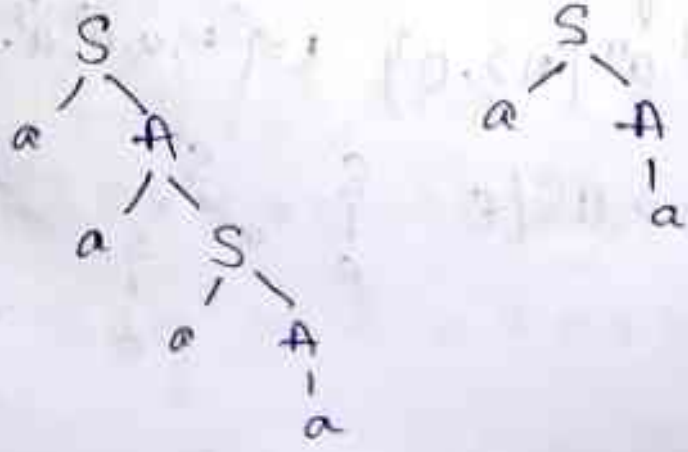
$L = \{ aa, aaaa, a^6 \ldots \}$

$S \rightarrow aaS \mid aa$

(or)

$S \rightarrow aA$

$A \rightarrow aS \mid a$



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

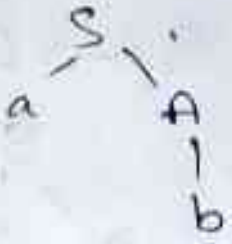
Sol $L = \{ab, abab, ababab \ldots\}$

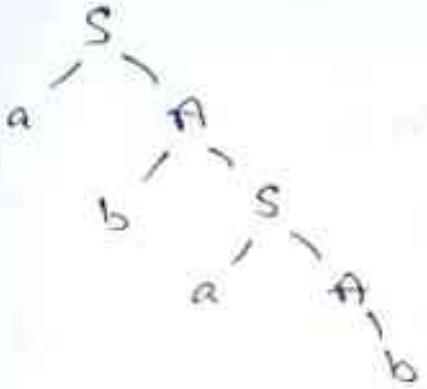
$S \rightarrow abS \mid ab$

(or)

$S \rightarrow aA$

$A \rightarrow bS \mid b$





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

Sol $L = \{abc, abcabc, abcabcabc \ldots\}$

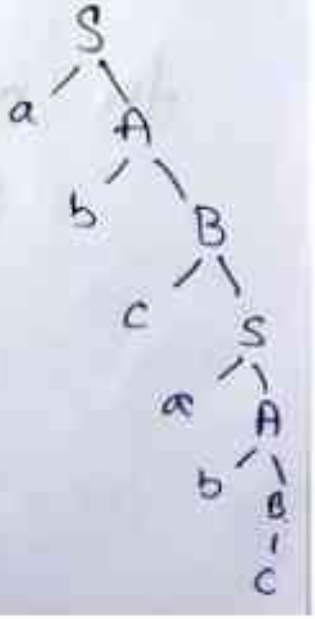
$S \rightarrow abcS \mid abc$

(or)

$S \rightarrow aA$

$A \rightarrow bB$

$B \rightarrow cS \mid c$

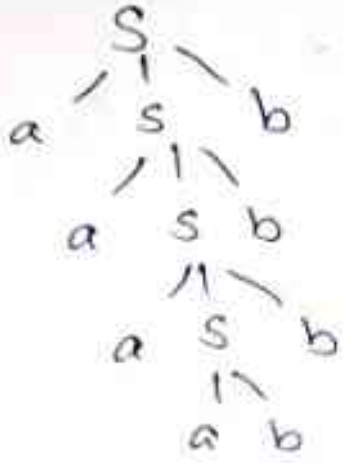


$\omega = abcabc$

⑥ $L = \{ a^n b^n \mid n \geq 1 \}$

Sol $L = \{ ab, aabb, aaabbb \ldots \}$

$S \rightarrow asb \mid ab$



aaaabbbb

⑦ $L = \{ ww^R \mid w \in (a,b) \}$

for even palindrome

$S \rightarrow aSa \mid bSb \mid \epsilon$

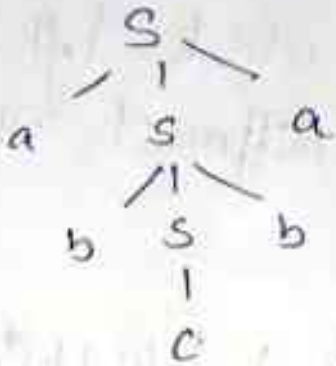
for odd palindrome

$S \rightarrow aSa \mid bSb \mid a \mid b.$

(8) $L = \{ wcw^R \mid w \in (a,b)^* \}$

Sol $L = \{ abcba \}$

$S \rightarrow aSa \mid bSb \mid c$



abcba //

(9) language having aa in between

Sol $R.E \rightarrow (a+b)^* aa (a+b)^*$

$S \rightarrow AaaA$

$A \rightarrow aA \mid bA \mid a \mid b$

(10) language having atleast two a's

Sol $R.E \rightarrow (a+b)^* a (a+b)^* a (a+b)^*$

$S \rightarrow AaAaA$

$A \rightarrow aA \mid bA \mid a \mid b$

(11) Language Containing Exactly two a's.

Sol

R.E -> $b^* a b^* a b^*$

S -> AaAaA

A -> bA|b

(12) Language in which left most Symbol is different than right most.

Sol

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

S -> aAb | bAa

A -> aA|bA|a|b

(13) Equal no of a's & equal no. of b's

Sol

S -> aSbS|bSaS|ε

(OR)

S -> aB|bA

A -> a|aS|bAA

B -> b|bS|aBB

(14) $L = \{ a^n b^m \mid n \neq m \}$

Sol

$n \neq m$

$n > m$: $S \to aAb$, $A \to aAb$, $A \to aA \mid a$

$m > n$: $S \to aBb$, $B \to aBb$, $B \to bB \mid b$

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

Sol

$S \to aSc$

$S \to bS \mid b$

(16) $L = \{ a^n b^n c^m \}$

Sol

$S \to abB$

$B \to cB \mid c$

* Push Down Automata [PDA]

→ Push down Automata is a finite Automata with extra memory called stack which helps PDA to recognize CFG.

→ PDA can be defined as.

$$M = (Q, \Sigma, \Gamma, q_0, z_0, \delta, F)$$

Q → set of states
Σ → set of input symbols
Γ → set of pushdown symbols
q_0 → Initial state
z_0 → Initial pushdown symbol
F → Final state
δ → Transition function

$$\left[\delta : Q \times \{\Sigma \cup \epsilon\} \times \Gamma \rightarrow Q \times \Gamma^*\right]$$

Eg:- $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

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

$\Sigma = \{a, b\}$ $F = \{q_2\}$

$\Gamma = \{a, b, z_0\}$

* Instantaneous Description (ID):-

→ Instantaneous Description (ID) is an informal notation of how a PDA computes a input string and make a decision that string is accepted or rejected.

→ An ID is a triple (q, w, α) where

1. q → current state
2. w → remaining inputs
3. α → stack content, top at the left.

→ $\vdash$ sign is called a turnstile notation.

Importance of stack:-

PDA acccepts the language in two ways

i) Acceptance by empty stack
ii) Acceptance by final state

* Push-Down Automata (PDA)

→ Design a PDA for the language wcw^R

Sol $L = \{ wcw^R \mid w \in (a+b)^+ \}$

$\delta(q_0, a, z_0) = (q_0, az_0)$

$\delta(q_0, b, z_0) = (q_0, bz_0)$

$\delta(q_0, a, a) = (q_0, aa)$

$\delta(q_0, b, a) = (q_0, ba)$

$\delta(q_0, a, b) = (q_0, ab)$

$\delta(q_0, b, b) = (q_0, bb)$

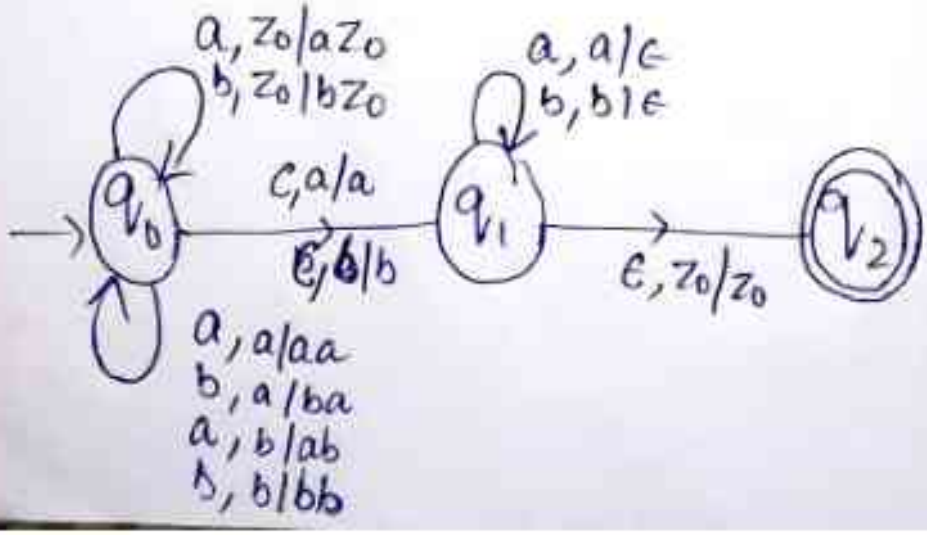
$\delta(q_0, c, a) = (q_1, a)$

$\delta(q_0, c, b) = (q_1, b)$

$\delta(q_1, a, a) = (q_1, \epsilon)$

$\delta(q_1, b, b) = (q_1, \epsilon)$

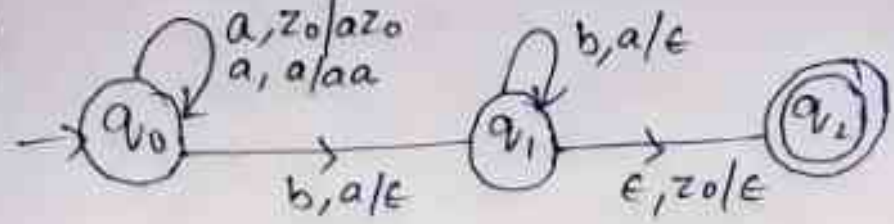
$\delta(q_1, \epsilon, z_0) = (q_2, z_0)$



→ Design a PDA for the language

$L = \{a^n b^n \mid n > 0\}$

Sol



$\delta(q_0, a, z_0) = (q_0, az_0)$

$\delta(q_0, a, a) = (q_0, aa)$

$\delta(q_0, b, a) = (q_1, \epsilon)$

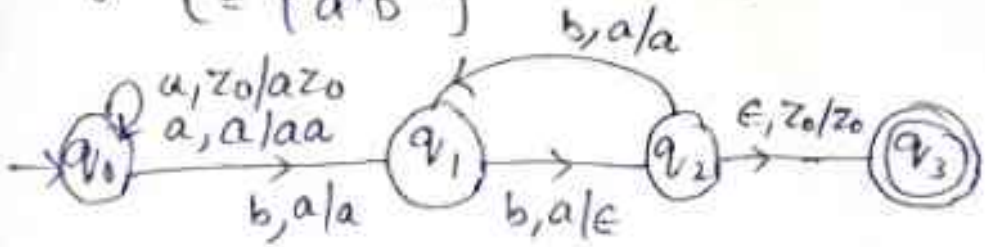
$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, z_0) = (q_2, z_0)$

→ Design a PDA for the language

$L = \{a^n b^{2n}\}$

Sol



$\delta(q_0, a, z_0) = (q_0, az_0)$

$\delta(q_0, a, a) = (q_0, aa)$

$\delta(q_0, b, a) = (q_1, a)$

$\delta(q_1, b, a) = (q_2, \epsilon)$

$\delta(q_2, b, a) = (q_1, a)$

$\delta(q_2, \epsilon, z_0) = (q_3, z_0)$

* PDA to CFG

Rules :-

1. S productions are given by.

$$S \rightarrow [q_0 z_0 q]$$

for every $q \in Q$.

2. For every popping move

$\delta(q, a, z) = (q', \epsilon)$ induces

$[q \; z \; q'] \Rightarrow a$

3. For every push move

$\delta(q, a, z) = (q_1, z_1 z_2 z_3 \ldots z_n)$

induces many productions

$$[q \; z \; q'] \rightarrow a [q_1 z_1 q_2][q_2 z_2 q_3] \ldots [q_m z_n q']$$

where $q', q_1, q_2 \ldots q_m$ be any state in Q

Convert the PDA to CFG

1) i) $\delta(q_0, a, z) = (q_0, Az)$

ii) $\delta(q_0, a, A) = (q_0, A)$

iii) $\delta(q_0, b, A) = (q_1, \epsilon)$

iv) $\delta(q_1, \epsilon, z) = (q_2, \epsilon)$

$G_1 = \{q_{q_0,q_1,q_2}\}$

~~for terminals(s)~~

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

$S \rightarrow [q_0 z q_0] \mid [q_0 z q_1] \mid [q_0 z q_2]$

transition (i)

$$\delta(q_0, a, z) = (q_0, Az)$$

$[q_0 z q_0] \rightarrow a [q_0 A q_0] [q_0 z q_0]$

$[q_0 z q_0] \rightarrow a [q_0 A q_1] [q_1 z q_0]$

$[q_0 z q_0] \rightarrow a [q_0 A q_2] [q_2 z q_0]$

$[q_0 z q_1] \rightarrow a [q_0 A q_0] [q_0 z q_1]$

$[q_0 z q_1] \rightarrow a [q_0 A q_1] [q_1 z q_1]$

$[q_0 z q_1] \rightarrow a [q_0 A q_2] [q_2 z q_1]$

$[q_0 z q_2] \rightarrow a [q_0 A q_0] [q_0 z q_2]$

$[q_0 z q_2] \rightarrow a [q_0 A q_1] [q_1 z q_2]$

$[q_0 z q_2] \rightarrow a [q_0 A q_2] [q_2 z q_2]$

transition (ii) $\delta(q_0, a, A) \rightarrow (q_0, A)$

$[q_0 A q_0] \rightarrow a [q_0 A q_0]$

$[q_0 A q_1] \rightarrow a [q_0 A q_1]$

$[q_0 A q_2] \rightarrow a [q_0 A q_2]$

Transition (iii)

$$\delta(q_0, b, A) = (q_1, \epsilon)$$

$$[q_0\ A\ q_1] \rightarrow b$$

Transition (iv)

$$\delta(q_1, \epsilon, z) = (q_2, \epsilon)$$

$$[q_1\ z\ q_2] \rightarrow \epsilon$$

* Convert the CFG to PDA.

1. Convert the CFG to PDA
2. The PDA will only have one state $\{q\}$
3. The Initial symbol of CFG will be the initial symbol of PDA
4. For non-terminal symbol add the following

$$\delta(q, \epsilon, A) = (q, a) \text{ when } A \rightarrow a$$

5. For each terminal symbol, add the following rule

$$\delta(q, a, a) = (q, \epsilon)$$

1) Given Grammary:-

$S \rightarrow bA|bB$

$A \rightarrow bAA|aS|a$

$B \rightarrow aBB|bS|b$

∴ Its in GNF

2. $\delta(q,\epsilon,S) = \{(q,bA),(q,aB)\}$

$\delta(q,\epsilon,A) = \{(q,bAA),(q,aS),(q,a)\}$

$\delta(q,\epsilon,B) = \{(q,aBB),(q,bS),(q,b)\}$

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

$\delta(q,b,b) = \{(q,\epsilon)\}$

Testing with some string.

$\delta(q,bbaa,S)$

$\delta(q,bbaa,bA)$

$\delta(q,bbaa,bA)$

$\delta(q,baa,A)$

$\delta(q,baa,bAA)$

$\delta(q,aa,AA)$

$\delta(q,aa,aA)$

$\delta(q,a,A)$

$\delta(q,a,a)$

(q,ϵ) Accepts

16/01/2021 UNIT :- 03

* Simplification / Minimization of CFG:-

- Removing Useless Symbols
  - Removing non generating Symbol
  - Removing unreachable Symbol
- Eliminate ∈ production
- Eliminate Unit production

> Useful Symbol :- If it is
  i) generating Symbol
  ii) Reachable

> ∈ - production :-
  Eg :- $A \rightarrow \epsilon$

> Unit production :-
  Eg :- $A \rightarrow B$

Q. Remove useless Symbols.

Grammar G : $S \to AB \mid a$

$A \to b$

Sol

$S \to AB \mid a$

$A \to b$

∵ B is non generating symbol

remove B

we get,

$S \to a$

$A \to b$

A is unreachable from start symbol

∴ remove A.

we get,

$S \to a$ //

Q. Remove useless symbol from grammar

$S \to aC \mid SB$

$A \to bSCa$

$B \to aSB \mid bBE$

$C \to aBC \mid ad.$

Sol

C directly derives terminals that is ad

Hence it is useful.

Since C is useful A & S is also useful as

$S \to aC$ & $A \to bSCa$

But B doesn't derive any string so it is non-generating

→ Remove productions with B we get

$S \to aC$

$A \to bSCa$

$C \to ad$

A is unreachable from start symbol S

∴ remove A production we get

$S \to aC$

$C \to ad$

//

Q. Eliminating ϵ (Null production)

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

→ Removing ϵ

$S \to aSa$

$S \to bSb \quad S \to \epsilon$

Substitute ϵ in place of S

$S \to a\epsilon a$, $S \to b\epsilon b$

$\Rightarrow S \to aa \quad S \to bb$

$\therefore S \to aSa \mid bSb \mid aa \mid bb$.

ii) $S \to XYX$

$X \to 0X \mid \epsilon$

$Y \to 1Y \mid \epsilon$

$X \to 0X \quad , X \to \epsilon$

$Y \to 1Y \quad , Y \to \epsilon$

$X \to 0X \mid 0$,,

$Y \to 1Y \mid 1$,

$S \to XYX \mid XY \mid XX \mid YX \mid X \mid Y$,,

iii) $A \to 0B1 \mid 1B0$

$B \to 0B \mid 1B \mid \epsilon$

Sol $B \to 0B \mid 1B$, $B \to \epsilon$

$B \to 0B \mid 1B \mid 0 \mid 1$ //

$A \to 0B1 \mid 1B0 \mid 01 \mid 10$ //

iv) $S \to a \mid Ab \mid aBa$

$A \to b \mid \epsilon$

$B \to b \mid A$

Sol $A \to \epsilon$

$S \to a \mid Ab \mid aBa \mid b$

$A \to b$

$B \to b \mid A$

Q. Eliminating Unit productions from grammar

i) $S \rightarrow 0A|1B|C$

$A \rightarrow 0S|00$

$B \rightarrow 1|A$

$C \rightarrow 01$

Sol Unit production in above grammar are

$S \rightarrow C$

$B \rightarrow A$

To eliminate Unit production we write

$S \rightarrow C$ as $S \rightarrow 01$

$B \rightarrow A$ as $B \rightarrow 0S|00$

Now, re-write the grammar

$S \rightarrow 0A|1B|01$

$A \rightarrow 0S|00$

$B \rightarrow 1|0S|00$

$C \rightarrow 01$

11

ii) $S \to AB$

$A \to a$

$B \to C \mid b$

$C \to D$

$D \to E \mid bC$

$E \to d \mid Ab$

Sol Unit productions are

$B \to C$

$C \to D$

$D \to E$

So

$\Rightarrow B \to C$

$\Rightarrow C \to D$

$\Rightarrow D \to E \mid bc$

$\Rightarrow E \to d \mid Ab \mid bc$

$\therefore B \to d \mid Ab$

$\Rightarrow C \to D \mid bc$

$D \to E$

$\therefore C \to d \mid Ab \mid bc$

$\Rightarrow D \to d \mid Ab$

Recursive the grammar we get.

$S \to AB$

$A \to a$

$B \to d|Ab|b|bc$

$C \to d|Ab|bc$

$D \to d|Ab|bc$

$E \to d|Ab$ //

iii) $S \to AA$

$A \to B|BB$

$B \to abB|b|bb$

8| Unit production is $A \to B$

$A \to B|BB$

$A \to abB|b|bb|BB$

$S \to AA$

$A \to abB|b|bb|BB$

$B \to abB|b|bb$ //

* Safe order or Correct order of Minimization.

i) Eliminate ∈ production
ii) Eliminate Unit production
iii) Eliminate useless Symbol

* Normalization of grammar

→ Grammar to appear in certain format

Normal Form

- Chomsky Normal Form (CNF)
- Greibach Normal Form (GNF)

Note:- Before Normalizing the grammar it must be Simplified / Minimized.

* CNF :-

Format :-

var → [var][var] (OR)

var → terminal

Q. Convert the following grammar into CNF

i) S → aaaaS
S → aaaa

Sol Given :- S → aaaaS
S → aaaa

let A → a (∵ its in CNF)
∵ A-terminal

S → AAAAS
P1

Consider S → AP1 (∵ its in CNF)
∵ Var → Var Var

Now P1 → AAAS
P2

P1 → AP2 (∵ its in CNF)

Lly P2 → AAS
P3

P2 → AP3 (∵ its in CNF)

P3 → AS (∵ its in CNF)

Similarly for

$S \rightarrow aaaa$

$S \rightarrow AAAA$

P_4

$S \rightarrow AP_4$ (Rts in CNF)

$P_4 \rightarrow AAA$

P_5

$P_4 \rightarrow AP_5$ (Rts in CNF)

$P_5 \rightarrow AA$ (Rts in CNF)

∴ The resultant grammar is

$S \rightarrow AP_1$

$P_1 \rightarrow AP_2$

$P_2 \rightarrow AP_3$

$P_3 \rightarrow AS$

$S \rightarrow AP_4$

$P_4 \rightarrow AP_5$

$P_5 \rightarrow AA$

$A \rightarrow a$

The above grammar is in CNF.

ii) $S \to aSa \mid bSb \mid a \mid b$

Sol Given:- $S \to aSa \mid bSb \mid a \mid b$

Let $A \to a$

$B \to b$

$S \to ASA \mid BSB \mid a \mid b$

(P_1 = SA, P_2 = SB)

$\to S \to AP_1 \mid BP_2 \mid a \mid b$

$P_1 \to SA$, $A \to a$

$P_2 \to SB$, $B \to b$

The above grammar is in CNF

iii) $S \to aSb \mid ab$

Sol Let $A \to a$

$B \to b$

$S \to ASB \mid AB$

(P_1 = SB)

$S \to AP_1 \mid AB$, $A \to a$, $B \to b$

$P_1 \to SB$ //

Is in CNF.

iv) $S \to aAS \mid a$

$A \to SbA \mid SS \mid ba$

Sol) $S \to a$ & $A \to SS$ are already in CNF

Let $R_1 \to a$

$S \to R_1 \underbrace{AS}_{P_1}$

$S \to R_1 P_1$ (In CNF)

$P_1 \to AS$ (Its In CNF)

Let $R_2 \to b$

$A \to S \underbrace{R_2 A}_{P_2} \mid R_2 R_1$

$A \to SP_2 \mid R_2 R_1$ (Its in CNF)

$P_2 \to R_2 A$ (It in CNF)

∴ The grammar is

$R_1 \to a$

$R_2 \to b$

$S \to R_1 P_1 \mid a$

$A \to SP_2 \mid R_2 R_1 \mid SS \mid ba$

$P_1 \to AS$, $P_2 \to R_2 A$ //

v) $S \to aB \mid bA$

$A \to bAA \mid aS \mid a$

$B \to aBB \mid bS \mid b$

Sol Let $R_1 \to a$, $R_2 \to b$ (Rhs in CNF)

$S \to R_1B \mid R_2A$ (Rhs in CNF)

$A \to R_2AA \mid R_1S \mid a$

P_1

$A \to R_2P_1 \mid R_1S \mid a$

$P_1 \to AA$ (Rhs in CNF)

$B \to R_1BB \mid R_2S \mid b$

P_2

$B \to R_1P_2 \mid R_2S \mid b$

$P_2 \to BB$ (Rhs in CNF)

∴ The resultant grammar is

$S \to R_1B \mid R_2A$

$A \to R_2P_1 \mid R_1S \mid a$

$B \to R_1P_2 \mid R_2S \mid b$

$P_1 \to AA$

$P_2 \to BB$

$R_1 \to a$

$R_2 \to b$

20/01/2021

* Minimize the grammar and Convert it into CNF

Q. S -> aAa | aBC
A -> aS | bD | ∈
B -> aBa | ∈ | b
C -> abb | DD
D -> aDa

Sol To minimize the grammar follow

Step i) Remove ∈ production
Step ii) Remove Unit production
Step iii) Remove Useless symbols.

1. Removing ∈ production

A -> aS | bD | ∈
A -> ∈

we get

S -> aAa | aBC | aa
A -> aS | bD
B -> aBa | ∈ | b
C -> abb | DD
D -> aDa

2. Remove Unit production

Unit production in grammar is

$B \to aBa \mid C \mid b$

$\Rightarrow B \to C$

we get

$B \to aBa \mid abb \mid DD \mid b$

Resultant grammar

$S \to aAa \mid aBC \mid aa$

$A \to aS \mid bD$

$B \to aBa \mid abb \mid DD \mid b$

$C \to abb \mid DD$

$D \to aDa$

3. Remove Useless Symbols.

S, B, C are useful as

- they generate terminal symbol

A is also useful as it derives S

But, D is non-generating. So

Remove D

we get, $S \to aAa \mid aBC \mid aa$

$A \to aS$

$B \to aBa \mid abb \mid b$

$C \to abb$

Now, Convert the minimize grammar in CNF

The minimized grammar is

$S \to aAa \mid aBC \mid aa$

$A \to aS$

$B \to aBa \mid abb \mid b$

$C \to abb$

Let $R_1 \to a$, $R_2 \to b$

(Rts in CNF)

$S \to R_1 \underbrace{AR_1}_{P_1} \mid R_1 \underbrace{BC}_{P_2} \mid R_1R_1$

$S \to R_1P_1 \mid R_1P_2 \mid R_1R_1$

$P_1 \to AR_1$

$P_2 \to BC$ (Rts in CNF)

$A \to R_1S$ (Rts in CNF)

$B \to R_1 \underbrace{BR_1}_{P_3} \mid R_1 \underbrace{R_2R_2}_{P_4} \mid b$

$B \to R_1P_3 \mid R_1P_4 \mid b$

$P_3 \to BR_1$

$P_4 \to R_2R_2$ (Rts in CNF)

$C \rightarrow R_1R_2R_2 \quad \because (P_4 \rightarrow R_2R_2)$

$C \rightarrow R_1P_4$ (2 bs in CNF)

The resultant grammar in CNF is

$S \rightarrow R_1P_1 \mid R_1P_2 \mid R_1R_1$

$P_1 \rightarrow AR_1$

$P_2 \rightarrow BC$

$A \rightarrow R_1S$

$B \rightarrow R_1P_3 \mid R_1P_4 \mid b$

$P_3 \rightarrow BR_1$

$P_4 \rightarrow R_2R_2$

$C \rightarrow R_1P_4$

$R_1 \rightarrow a$

$R_2 \rightarrow b$. //

* GNF :- Greibach normal Form

→ A CFG without ∈ is said to be in GNF if all of its Production is in the form

$A \rightarrow aV$

(or) where V = 0 or more no. of Variables

$A \rightarrow a$

* Eliminating left recursion

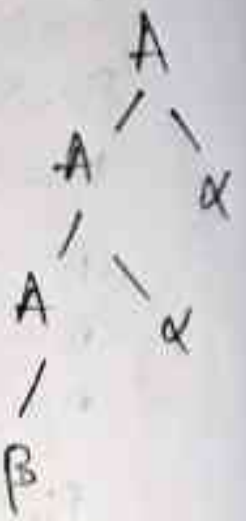
$\Rightarrow$ $A \rightarrow A\alpha \mid \beta$

Left recursion.

- for eliminate left recursion we get,

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \in$



In General,

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \beta_3 \ldots \mid \beta_n$

we get,

$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \beta_3 A' \mid \ldots \mid \beta_n A'$

$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \ldots \mid \alpha_m A' \mid \in$

* Convert the grammar into GNF

i) $S \to AA \mid a$

$A \to SS \mid b$

Sol Assume $S = A_1$ & $A = A_2$

$A_1 \to A_2A_2 \mid a$

$A_2 \to A_1A_1 \mid b$ (Here $A_2 \to A_1$, $2 > 1$)

$\to A_2 \to A_1A_1 \mid b$

Here replace A_1 with with we get. $A_1 \to A_2A_2 \mid a$

$A_2 \to A_2A_2A_1 \mid aA_1 \mid b$

($A_2A_2A_1$: α; aA_1: β_1; b: β_2)

↳ Its in the form of

$A \to A\alpha \mid \beta_1 \mid \beta_2$

for eliminating left recursion we get

$\Rightarrow A \to \beta_1 Z \mid \beta_2 Z \mid \beta_1 \mid \beta_2$

$\Rightarrow Z \to \alpha Z \mid \alpha$

Substituting values

$A_2 \to aA_1Z \mid bZ \mid aA_1 \mid b$ (Its in GNF)

$Z \to A_2A_1Z \mid A_2A_1$

Substitute A_2 in Z

$Z \rightarrow aA_1ZA_1Z \mid bZA_1Z \mid aA_1A_1Z \mid$
$bA_1Z \mid aA_1ZA_1 \mid bZA_1 \mid$
$aA_1A_1 \mid bA_1$ (βs in GNF)

Similarly

$\rightarrow A_1 \rightarrow A_2A_2 \mid a$

Replace A_2 with $A_2 \rightarrow aA_1Z \mid bZ \mid bA_1$

we get

$A_1 \rightarrow aA_1ZA_2 \mid bZA_2 \mid aA_1A_2 \mid bA_2 \mid a$

(βs in GNF).

∴ The resultant grammar in GNF is

$A_1 \rightarrow aA_1ZA_2 \mid bZA_2 \mid aA_1A_2 \mid bA_2 \mid a$

$A_2 \rightarrow aA_1Z \mid bZ \mid aA_1 \mid b$

$Z \rightarrow aA_1ZA_1Z \mid bZA_1Z \mid aA_1A_1Z \mid bA_1Z \mid aA_1ZA_1 \mid$
$bZA_1 \mid aA_1A_1 \mid bA_1$

Now Replace A_1 with S &
A_2 with A

we get

$S \rightarrow aszA \mid bzA \mid aSA \mid bA \mid a$

$A \rightarrow asz \mid bz \mid aS \mid b$

$Z \rightarrow aszsz \mid bzsz \mid assz \mid bsz \mid$
$aszs \mid bzs \mid ass \mid bs$

The above grammar is in GNF.

* Pumping Lemma for CFLs

-> If A is a context-free language, then A has a pumping length 'p' such that any String S, where $|S| \geq p$ may be divided into 5 pieces $S = uvxyz$ such that following conditions must be true.

i) $uv^ixy^iz \in A$, $i \geq 0$

ii) $|vy| > 0$

iii) $|vxy| \leq p$

Q1) Show that $L = \{a^n b^n c^n \mid n \geq 0\}$ is not context free

Ans Assume that $L = \{a^n b^n c^n \mid n \geq 0\}$ is context free language.

let the pumping length be $p = 3$

$$S = a^3 b^3 c^3$$

Case i:- $\underbrace{aaa}_{U}\underbrace{a}_{V}\underbrace{bb}_{X}\underbrace{b}_{Y}\underbrace{ccc}_{Z}$

UV^2XY^2Z gives ($\because i = 2$)

$aaaaabbbbbcccc \notin L$

$a^4b^4c^3 \notin L$ ✗

Case ii :- $\underbrace{aaabb}_{U\ V}\underbrace{b}_{X}\underbrace{cc}_{Y}\underbrace{c}_{Z}$

Case ii $U = aaa$ $V = abb$

$X = b$ $Y = cc$ $Z = c$

UV^2XY^2Z

$aaabbabbbccccc$

$\notin L$

Therefore, our assumption was wrong & the given language is not context free language.

* Closure properties of Context free languages.

Substitution :-

→ If L is a Context free language over an alphabet Σ and S is a substitution on Σ such that S(a) is a CFL for each a in Σ, then S(L) is a CFL.

→ The Context free languages are closed under the following operations

1. Union
2. Concatenation
3. Kleen closure (*) & positive closure (+)
4. Homomorphisms

Union:-

Let L_1 & L_2 be two CFL's. Then $L_1 \cup L_2$ is the language $S(L)$ where L is the language $\{1,2\}$ and S is the substitution defined by

$S(1) = L_1$ & $S(2) = L_2$

$L = \{1,2\}$

$S(1) = L_1$

$S(2) = L_2$

$S(L) = L_1 \cup L_2$

$S(L) = S(1) \cup S(2)$

$= L_1 \cup L_2$

Concatenation:-

Let L_1 & L_2 be two CFL's. Then $L_1 L_2$ is the language $S(L)$ where L is the language $\{1,2\}$ and S is the substitution defined

$S(1) = L_1$ $S(2) = L_2$

$S(L) = L_1 L_2$

3. Kleen closure & positive closure

If L_1 is a context free language, its kleen closure $(L_1)^*$ will also be context free.

Language = $\phi \mid y^*$

$S(1) = L_1$

then $(L_1)^* = S(L)$

similarly,

If Language = $\phi \mid y^+$

then $L_1^+ = S(L)$

$L = \phi \mid y^+$

$S(1) = L_1$

$S(L) = L_1^+$

4. Homomorphism :-

Suppose L is a CFL over Alphabet Σ and h is a homomorphism on Σ. Let S be the substitution that replaces each symbol a in Σ by the language consisting of the one string that is $h(a)$ ie

$S(a) = \phi h(a) y^*$ $\forall$ a in Σ. Then

$h(L) = S(L)$.

* CFL are not closed under intersection and Complementation.

* Decision properties of Context free language

1) Emptiness test :- Generating test,
   Reachability test

2) Membership test :- PDA acceptance

3) Finiteness test

4) Definiteness test

* CYK Algorithm (Unit 3)

→ It is a parsing algorithm for Cintex free grammar

→ In order to appy CYK Algorithm to a grammar, it must be in chomsky normal Foorm.

→ It Uses dynamic programming algorithm to tell wheather a string is in language of a grammar or not.

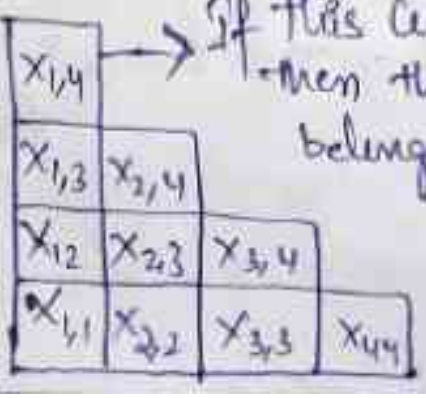
→ Time Complexity : $O(n^3.|G|)$
Space Complexity : $O(n^2)$

→ CYK :- Cocke - Younger - Kasami

→ Also known as Membership Algorithm

→ It Uses triangular table.

for Example if $|w| = 4$



→ If this cantains Initial symbol then the given string belong to grammar otherwise it doesn't belong to given grammar

1) Consider the CFG

$S \rightarrow A_1A_2 \mid A_2A_3$

$A_1 \rightarrow A_1A_2 \mid A_2A_3$

$A_1 \rightarrow A_2A_1 \mid 0$

$A_2 \rightarrow A_3A_3 \mid 1$

$A_3 \rightarrow A_1A_2 \mid 0$

Test 10010 is member or not using CYK Algorithm

Sol $w = 10010$

$|w| = 5$

Initial state

A_1				
ϕ	A_1, S			
ϕ	A_2	ϕ		
A_1, S	A_2	A_1, S, A_3	A_1, S	
A_2	A_1, A_3	A_1, A_3	A_2	A_1, A_3
1	0	0	1	0

→ This is not S so 10010 ∉ above grammar

∴ $10010 \notin L(G)$.

10010 does not belongs to grammar.

$X_{1,2} = A_2 \cdot (A_1, A_3) = A_2A_1 \cup A_2A_3 = A_1, S$

$X_{2,3} = A_1A_3(A_1, A_3)$

$X_{3,4} = A_1A_1 \cup A_3A_1 \cup A_1A_3 \cup A_3A_3$

$X_{4,1} = A_2$

$X_{3,4} = (A_1, A_3) \cdot A_2$

$= A_1 \cdot A_2 \cup A_3A_2$

$= A_1, S, A_3$

$X_{4,5} = A_2(A_1, A_3)$

$= A_2A_1 \cup A_2A_3$

$= A_1, S$

$X_{1,3} = A_2A_2 \cup (A_1, S)(A_1, A_3)$

$= A_2A_2 \cup (A_1A_1 \cup A_1A_3 \cup SA_1 \cup SA_3)$

$= \phi$

$X_{2,4} = (A_1, A_3)(A_1, S, A_3) \cup A_2A_2$

$= A_1A_1 \cup A_1S \cup A_1A_3 \cup A_3A_1 \cup A_3S \cup A_3A_3 \cup A_2A_2$

$= A_2$

$X_{3,5} = (A_1, A_3)(A_1, S) \cup (A_1, S, A_3)(A_1, A_3)$

$= A_1A_1 \cup A_1S \cup A_3A_1 \cup A_3S \cup A_1A_1 \cup A_1A_3 \cup SA_1 \cup SA_3 \cup A_3A_1 \cup A_3A_3$

$= \phi$

(b)

$$X_{1,4} = A_2A_2 \cup (A_1,S)(A_1,S,A_3) \cup A_2 \cdot \phi$$

$$= A_2A_2 \cup A_1A_1 \cup SA_1 \cup A_1S \cup SS \cup A_1A_3 \cup SA_3 \cup A_2\phi$$

$$= \phi$$

$$X_{2,5} = (A_1,A_3)\phi \cup A_2(A_1,S) \cup A_2(A_1,A_3)$$

$$= \phi \cup A_2A_1 \cup A_2S \cup A_2A_1 \cup A_2A_3$$

$$= A_1, S$$

$$X_{1,5} = A_2(A_1,S) \cup (A_1,S)\phi \cup \phi(A_1,S) \cup \phi(A_1,A_3)$$

$$X_{1,5} = A_2A_1 \cup A_2S$$

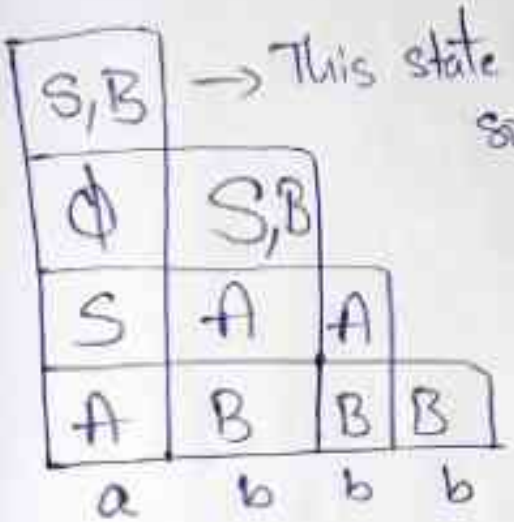
$$= A_1$$

(2) Consider the CFG

$S \rightarrow AB$
$A \rightarrow BB \mid a$
$B \rightarrow AB \mid b$

check abbb belongs to above grammar or not

$w = abbb \quad |w| = 4$



→ This state contain Initial Symbol so the string is accepted.

$\Rightarrow A \cdot (S,B) \cup SA \cup \Phi B$
$\Rightarrow AS \cup AB$
$\Rightarrow S, B$

∴ $w = abbb$ is a member of the given grammar.

UNIT:-04

* Turing Machine

→ Turing Machine is used to accept recursively enumerable languages. It consists of a tape of infinite length on which read and writes operations are performed.

→ It is represented using 7-tuples

$$TM = (Q, T, B, \Sigma, \delta, q_0, F)$$

Q → Finite set of states
T → Tape Alphabet
B → Blank Symbol
Σ → Input Alphabets
δ → Transition functions
q_0 → Initial State
F → set of Final states.

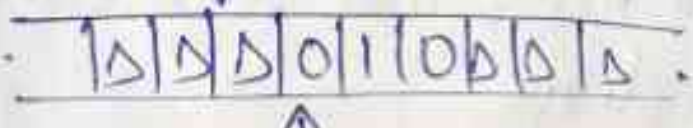
$$Q \times T \rightarrow Q \times T \times \{L, R\}$$

L → left
R → Right

→ A Turing machine is a hypothetical machine thought of by mathematician Alan Turing in 1936.

→ The Machine can Simulate Any Computer algorithm, no matter how complicated it is.

→ It contains a finite control, tape and tape head. Tape head moves either right or left by one square so that the machine can read and edit the symbol on a neighbouring square.



→ It accepts Recursively Enumerable languages (Generated by type-0 grammar)

→ It can be Expressed as a 7 tuple

$$M = (Q, T, B, \Sigma, \delta, q_0, F)$$

Q → No. of states
T → Tape alphabets
B → Blank symbol
Σ → Input Alphabet
δ : transition function
q_0 : Initial state
F : Final state

Transition function:-

$\delta: Q \times T \rightarrow Q \times T \times \{L, R\}$

* Acceptance of language by TM:-

→ The TM accepts the language by doing one of the following things

1) Halt and accept by entering into final state
2) Halt and reject which is possible if transition is not defined

(OR)

3) TM will never halt and enters into infinite loop.

* Programming techniques for TM:-

1. Storage in state:-

→ We can use the finite control not only to represent a position in the program of TM, but to hold a finite amount of data.

→ State

→ storage

finite control.

→ The control portion, q_0, or q_1, remembers what TM is doing

→ A data portion, which remembers the first symbol seen (i.e 0/1). The symbol B/Δ means no. symbol has been read.

2. Multiple tracks:-

→ The tape of TM can composed of several tracks. Each track can hold one symbol and the tape alphabet of TM consists of tuple with one component from each track.

In the above Tape, tape head contains the symbol [X, Y, Z].

→ It doesnot extend the working of TM.

3. Checking off Symbols:-

→ It is a useful trick for visualizing how a TM recognize language defined by repeated strings.

Eg:- $\{ ww \mid w \text{ in } \Sigma^* \}$

$\{ ww^R \mid w \text{ in } \Sigma^* \}$

→ It is also useful when length of substring must be compared

Eg:- $\{ a^n b^n \mid n \geq 1 \}$

4. Subroutines:-

→ TM's can be built from a collection of interacting component or subroutine.

→ A TM subroutine is a set of states that perform some useful process. This set of states includes start state and another states that temporarily has no moves and serve as return state to pass control.

→ The Call of a Subroutine occurs whenever there is transition to its Initial State

Eg:- TM for $f(a,b) = a * b$
where a & b are unary numbers

* Modifications of TM:-

→ Power of TM is defined by the no. of language it accepts.

1. TM with stay option:-

→ If instead of moving left or right on seeing input, the head could also stay at one position without moving.

$f: Q \times \Gamma \to Q \times \Gamma \times \{L, R, Stay\}$

Still the no. of languages accepted by TM will remain same.

2. TM with Semi-Infinite tape:-

TM has Infinite Input tape with extends in both directions Infinitely. If we restrict it to extend only in one direction (i.e semi Infinite) still the no. of languages accepted remains same

3. Offline TM:-

In Standard TM both the I/p & O/p are present on the tape. The head has authority to move across the Input and can change or modify Input. If we don't want to modify Input we provide the Input in seperate file, which cannot be changed.

If TM wants to modify the Input, the Input need to be copied on tape and the changes can be made by head but still the Input remains unchanged. By doing this modification still the number of languages accepted remains same.

4. Jumping TM:-

The Standard TM's head can move only one step, but in case of jumping TM it can move or jump to to any cell either left or right

$$f: Q \times X \to Q \times X \times \{L, R\} \times \{n\}$$

$n \to$ no. of states / steps to jump

but still the no. of languages accepted remains same.

5. Non erasing TM :-

→ In standard TM the Input symbol can be changed to blank, but if we remove the facility to change i/p symbol to blank then such type of TM is called non-erasing TM.

→ we can replace input symbol to any other except blank.

→ By doing this modification still the no. of languages accepted remains same.

6. Always writing TM :-

In Always writing TM it is Compulsory to modify the Input wherever we see we cannot leave as it is. But this will not help to increase the no of languages accepted.

→ Hence, we see that inspite of doing so many modification to TM still no. of languages accepted remains same. Therefore TM is most powerful machine.

* Turing machine as Enumerator:-

Enumerator = system + printer.

→ Enumeration is a process of generation and listing in finite amount of time

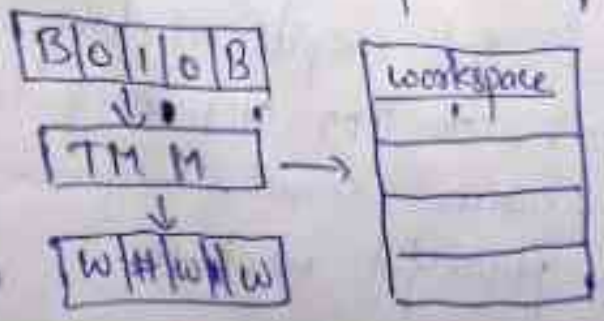
→ Enumeration process can be implemented by TM. So it is called as Enumerators.

→ The TM that implement enumerator consists of 3 tapes.

Tape1 → Input
Tape2 → workspace
Tape3 → output

→ The TM take Input from Input tape to generate string that will return on output tape.

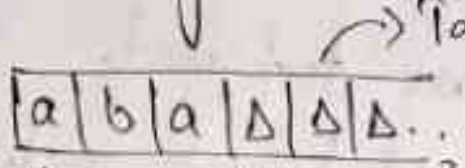
→ The string written on output cannot be modified. (Separated by #)

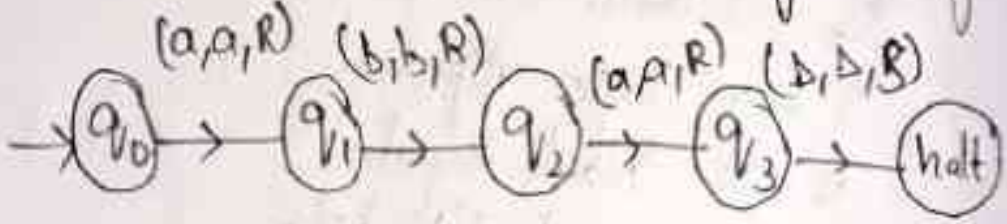


→ Turing Machine is a Universal Acceptor. It accepts all the grammar.

* Design a TM for the acceptance of string $w = aba$, $\Sigma = \{a, b\}$

Sol



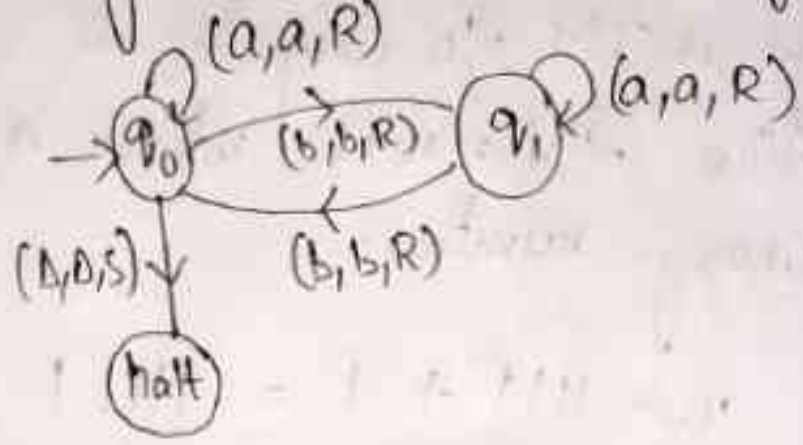


δ/T	a	b	Δ
q_0	(q_1, a, R)	–	–
q_1	–	(q_2, b, R)	–
q_2	(q_3, a, R)	–	–
q_3	–	–	$(halt, \Delta, S)$
halt	–	–	–

Note:- The Entries in transition tables can be defined or left undefined.

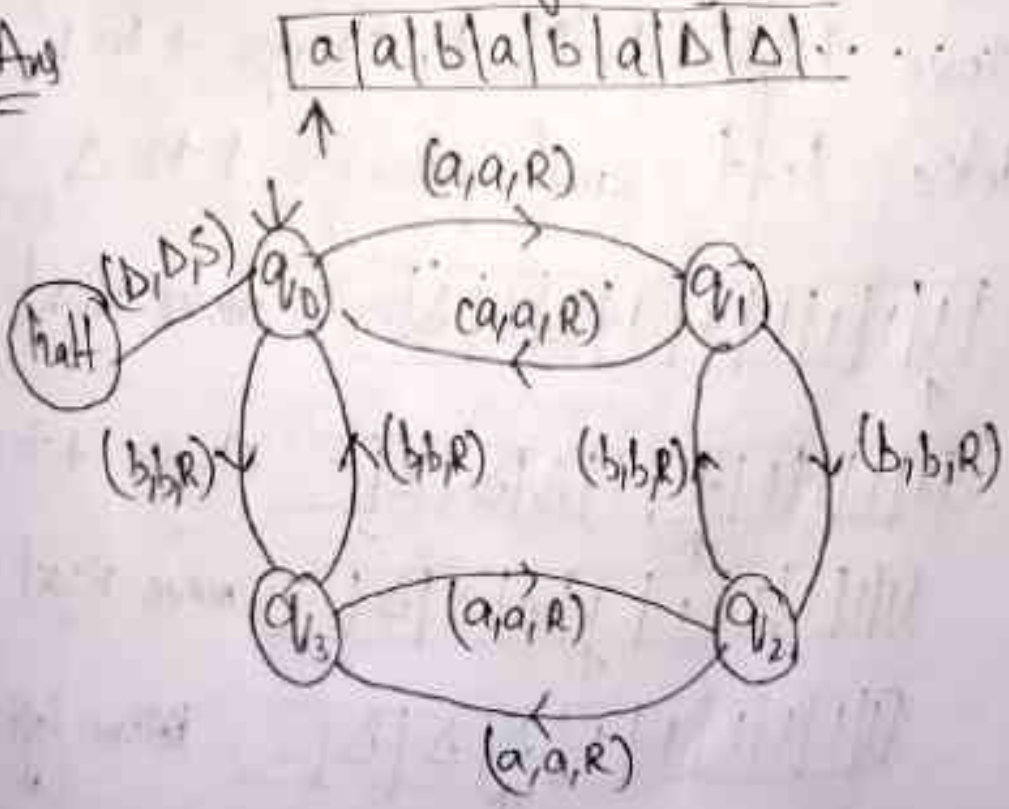
* Design a TM for accepting any no. of a's and even no. of b's

Ans =



* Design a TM for even no. of a's & even no. of b's

Ans =



* Turing machine

→ Function Computation :-

A TM computes mathematical computations.

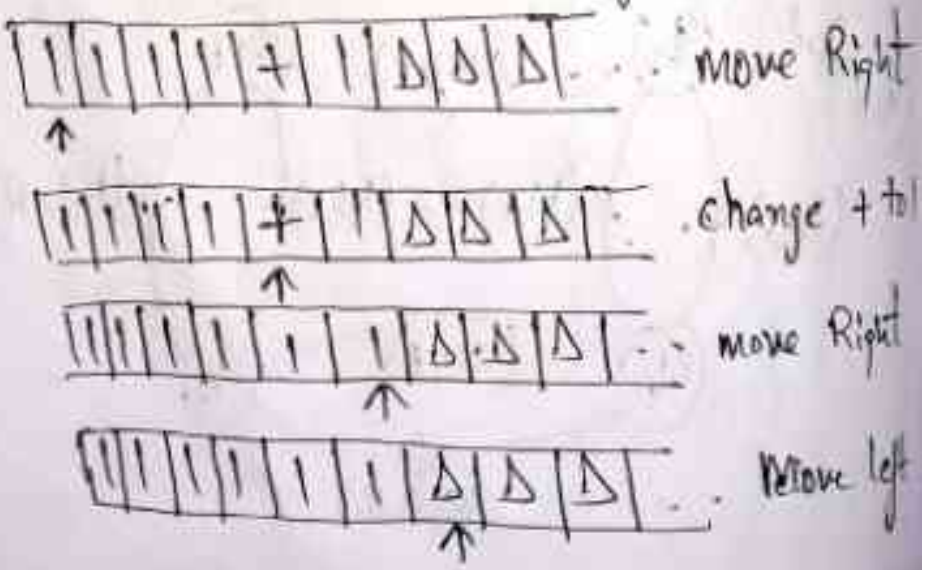
* Design a TM to compute a function $f(n) = n+1$ where n is Unary number.

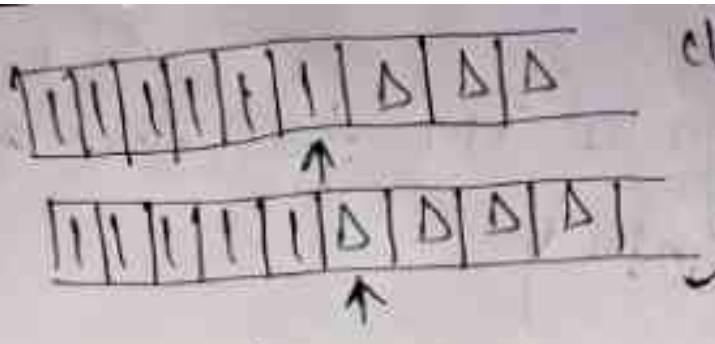
Ans :-

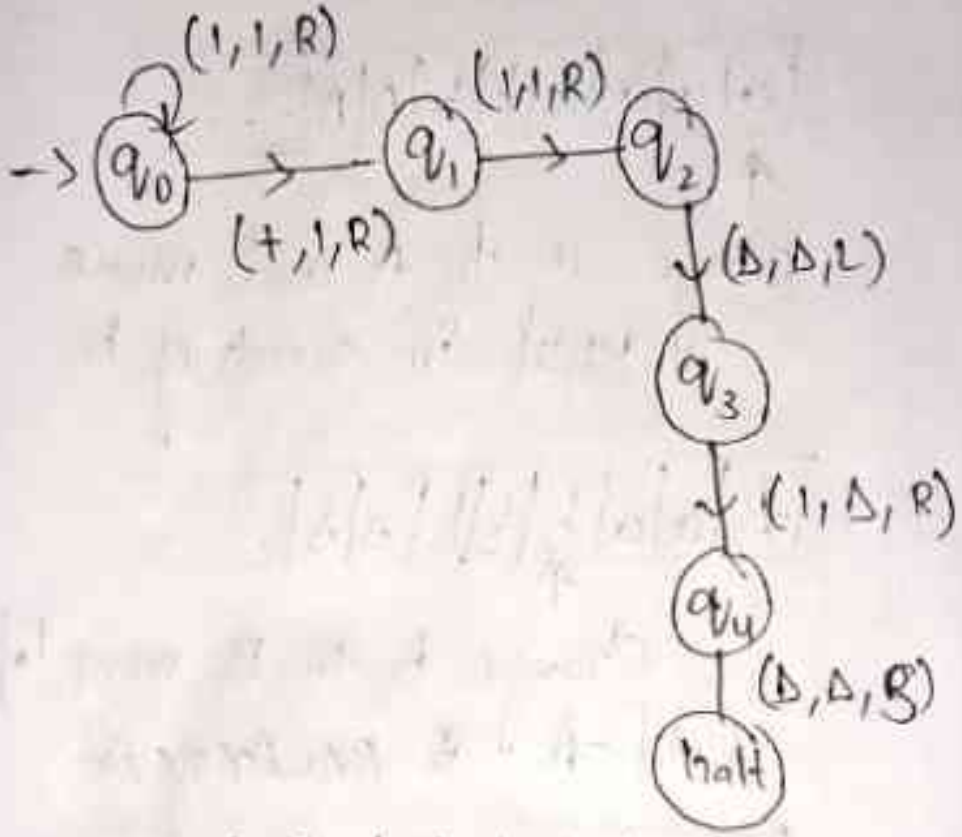
$$w = 1111 + 1 = 11111$$

i) Move Right Until +, change + to 1

ii) Take left and change 1 to Δ



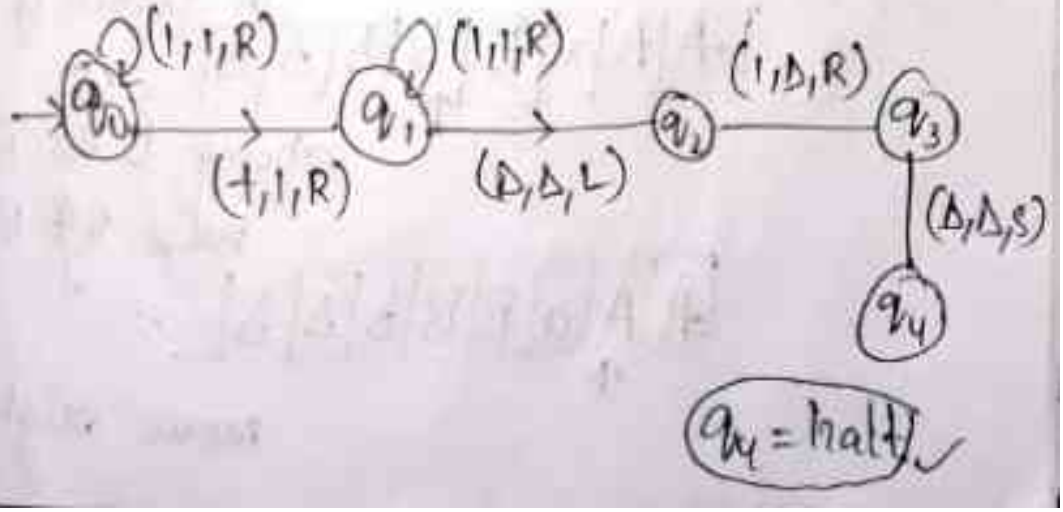




* Design a TM to Compute addition of two Unary no's $f(n) = x + y$.

Ans Let $x = 2$, $y = 3$

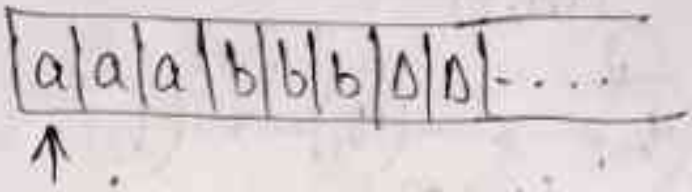
$f(n) = 11 + 111$



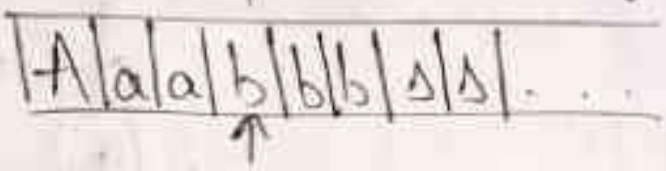
q_4 = halt

* Design a TM for the language $L = \{a^m b^n\}$

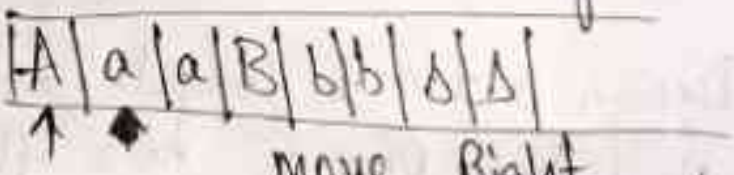
Ans



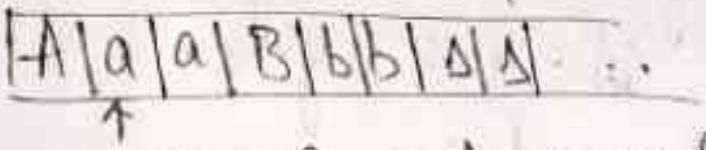
change a to A and move Right in search of b



Change b to B move left Until A the Boundry of a



move Right



a → A move R in search of b

chang b to B move left Until A

move right

| A | A | a | B | B | b | Δ | Δ | . . .

change a to A move R in search of B

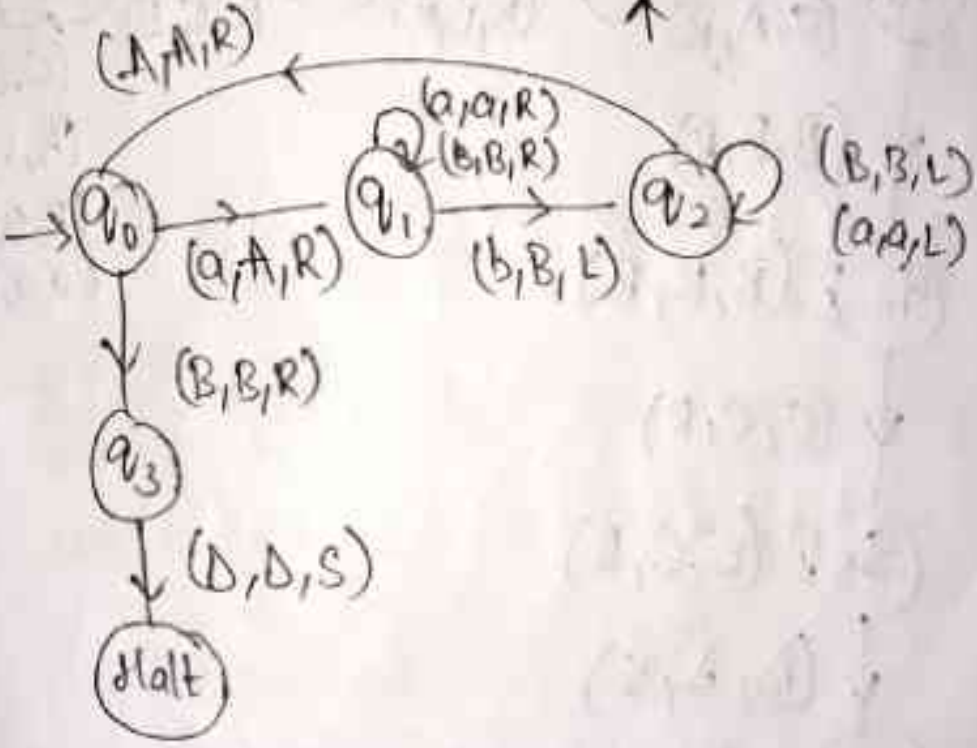
| A | A | A | B | B | b | Δ | Δ | . . .

change b → B move L in search of A

| A | A | A | B | B | B | Δ | Δ | . . .

move Right

| A | A | A | B | B | B | Δ | Δ | Δ



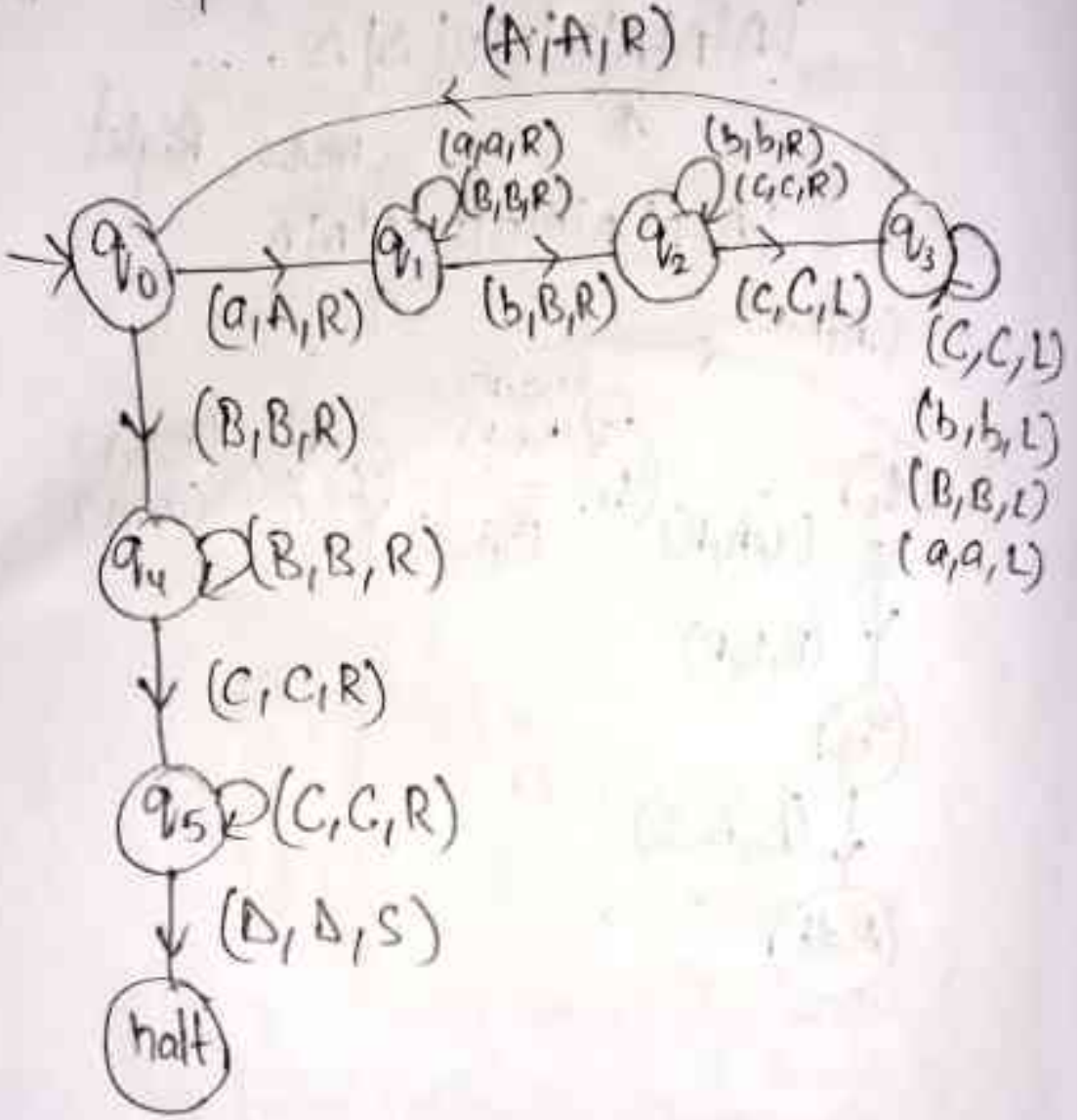
* Design a TM for the language

$$L = \{ a^n b^n c^n \mid n \geq 1 \}$$

Ans:-

a	a	a	b	b	b	c	c	c	Δ	Δ	Δ	...

↑

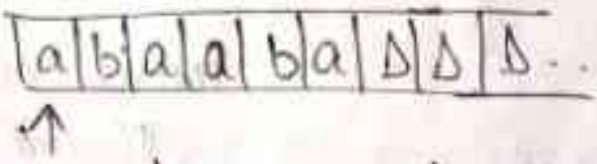


Transition table.

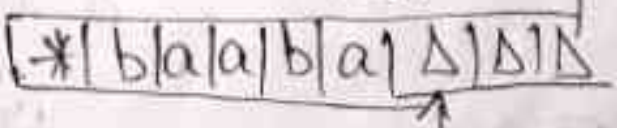
	a	b	c	A	B	C	Δ
q_0	(q_1, A, R)				(q_4, B, R)		
q_1	(q_1, a, R)	(q_2, B, R)			(q_1, B, R)		
q_2		(q_2, b, R)	(q_3, c, R)			(q_3, C, L)	
q_3	(q_3, a, L)	(q_3, b, L)		(q_0, A, R)	(q_1, B, L)	(q_3, C, L)	
q_4					(q_4, B, R)	(q_5, C, R)	
q_5						(q_5, C, R)	$(halt, \Delta, S)$
halt							$(halt, \Delta, S)$

* Design a TM for even palindrome

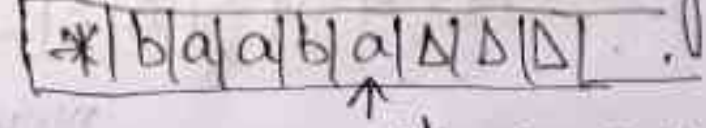
Sol $w = abaaba$ (let)



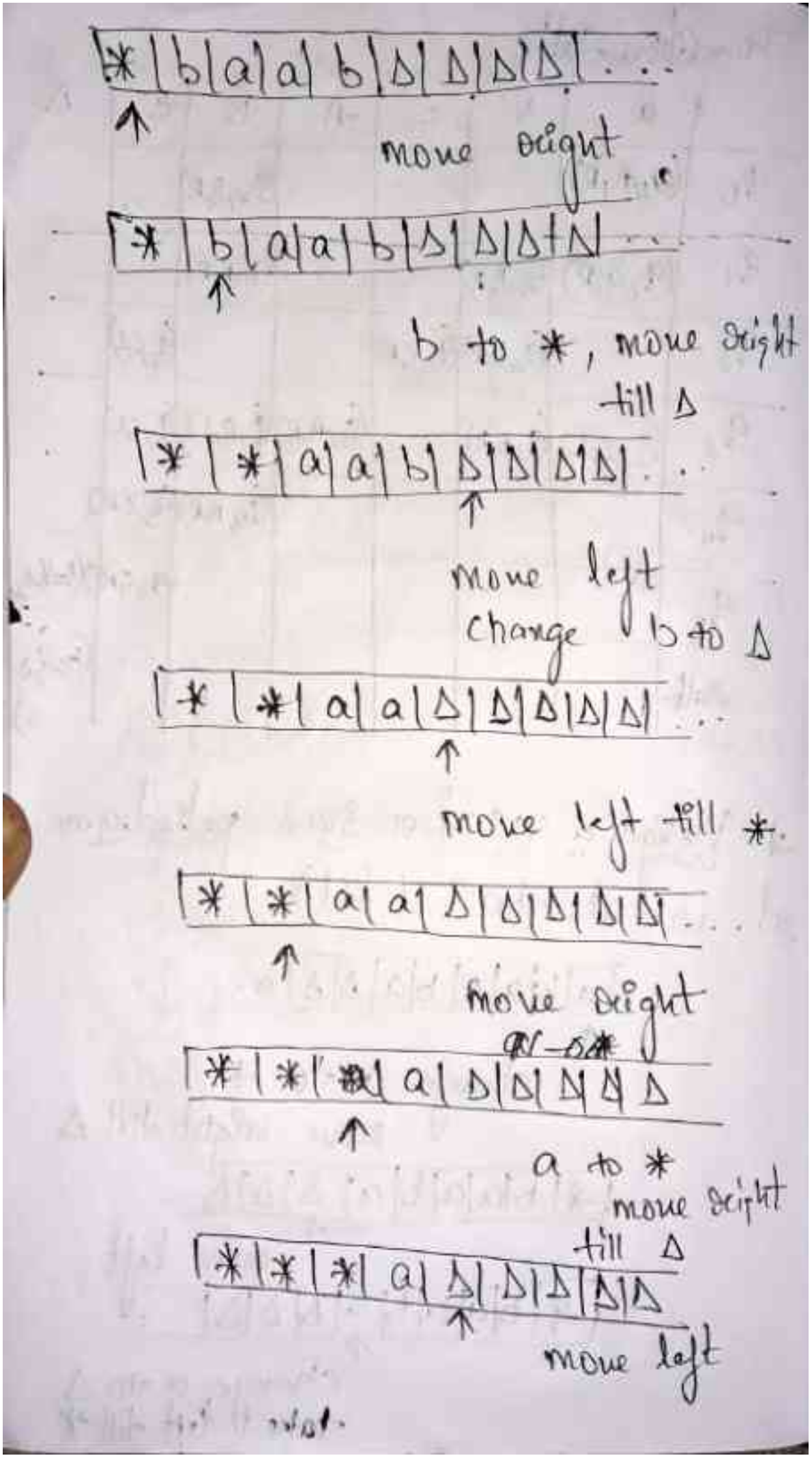
change a to *
move right till Δ

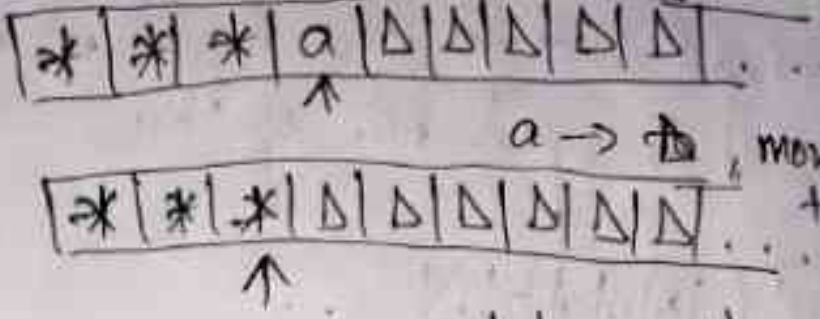


move left



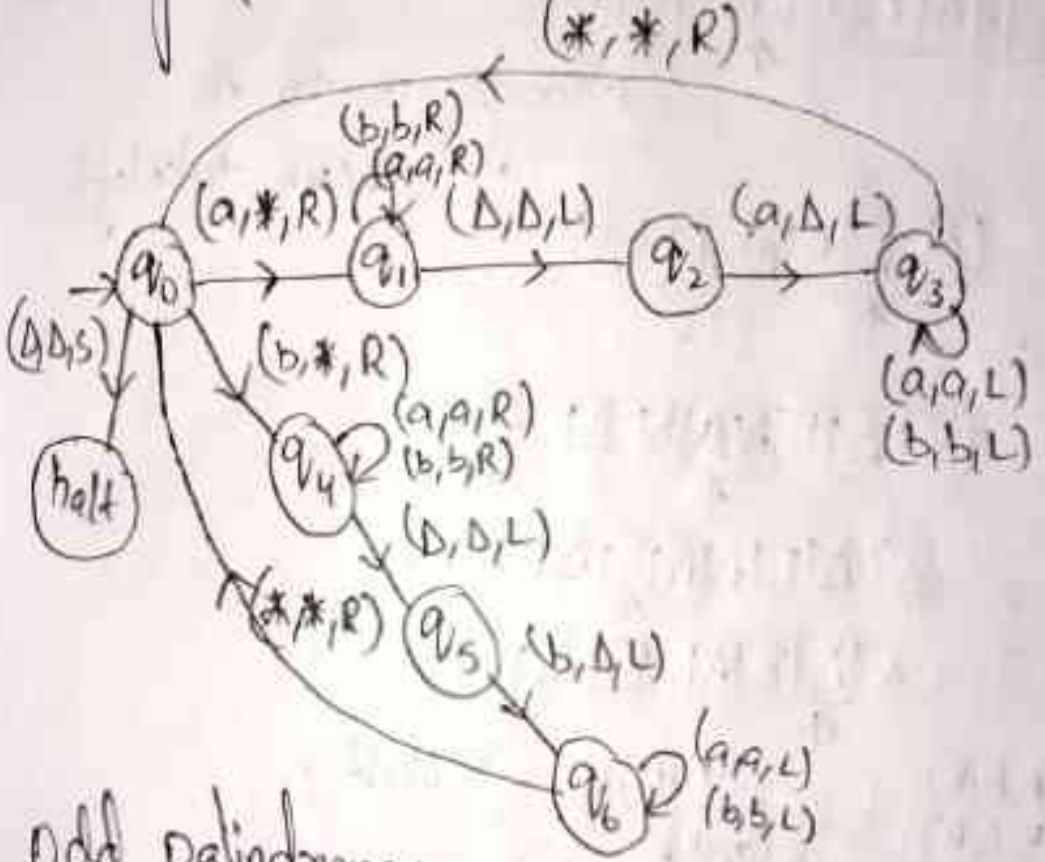
change a to Δ
take left till *


* | b | a | a | b | Δ | Δ | Δ | Δ | . . .
move right
* | b | a | a | b | Δ | Δ | Δ | Δ | . . .
b to *, move right till Δ
* | * | a | a | b | Δ | Δ | Δ | Δ | . .
move left
change b to Δ
* | * | a | a | Δ | Δ | Δ | Δ | Δ | . .
move left till *.
* | * | a | a | Δ | Δ | Δ | Δ | Δ | . .
move right
* | * | a | Δ | Δ | Δ | Δ | Δ
a to *
move right till Δ
* | * | * | a | Δ | Δ | Δ | Δ | Δ
move left

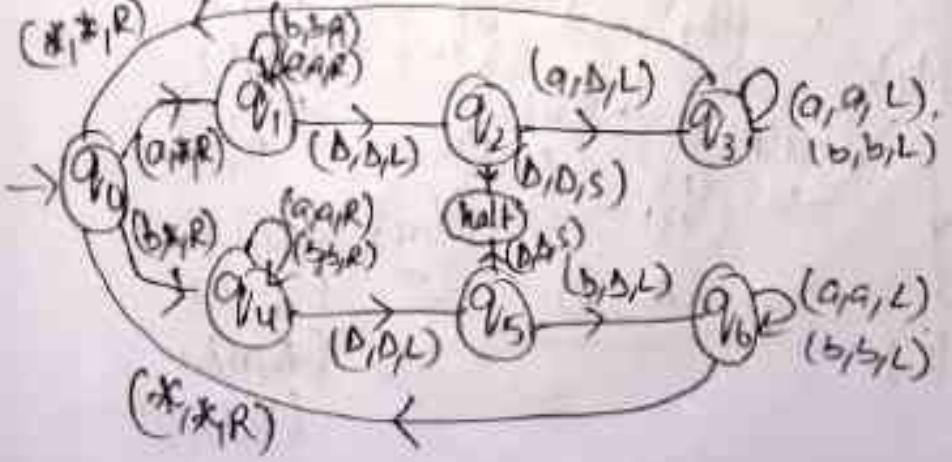


-take right & halt.

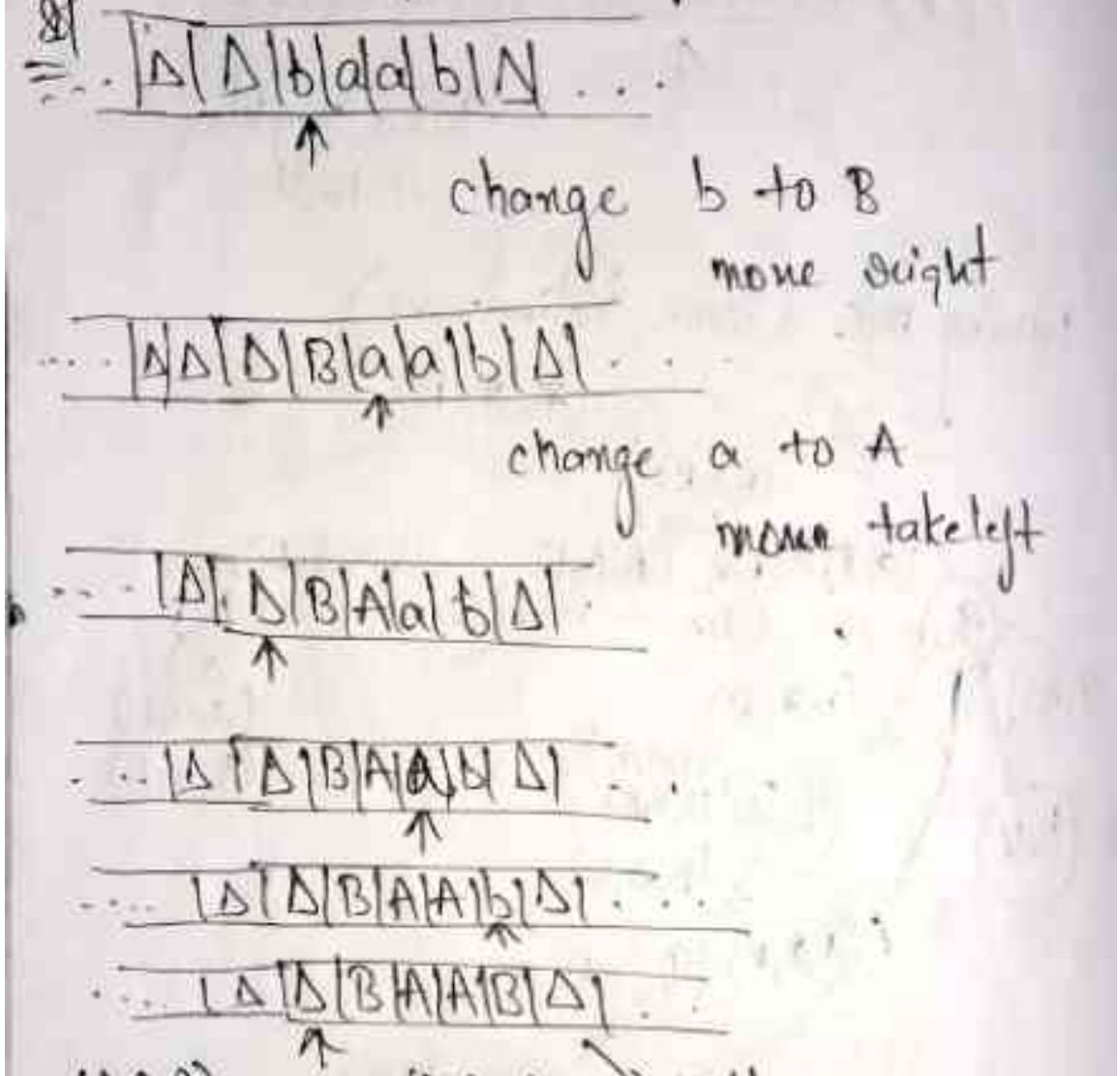
Turing m/c (Even Palindrome)

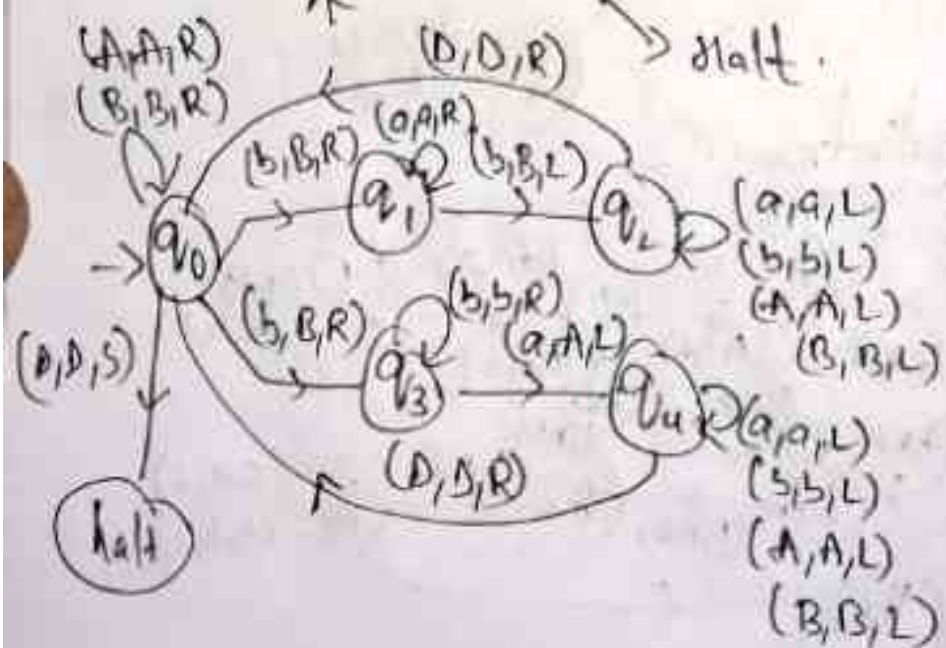


Odd palindrome :-



* Design a TM for Equal no. of a's & Equal no. of b's





* Design a Turing machine for balanced Parenthesis.

Ans

Δ Δ (( )) Δ Δ

Δ Δ ( ) ( ) Δ Δ

Take Δ Δ ( ) ( ) Δ Δ

↑

Δ ( ) ( ) Δ Δ

↑

move right in Search of closing ')'

Δ ( ) ( ) Δ Δ

↑

Mark ')' by * move left in search of '('

Δ (* ( ) Δ Δ

↑

Mark (with '*' move right

Δ * * ( ) Δ Δ

↑

move left by Marking) - *

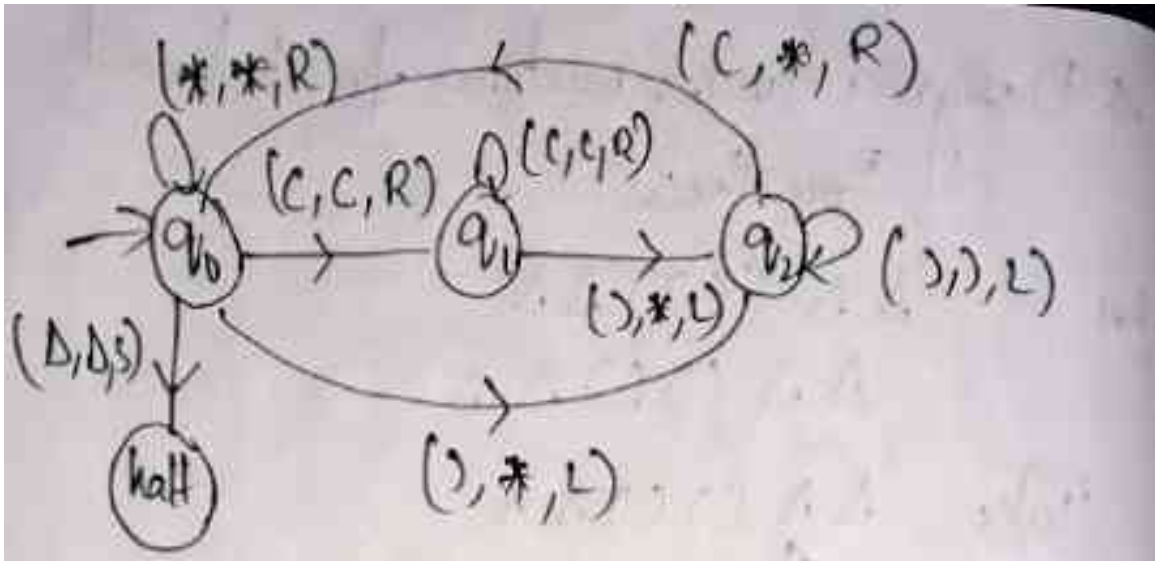
Δ * * (* Δ Δ

↑

(— *

Δ * * * * Δ

← ↑ →



* Turing machine for reverse of a string

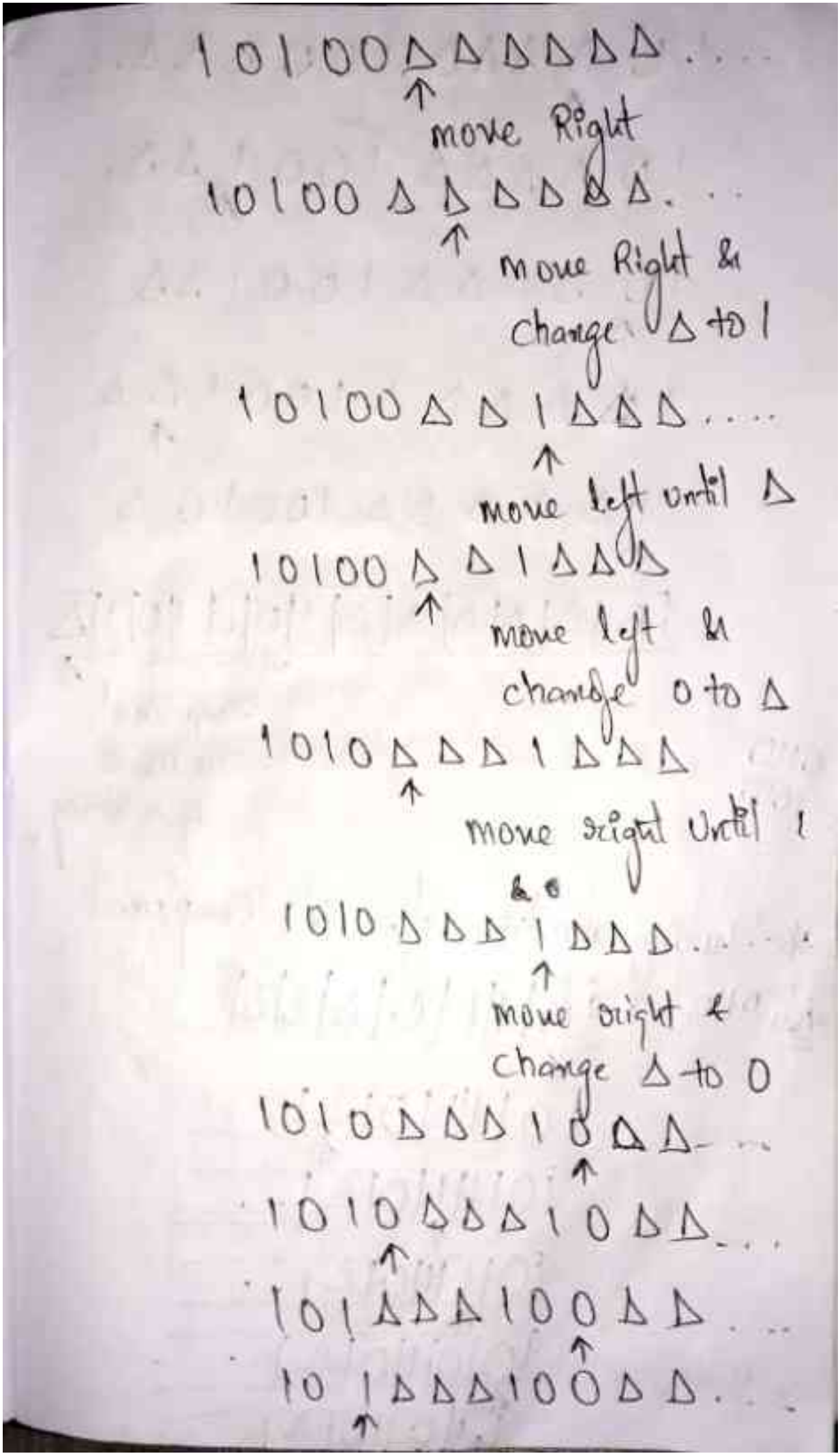
$w = 101001$

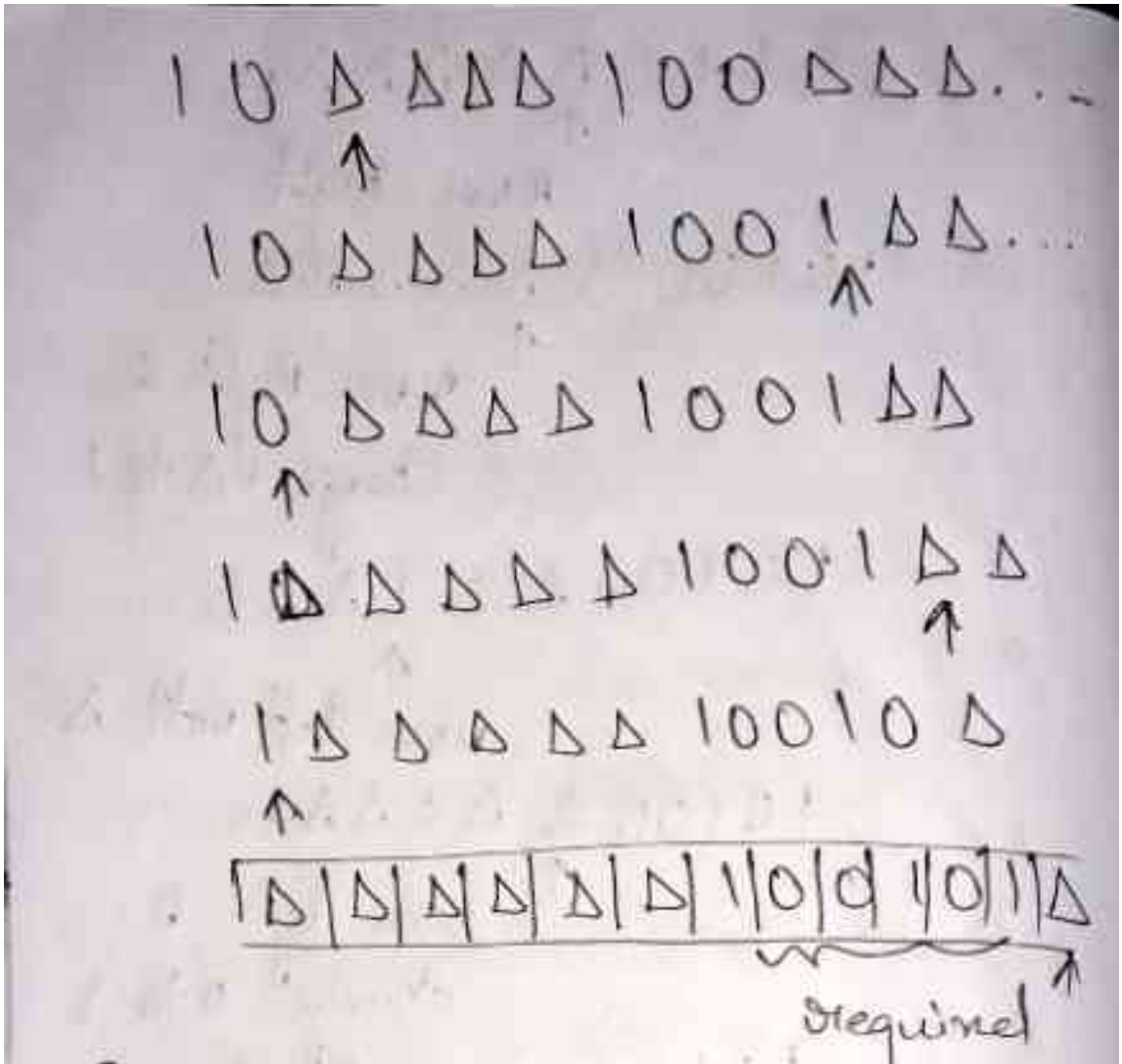
Note:- A TM can act as a transducer i.e in one position of the tape input is given and in another position output is given out.

Move right until Δ

1 0 1 0 0 1 Δ Δ Δ Δ Δ Δ . . . .

Move left change 1 to Δ



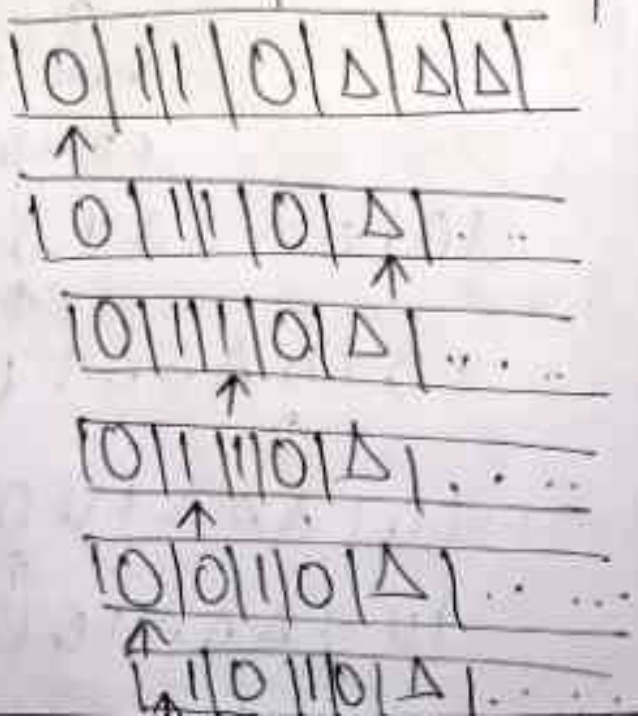


required reverse of a string

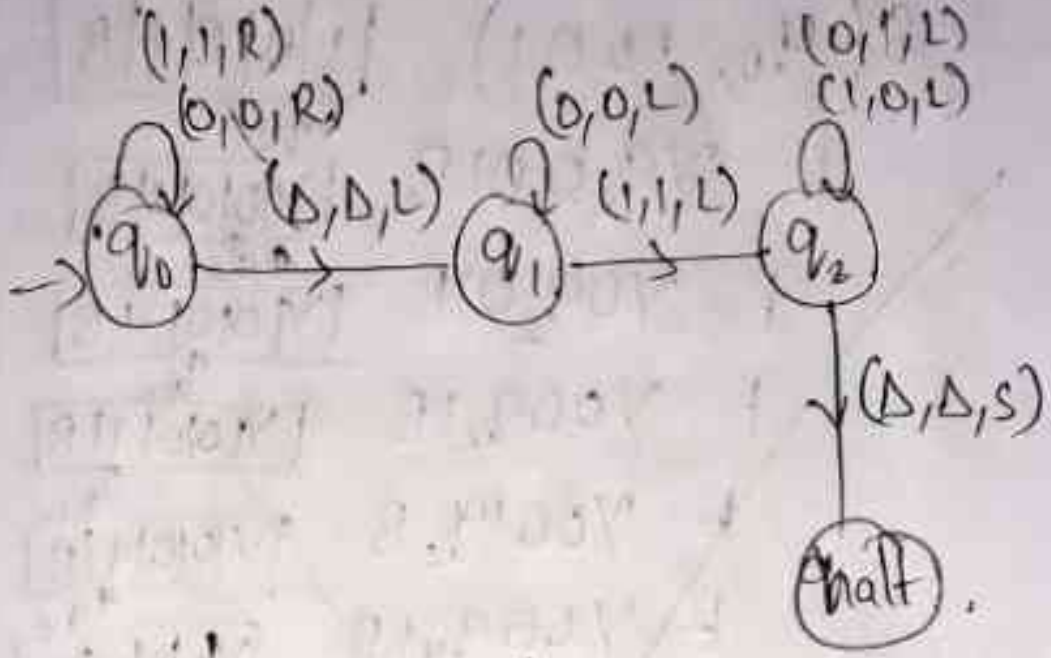
0110
1010

* Turing machine for 2's Complement

Ans: 0110



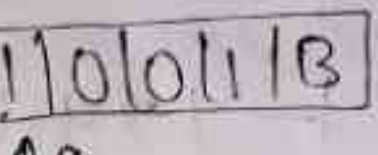
w=1010 ✓



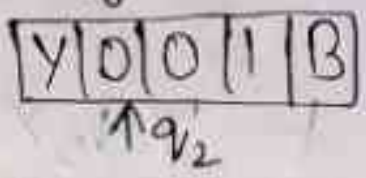
* Show that the string 1001 is accepted by given Turing machine

	0	1	X	Y	B
q_0	(q_1, X, R)	(q_2, Y, R)	(q_6, X, R)	(q_6, Y, R)	(q_6, B, X)
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_3, X, L)	(q_3, Y, L)	(q_3, B, L)
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	(q_4, X, L)	(q_4, Y, L)	(q_4, B, L)
q_3	(q_5, X, L)		(q_6, X, R)	(q_6, Y, R)	
q_4		(q_5, Y, L)	(q_6, X, R)	(q_6, Y, R)	
q_5	$(q_5, 0, L)$	$(q_5, 1, L)$	(q_0, X, R)	(q_0, Y, R)	
q_6					

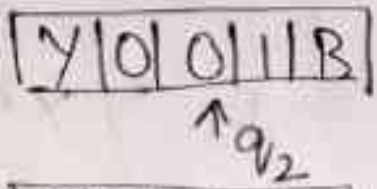
11) $\delta(q_0, 1, 001)$



$\vdash Yq_2001B$



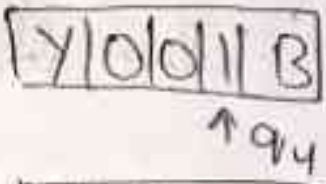
$\vdash Y0q_201B$



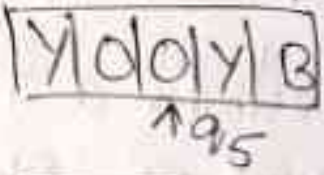
$\vdash Y00q_21B$

$\vdash Y001q_2B$

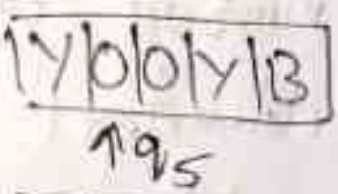
$\vdash Y00q_41B$



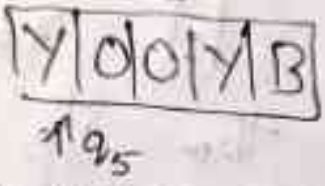
$\vdash Y0q_50YB$



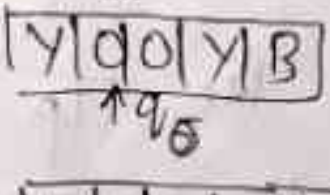
$\vdash Yq_500YB$



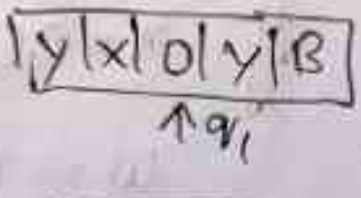
$\vdash q_5Y00YB$



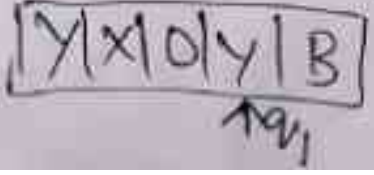
$\vdash Yq_000YB$



$\vdash YXq_10YB$



$\vdash YX0q_1YB$



$\vdash Y X q_3 0 Y B$ | Y | X | 0 | Y | B | ↑ q_3

$\vdash Y q_5 X X Y B$ | Y | X | X | Y | B | ↑ q_5

$\vdash Y X q_0 X Y B$ | Y | X | X | Y | B | ↑ q_0

$\vdash Y X X (q_6 Y) B$ halt | Y | X | X | Y | B | ↑ q_6

∴ The string is Accepted.

* Construct a TM to multiply 2 unary numbers over $\{0,1\}^+$ seperated by delimeter 1

Sol: Let $x = 0^m$, $y = 0^n$

then i/p $0^m 1 0^n$

The o/p should be 0^{m+n}

If $x = 00$ $y = 0000$

$w = 0010000$

Assume that y ends with 1 which acts as delimeter for i/p string which followed by blanks.

00100001BBB...

x y.

The output will be

BBB0000000BBB

(∵ y is repeated x no. of times)

00100001BBBBB....

↑

Change 0 to B, move right Until 1

B0100001BBBBB...

↑

move right.

B0100001BBBBB...

↑

change 0 to x, move right Until 1

B01x0001BBBBB

↑

move right.

B 0 1 X 0 0 0 1 B B B B

change B to 0 and move left until X then move Right

B 0 1 X 0 0 0 1 0 B B B

change 0 to X, move right until 1

B 0 1 X X 0 0 1 0 B B B

move right until B

B 0 1 X X 0 0 1 0 B B B B

change B to 0 & move left until X

B 0 1 X X 0 0 1 0 0 B B B

move right

B 0 1 X X 0 0 1 0 0 B B B

Change 0 to X & move Right until B

B 0 1 X X X 0 1 0 0 B B B

Change B to 0, move left until X

B 0 1 x x x 0 1 0 0 0 B B

move right

B 0 1 x x x 0 1 0 0 0 B B

change 0 to x move right
Until B

B 0 1 x x x x 1 0 0 0 B B B

change B to 0 move Until x

B 0 1 x x x x 1 0 0 0 0 B B.

move right

B 0 1 x x x x 1 0 0 0 0 B B...

move left change x to 0
Until B

B 0 1 0 0 0 0 1 0 0 0 0 B B....

move right

B 0 1 0 0 0 0 1 0 0 0 0 B B....

change 0 to B, move right
Until 1

B B 1 0000 1 0000 BBB

move ↑ right

B B 1 0 0 0 0 1 0000 BBBB

↑

change 0 to x, move right

until B

B B 1 x 0 0 0 1 0000 B BBB

↑

change B to 0, move left

until x

B B 1 x 0 0 0 1 0000 0 B B B

↑

move right, change 0 to x

move right until B

B B 1 x x 0 0 1 0000 0 B B B

↑

change B to 0 move left

until x

B B 1 x x 0 0 1 0000 0 0 B B

↑

move right, change 0 to x

move right until B

B B 1 x x x 0 1 0000 0 0 B B...

↑

change B to 0 move left until X

BB1xxx01000000BBBB

move right, change 0 to X

move right until B

BB1xxxx10000000BBB

change B to 0 move left until X

BB1xxxx10000000BBB

move right

BB1xxxx100000000BBB

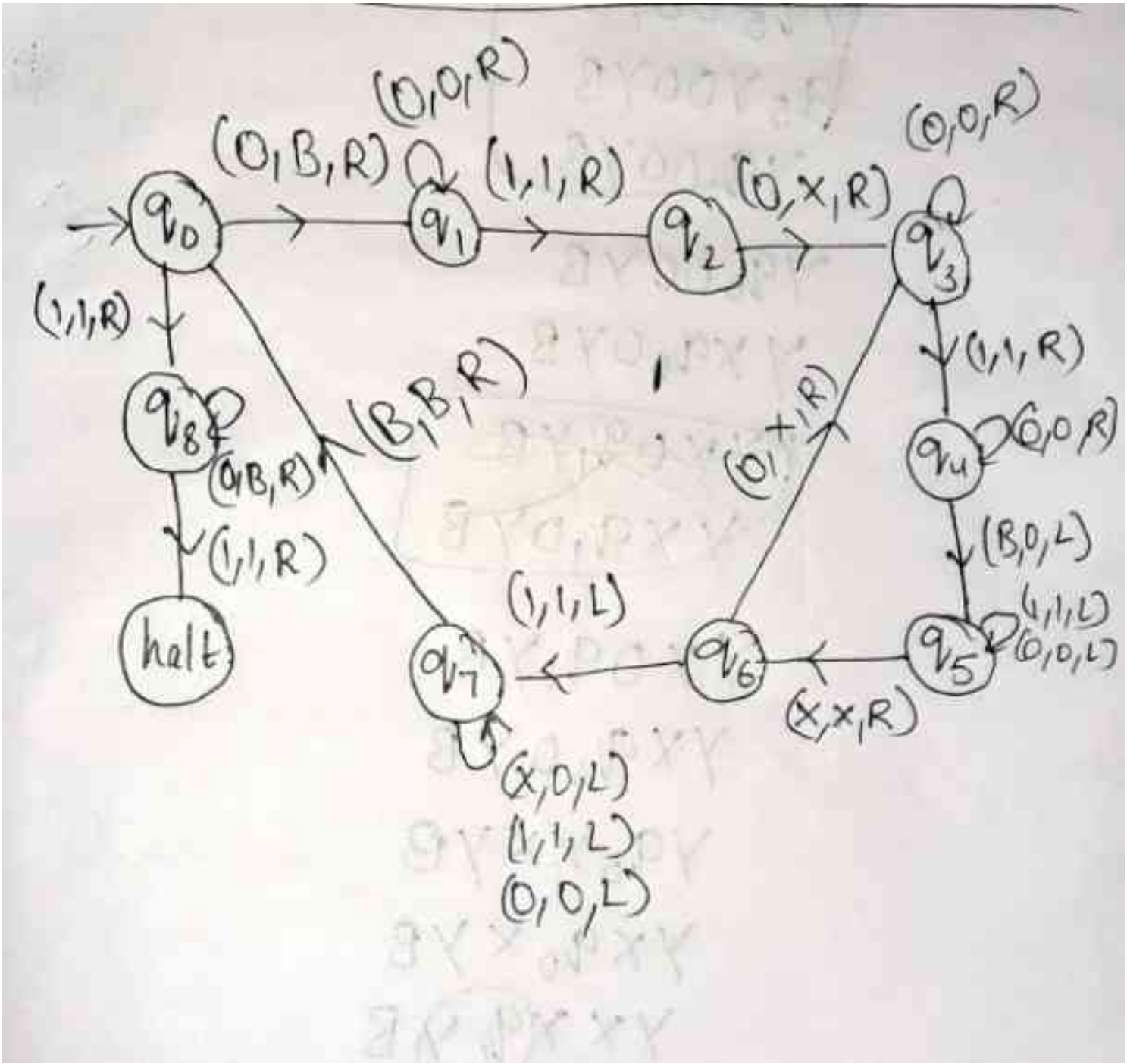
move left, change X to 0 until B.

BB1000011000000BBB

move right change all 0 to B until 1

BB1BBBB10000000BBB

Now the TM comes in halt state.



UNIT:-05:-

* Recursive and Recursively Enumerable language.

* Recursively Enumerable language:-[RE]

-> RE language or type-0 language are generated by type-0 grammar. An Recursively enumerable language can be accepted or recognize by Turing machine which means it will enter into final state for string of language and may or may not enter into rejecting state for strings which are not part of the language. It means TM can loop forever for those strings which are not part of the language.

-> RE languages are also called as Turing Recognizable language.

-> These are closed under all properties except difference & Complement

* Recursive language :-

-> A Recursive language (subset of RE) can be decided by Turing machine which means it will enter into final state for strings of language and rejecting states for strings which are not part of the language.

-> TM will always halt in this case.

-> Recursive language are also called as Turing decidable language.

-> These are closed under all properties except Homomorphism & Substitution.

* Closure properties of Recursive language.

1) UNION :- If L_1 and L_2 are two recursive languages then $L_1 \cup L_2$ will also be recursive

2. Concatenation:- If L_1 & L_2 are two recursive languages then $L_1.L_2$ will also be recursive.

3. Kleene closure:- If L_1 & L_2 are two recursive languages then $L_1 \cap L_2$ will also be recursive.

4. Intersection:- If L_1 & L_2 are two recursive languages then $L_1 \cap L_2$ and will also be recursive

5. Complement:- If L_1 is recursive language then $\Sigma^* - L_1$ will also be recursive.

* Decidable language:-

A language L is decidable if it is a recursive language. All decidable languages are recursive languages and vice versa

* Partially decidable language:-

A language L is partially decidable if 'L' is a recursively enumerable language.

* Undecidable language:-

-> A language is undecidable if it is not decidable

-> An undecidable language may sometime be partially decidable but not decidable

-> If a language is not even partially decidable, then there exists no Turing machine for that languages.

* Universal Turing machine :- (UTM)

-> A UTM is a turing machine that simulates an arbitrary turing machine on arbitrary input. The Universal machine essentially achieves this by reading both the description of the machine to be simulated as well as

input to that machine from its own tape.

The language

$A_{TM} = \{ \langle M, w \rangle \mid M$ is a Turing machine M accepts $w \}$

is Turing Recognizable.

If M accepts w : Algorithm will halt & accept

M rejects w : Algorithm will halt & reject

M loops on w : Algorithm will not halt.

→ The Universal Turing machine takes

Input : T = the description of some Turing machine

w = an input string for M

Actions:

→ Simulate M

→ It behaves just like T, would behave. That is

It may also accepts, reject or loop.

→ The Universal Turing machine can be Compared with general purpose Computers. Programs are given to Computers which simulates/run based on the inputs given and provides us the output/result. In the same way Universal Turing machine simulates given Turing machine over the string given.

→ However, the Turing machine have an Infinite tape to store data but the Computers doesn't have Infinite memory. This is the main difference between Universal Turing machine & general Purpose Computers.

→ Therefore, The Universal Turing machine is a recognizers (but not a decider) for

$A_{TM} = \{\langle M, w \rangle \mid M \text{ is a Turing machine and } M \text{ accepts } w\}$

* Undecidable Problem:-

→ Halting problem is an undecidable problem.

Halting problem:-

→ Given a program/algorithm it will ever halt or not.

→ Halting means that the program on certain input will accept it and halt or reject it and halt and it would never go into an infinite loop.

→ Can we have an algorithm that will tell that the given program will halt or not.

→ The answer is no we cannot design a generalized algorithm which can appropriately say that given a program will ever halt or not. The only way is to run the program checks whether it halt or not.

→ This is Undecidable problem because we cannot have an algorithm which tells us whether the program halts or not.

Proof by Contradiction:-

P.S: Can we design a machine which if given a program can say whether it halts or not.

Solution:-

Assume that we can design a machine $H(P, I)$ where H -> machine
P -> Program
I -> Input
that tells whether the program P either halts or not.

If we can design such a machine this allows us to write another program $C(x)$

```
H(P,I) { Halt or
         May Not Halt };

C(x) { if( H(x,x) == Halt)
         loop forever;
       else
         return;
     }
```

If we run C(x) on itself.
We get,

C(C)

H(C,C) == Halt

{ loop forever;
//It means it never halt
}

H(C,C) == Not Halt

{ Halt;
//It will never halt because of non-halting condition
}

→ Both condition is not halting for C(C) even we had assumed in the beginning that it will halt.
So this is the contradiction and we can say that our assumption was wrong.

→ This is how we proved that halting problem is undecidable.

* RICE Theorem:-

Every non-trivial (answer is not known) Problem on recursively Enumerable language is undecidable.

Eg:- If a language is recursively Enumerable, its Complement will be recursively enumerable or not is undecidable.

Reducibility & undecidability:-

language A is reducible to language B (represented as $A \leq B$) if there exists a function f which will convert string in A to strings in B as

$$\omega \in A \Leftrightarrow f(\omega) \in B.$$

Theorem 1: If $A \leq B$ is decidable then A is also decidable.

Theorem 2: If $A \leq B$ is undecidable then B is also undecidable

* PCP :- Post Correspondence problem

→ PCP is a popular undecidable problem that was introduced by Emil Leon in 1946.

→ In this problem we have N number of Dominos (tiles). The aim is to arrange tiles in such order that string made by Numerators is same as string made by Denominators.

→ lets assume two lists both containing N words, aim is to find the concatenation of these words in some sequence such that both lists yeild same result.

→ Eg:- Consider A = [aa, bb, abb] (1) (2) (3)

B = [aab, ba, b]

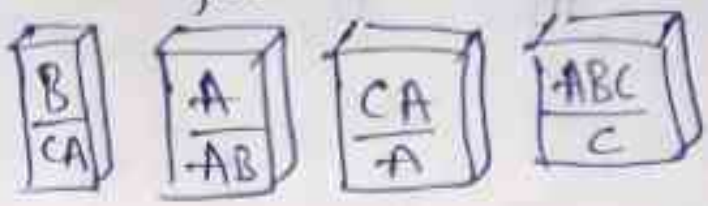
Now for sequence 1, 2, 1, 3 the first list yeilds aabbaaabb, the seand list yeilds the same string

So solution to PCP becomes 1, 2, 1, 3

--> This is represented in two forms

1) Dominos form
2) Tabular form

① Dominos-form



We need to find a sequence of dominos such that the top and bottom string are same.

② Tabular form

	Numerator	Denominator
1	B	CA
2	A	AB
3	CA	A
4	ABC	C

PCP sequence

Numerator: A B C A A A B C

Denominator: A B C A A A B C

Q)

	A	B
1	1	111
2	10111	10
3	10	0

A: $\frac{10111}{10}$ $\frac{1}{111}$ $\frac{1}{111}$ $\frac{10}{0}$

B:

Seq: 2 1 1 3 //

MPCP :-

Modified post Correspondence problem is just like PCP except that we specify both the set of tiles and also a special tile which matches for MPCP have to start with special tile.

Special tile will be the first tile given in question.

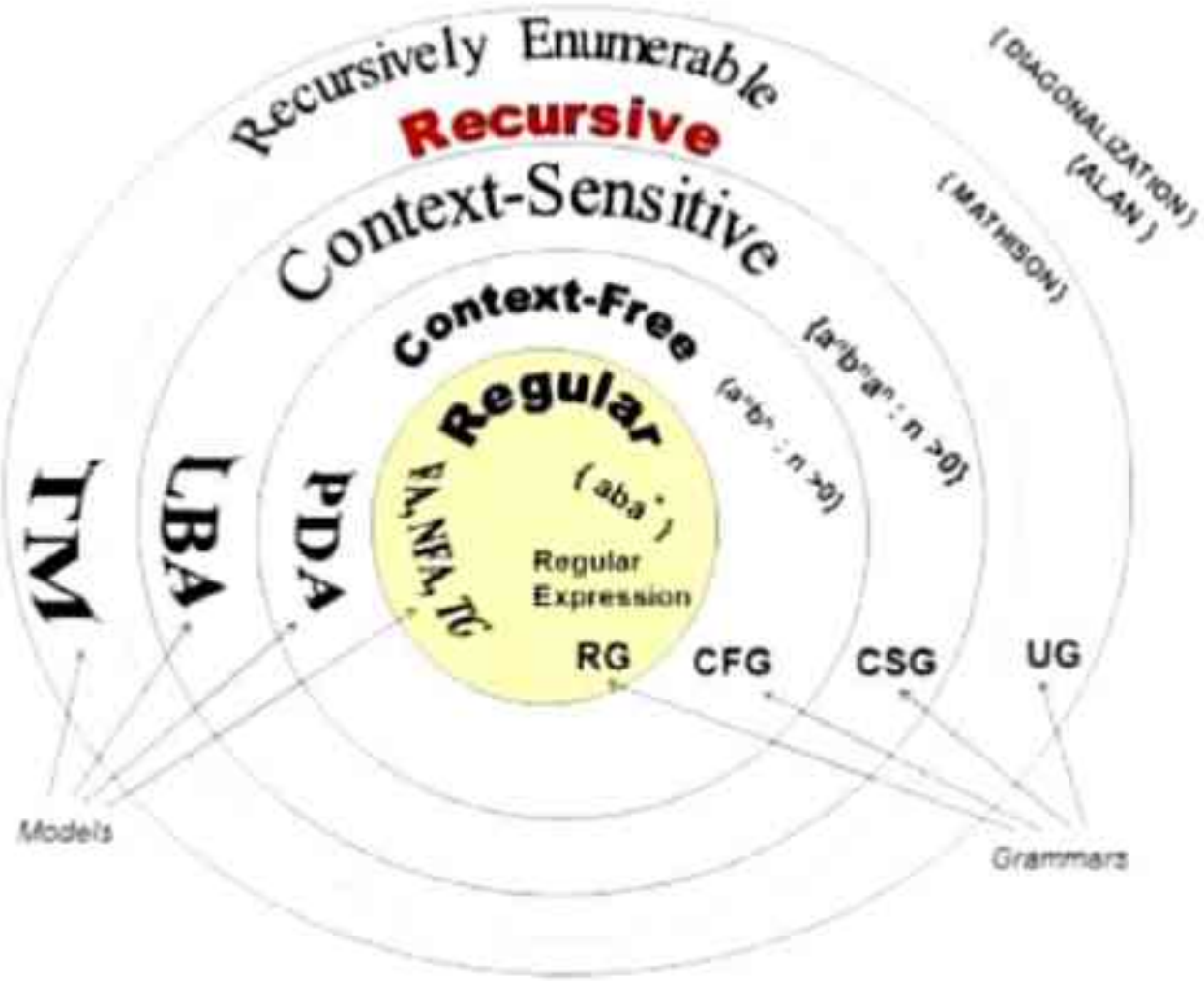
* Chomsky Heirarchy :-

-> Chomsky Heirarchy is a containment heirarchy of classes of formal grammars. This heirarchy of grammars was described by Noam Chomsky in 1956.

-> According to chomsky, grammar is divided into 4 types.

Type-0: Unrestricted grammar (TM)
Type-1: Context Sensitive grammar (LBA)
Type-2: Context free grammar (PDA)
Type-3: Regular grammar (FA)

-> Type-0:- Unrestricted grammar
> Includes all formal grammars.
> Type-0 grammar are recognized by turing machine.
> These languages are known as Recursively enumerable languages



Grammar production in the form

$\alpha \longrightarrow \beta$

α is $(V+T)^* V (V+T)^*$

β is $(V+T)^*$

V → Variables

T → Terminals.

→ In type 0, there must be atleast one variable on left side.

Eg:- $Sab \rightarrow ba$

$A \rightarrow S$

Type-1:- Context Sensitive grammar

→ These grammar generate context Sensitive language and recognized by linear bound Automata.

→ Type1 grammar should be in type-0

→ Grammar production $\alpha \rightarrow \beta$

$|\alpha| <= |\beta|$

i.e count of symbols in α is less than β.

Eg:- $S \rightarrow AB$

$B \rightarrow b$

Type-2:- Context free grammar

-> Type-2 grammars generate the context free language. The language generated by grammar is recognized by pushdown automata

-> It should be in Type 1

-> left side production can have only one variable $|\alpha| = 1$ no restriction on β

Eg:- $S \to AB$
$A \to C$

Type-3:- Regular grammar

-> Type-3 grammar generate regular language. These languages are exactly all languages that can accepted by Finite state automata

-> This is more restricted form of grammar

-> Type-3 should be in given form only

$V \to VT / T$ (left-regular grammar)

(or)

$V \to TV / T$ (right regular grammar)

Eg:- $S \to a$ The above form is called strictly regular

→ The another form of regular grammer is extended regular grammar.

$V \longrightarrow VT^* / T^*$

(extended left regular grammar)

$V \longrightarrow T^*V / T^*$

(Extended right regular grammar)