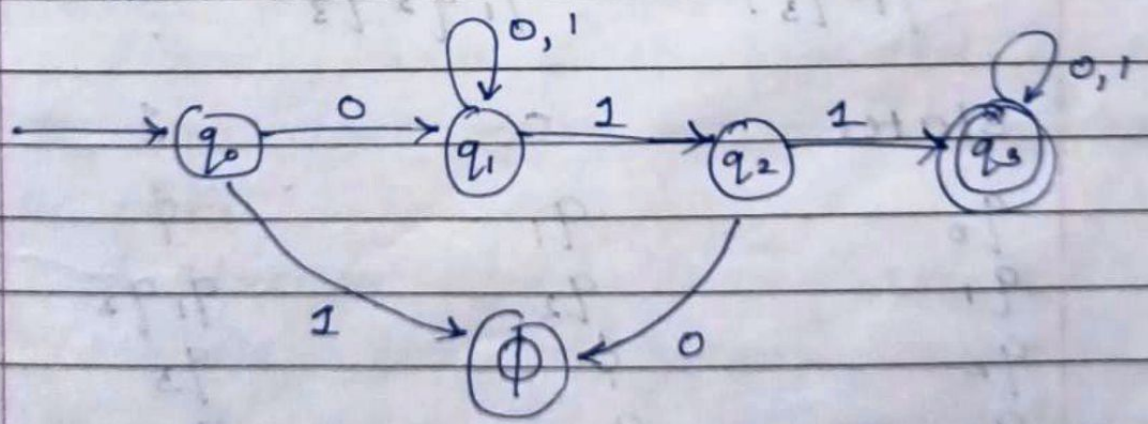
* Non-deterministic finite Automata to Deterministic finite Automata (NFA to DFA)

Q NFA → $M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, \{q_0\}, \{q_3\})$;

q_0 → initial state
q_3 → final state
where,

$\delta(q_0, 0) = \{q_1\}$, $\delta(q_0, 1) = \{\phi\}$
$\delta(q_1, 0) = \{q_1\}$, $\delta(q_1, 1) = \{q_1, q_2\}$
$\delta(q_2, 0) = \phi$, $\delta(q_2, 1) = \{q_3\}$
$\delta(q_3, 0) = \{q_3\}$, $\delta(q_3, 1) = \{q_3\}$

Diagramatic Representation of NFA



$\delta(q_0, 0) = q_1$
$\delta(q_0, 1) = \phi$
$\delta(q_1, 0) = q_1$

Date :

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

$\delta(q_1 q_2, 0) = \delta(q_1, 0) \cup \delta(q_2, 0)$

$= q_1 \cup \phi$

$= q_1$

$\delta(q_1 q_2, 1) = \delta(q_1, 1) \cup \delta(q_2, 1)$

$= q_1 q_2 \cup q_3$

$= q_1 q_2 q_3$

$\delta(q_1 q_2 q_3, 0) = \delta(q_1, 0) \cup \delta(q_2, 0) \cup \delta(q_3, 0)$

$= q_1 q_3$

$\delta(q_1 q_2 q_3, 1) = q_1 q_2 q_3$

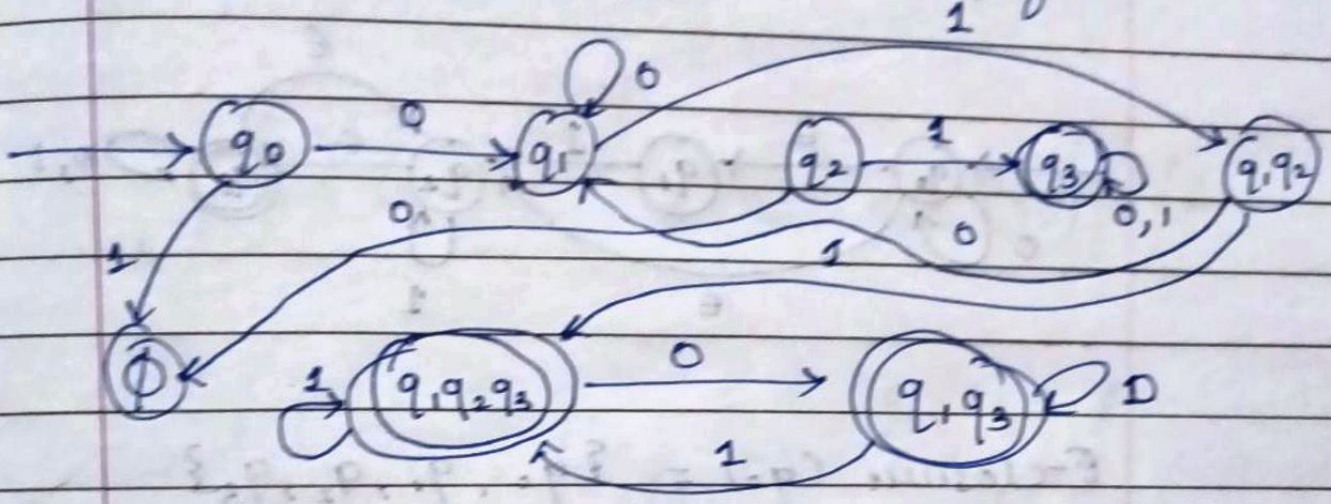
$\delta(q_1 q_3, 0) = q_1 q_3$

$\delta(q_1 q_3, 1) = q_1 q_2 q_3$

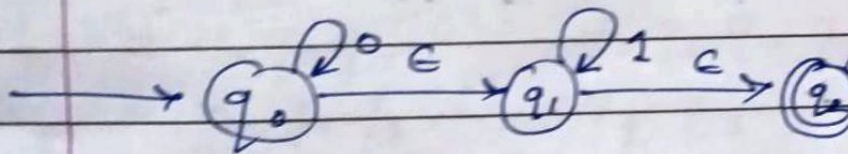
States	0	1
q_0	q_1	ϕ
q_1	q_2	$q_1 q_2$
q_2	ϕ	q_3
q_3	q_3	q_3
$q_1 q_2$	q_2	$q_1 q_2 q_3$
$q_1 q_2 q_3$	$q_1 q_3$	$q_1 q_2 q_3$
$q_1 q_3$	$q_1 q_3$	$q_1 q_2 q_3$

Page No.
Date :

Diagramatic Representation of DFA



* NFA with ϵ-moves.



ϵ-closure(q) = set of all those sets of the automata which can be reached from q and path labelled by ϵ. (q) as any state $q_0 q_1 q_2$ ---- but one at a time.

ϵ-closure of (q_0) = $\{q_0, q_1, q_2, q_3\}$

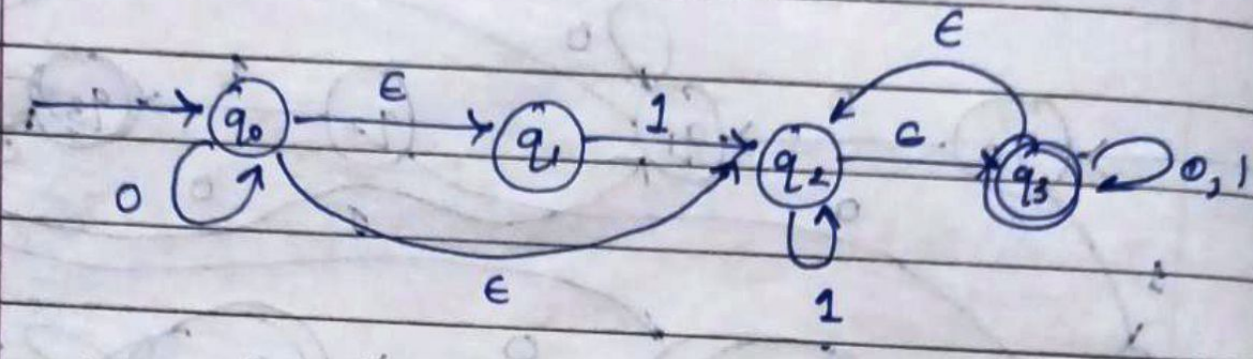
q_0 reaches $q_0 q_1 q_2$ over ϵ)

q_0 is reaching q_0 over 0 but ϵ-closure of q_0 is q_0 because q_0 reaches to itself over ϵ.

ϵ-closure (q_1) = $\{q_1, q_2\}$

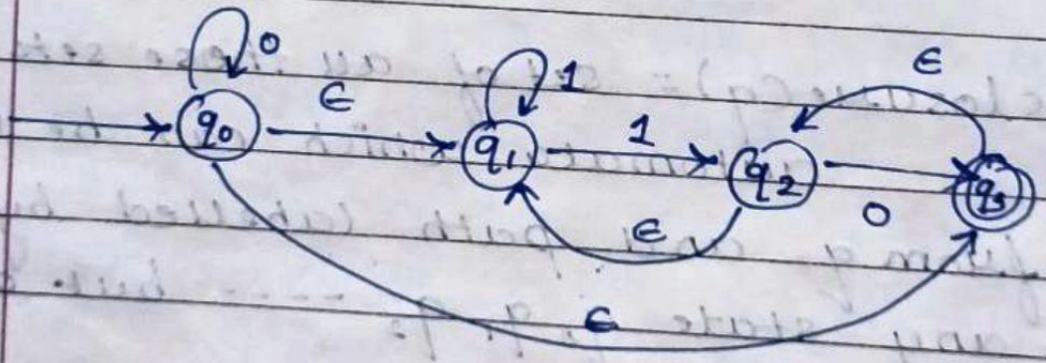
ϵ-closure (q_2) = $\{q_2\}$

Example:



E-closure $(q_0) = \{q_0, q_1, q_2, q_3\}$
E-closure $(q_1) = \{q_1\}$
E-closure $(q_2) = \{q_2, q_3\}$
E-closure $(q_3) = \{q_2 q_3\}$

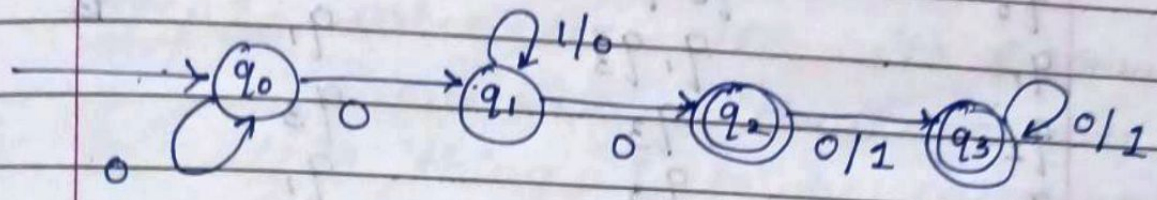
Example 1-



E-closure $(q_0) = \{q_0, q_1, q_2 q_3\}$
E-closure $(q_1) = \{q_1\}$
E-closure $(q_2) = \{q_2 q_1\}$
E-closure $(q_3) = \{q_1 q_2 q_3\}$

Page No.
Date :

* Convert to DFA



$\delta(q_0, 0) = q_0 q_1$ $\quad \delta(q_0, 1) = \phi$

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

$= q_0 q_1 \cup q_1 q_2$

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

$\delta(q_0 q_1, 1) = q_1$

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

$\delta(q_0 q_1 q_2, 1) = q_1 q_3$

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

$\delta(q_0 q_1 q_2 q_3, 1) = q_1 q_3$

$\delta(q_1 q_3, 0) = q_1 q_3$

$\delta(q_1 q_3, 1) = q_1 q_3$

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

$\delta(q_1 q_2, 0) = q_2 q_1 q_3$

$\delta(q_1 q_2, 1) = q_1 q_3$

$\delta(q_1 q_2 q_3, 0) = q_1 q_2 q_3$

$\delta(q_1 q_2 q_3, 1) = q_1 q_3$

$\delta(q_2, 0) = q_3$

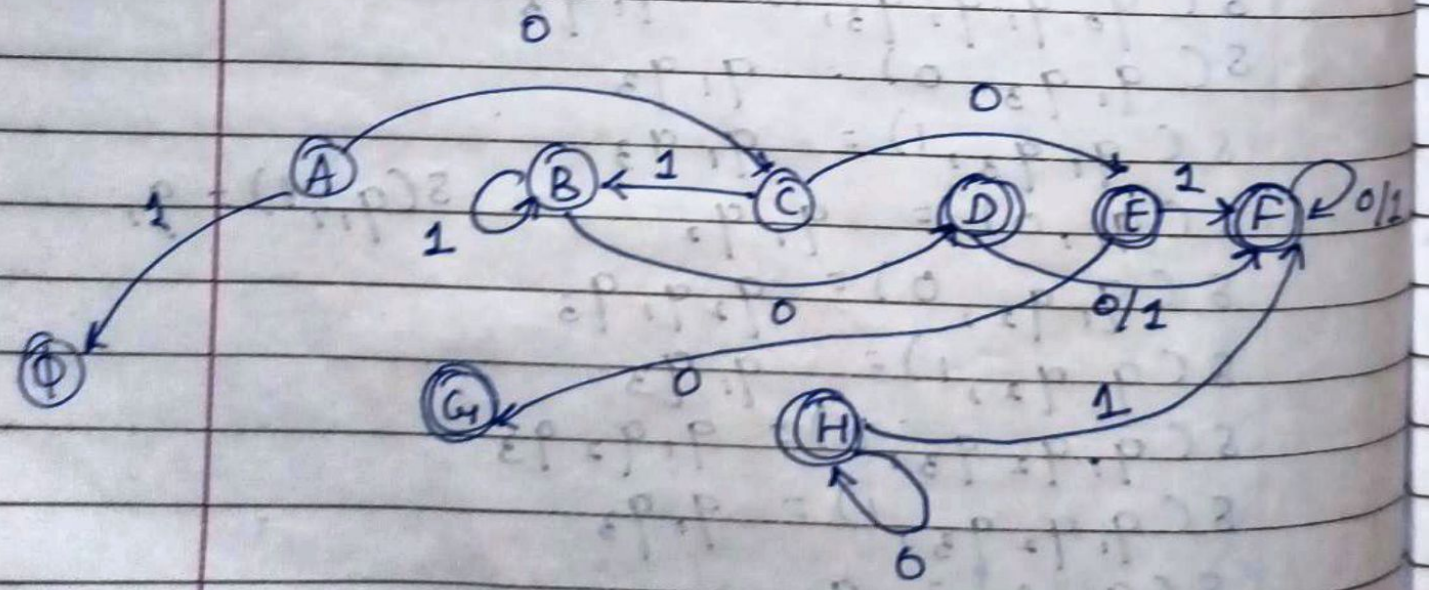
$\delta(q_2, 1) = q_3$

$\delta(q_3, 0) = q_3$

$\delta(q_3, 1) = q_3$

States	0	1	
q_0	$q_0 q_1$	ϕ	A
q_1	$q_1 q_3$	q_1	B
q_2	q_3	q_3	
q_3	q_3	q_1	
$q_0 q_1$	$q_0 q_1 q_2$	$q_1 q_3$	C
$q_1 q_2$	$q_1 q_3$	$q_1 q_3$	D
$q_0 q_1 q_2$	$q_0 q_1 q_2 q_3$	$q_1 q_3$	E
$q_1 q_3$	$q_1 q_3$	$q_1 q_3$	F
$q_0 q_1 q_2 q_3$	$q_0 q_1 q_2 q_3$	$q_1 q_3$	G
$q_1 q_2 q_3$	$q_1 q_2 q_3$	$q_1 q_3$	H

There is no state happening from $q_2 q_3$ so we'll ignore it



Page No.
Date :

* NFA with ϵ-moves to NFA without ϵ-moves.

Step 1:- Whenever NFA with ϵ-moves is to converted without ϵ-moves then first thing is to be calculated is as follows:-

calculate ϵ-closure of all states which are present in NFA with ϵ-moves

Step 2:- Apply the following formulae

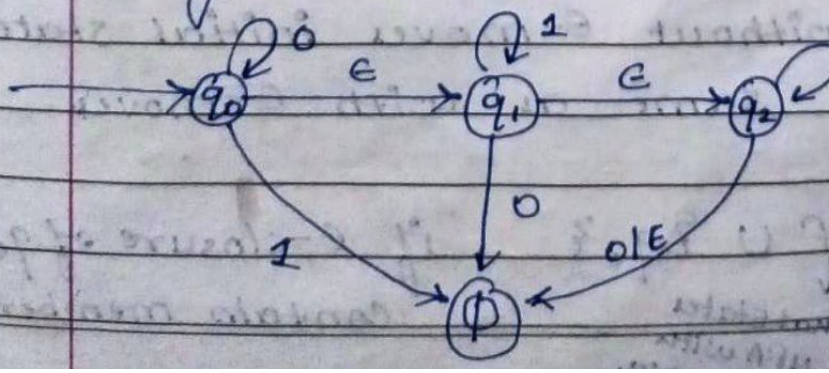
$$\delta'(q, a) = \text{E-closure}(\delta(\text{E-closure}(q), a))$$

Q. Consider the NFA with ϵ-moves given below.

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

δ	0	1	ϵ
q_0	q_0	ϕ	q_1
q_1	ϕ	q_1	q_2
q_2	ϕ	q_2	ϕ

Diagram:-



Page No.
Date :

Step 1:- Calculate the ε-closure of all states

ε-closure of q_0 = $q_0 q_1 q_2$
ε-closure of q_1 = $q_1 q_2$
ε-closure of q_2 = q_2

Step 2:- The formulae for calculating transition as following (transition of NFA without ε-moves)

$\delta_1(q_0, 0)$ = ε-closure (δ(ε-closure($q_0 q_1 q_2$), 0))
= ε-closure ($\delta(q_0 q_1 q_2, 0)$)
= ε-closure ($\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)$)
= ε-closure (q_0)
= $q_0 q_1 q_2$

$\delta_1(q_0, 1) = q_1 q_2$
$\delta_1(q_1, 0) = \phi$
$\delta_1(q_1, 1) = q_1 q_2$
$\delta_1(q_2, 0) = \phi$
$\delta_1(q_2, 1) = q_2$

NFA without ε-moves initial state will be same as with ε-moves.

$F_1 = F \cup \{q_0\}$, if ε-closure of q_0 contain member of

F: Final states of NFA without final states
F: Final states of NFA with ε-moves.

otherwise,

$F_1 = F$

Here,

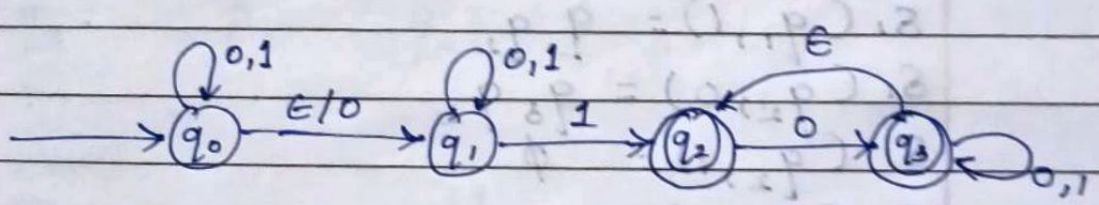
$F_1 = F \cup \{q_0\}$

ϵ-closure of q_0 is $q_0 q_1 q_2$ and q_2 is final state of NFA with ϵ-moves.

$\therefore F_1 = \{q_2\} \cup \{q_0\}$

$= q_2 q_0$

Question:-



NFA with ϵ-moves

States	0	1	ϵ
q_0	$q_0 q_1$	q_0	q_1
q_1	q_1	$q_1 q_2$	ϕ
q_2	q_3	ϕ	ϕ
q_3	q_3	q_3	q_2

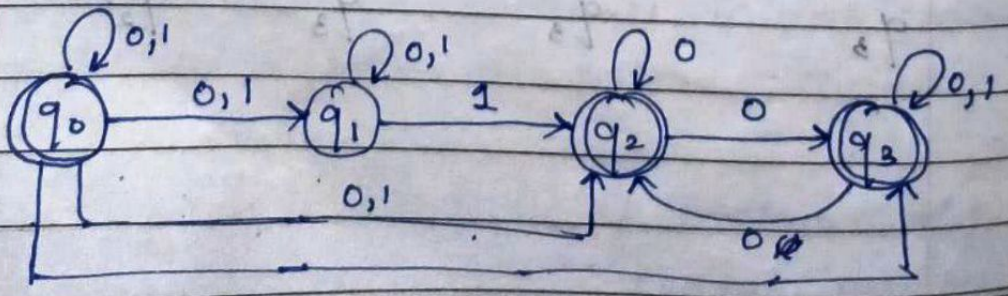
Page No.
Date :

→ E-closure of q_0 = $q_0 q_1 q_2$
E-closure of q_1 = q_1
E-closure of q_2 = q_2
E-closure of q_3 = $q_3 q_2$

$\delta_1(q_0, 0)$ = E-closure (δ(E-closure(q_0), 0))
= E-closure ($\delta(q_0 q_1 q_2, 0)$
= E-closure ($q_0 q_1 q_3$)
= $q_0 q_1 q_2 q_3$

$\delta_1(q_0, 1) = q_0 q_1 q_2$
$\delta_1(q_1, 0) = q_1$
$\delta_1(q_1, 1) = q_1 q_2$
$\delta_1(q_2, 0) = q_3 q_2$
$\delta_1(q_2, 1) = \phi$
$\delta_1(q_3, 0) = q_3 q_2$
$\delta_1(q_3, 1) = q_3 q_2$

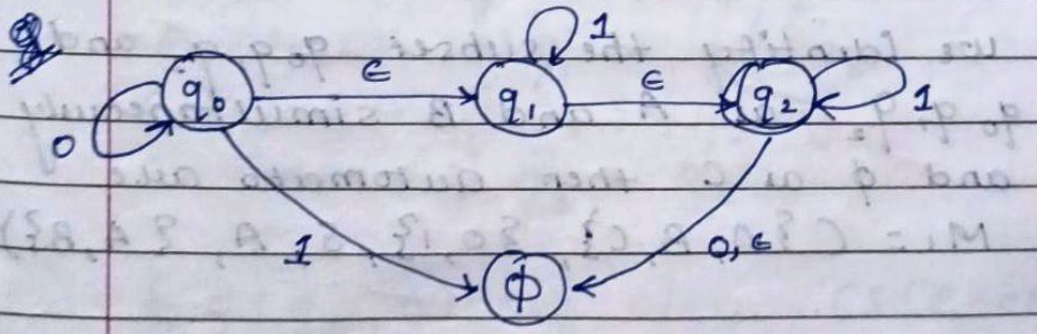
$F_1 = F \cup \{q_0\}$
$= \{q_2 q_3\} \cup \{q_0\}$
$= q_0 q_2 q_3$



δ	0	1
q_0	$q_0 q_1 q_2 q_3$	$q_0 q_1 q_2$
q_1	q_1	$q_1 q_2$
q_2	$q_3 q_2$	ϕ
q_3	$q_3 q_2$	$q_3 q_2$

Q] NFA with ϵ-moves to DFA.

δ	0	1	ϵ
q_0	q_0	ϕ	q_1
q_1	ϕ	q_1	q_2
q_2	ϕ	q_2	ϕ



E-closure of $q_0 = q_0 q_1 q_2$
E-closure of $q_1 = q_1 q_2$
E-closure of $q_2 = q_2$

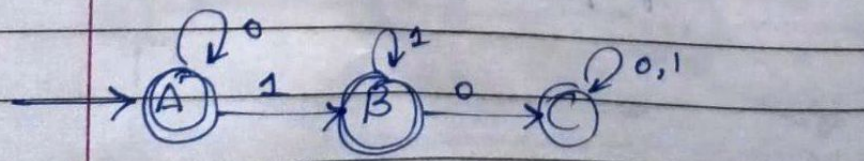
Page No.
Date :

$\delta_1(q_0, 0) = E\text{-closure}(\delta(E\text{-closure}(q_0), 0))$
$= E\text{-closure}(\delta(q_0 q_1 q_2), 0)$
$= E\text{-closure}(q_0)$
$= q_0 q_1 q_2$

$\delta_1(q_0, 1) = q_1 q_2$
$\delta_1(q_1, 0) = \phi$
$\delta_1(q_1, 1) = q_1 q_2$
$\delta_1(q_2, 0) = \phi$
$\delta_1(q_2, 1) = q_2$
$\delta_1(q_0 q_1 q_2, 0) = q_0 q_1 q_2$
$\delta_1(q_0 q_1 q_2, 1) = q_1 q_2$
$\delta_1(q_1 q_2, 0) = \phi$
$\delta_1(q_1 q_2, 1) = q_1 q_2$

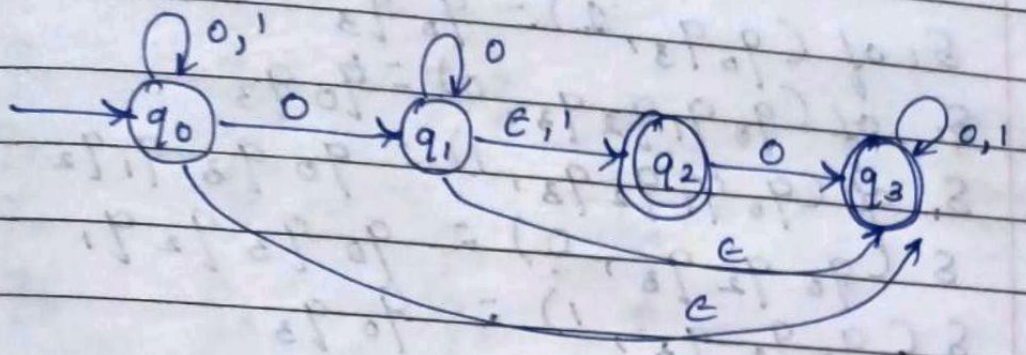
We identify the subset $q_0 q_1 q_2$ and $q_1 q_2$ as A and B simultaneously and ϕ as C then automata are
$M_1 = (\{A, B, C\}, \{0, 1\}, \delta, A, \{A, B\})$

δ_1	0	1
A	A	B
B	C	B
C	C	C



Page No.
Date :

Q



NFA with ε-moves to DFA

δ	0	1	ε
q_0	q_0q_1	q_0	q_3
q_1	q_1	q_3	q_2q_3
q_2	q_3	ϕ	ϕ
q_3	q_3	q_3	ϕ

→ ε-closure of q_0 = q_0q_3.
ε-closure of q_1 = $q_1q_2q_3$
ε-closure of q_2 = q_2
ε-closure of q_3 = q_3

$\delta'(q_0q_3, 0)$ = ε-closure (δ (ε-closure (q_0q_3), 0))
= ε-closure (δ (q_0q_3), 0)
= ε-closure ($q_0q_1q_3$)
= $q_0q_1q_2q_3$

δ_1 of $(q_0 q_3, 1) = q_0 q_3$

δ_1 of $(q_0 q_1 q_2 q_3, 0) = q_0 q_3$

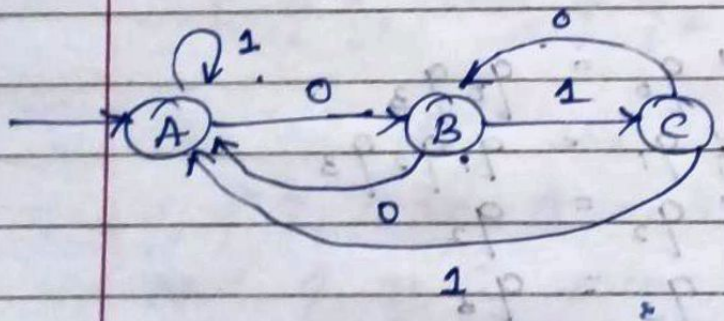
δ_1 of $(q_0 q_1 q_2 q_3, 1) = q_0 q_3 q_1 q_2$

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

$\delta(q_0 q_2 q_3, 1) = q_0 q_3$

Taking $\{q_0 q_3\}$, $\{q_0 q_1 q_2 q_3\}$ and $\{q_0 q_2 q_3\}$ as A, B and C

	δ_1	0	1
	A	B	A
*	B	A	C
*	C	B	A



Q

δ	0	1	∈
q_0	q_0	q_0q_1	q_3
q_1	q_2	q_1	q_2q_3
q_2	ϕ	q_3	ϕ
q_3	q_3	q_3	ϕ

NFA with ∈-moves to DFA

→ ∈-closure of $q_0 = q_0q_3$

∈-closure of $q_1 = q_1q_2q_3$

∈-closure of $q_2 = q_2$

∈-closure of $q_3 = q_3$

$\delta_1(\{q_0q_3\}, 0) = \in\text{-closure}(q_0q_3)$

$= q_0q_3$

$\delta_1(q_0q_3, 1) = q_0q_1q_2q_3$

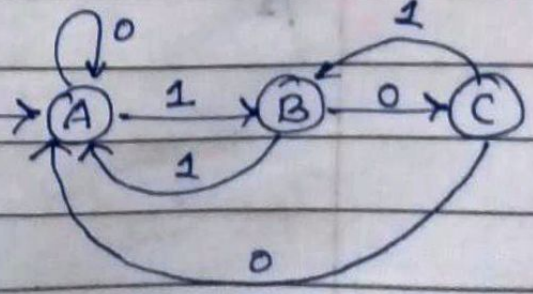
$\delta_1(q_0q_1q_2q_3, 0) = q_0q_2q_3$

$\delta_1(q_0q_2q_3, 0) = q_0q_3$

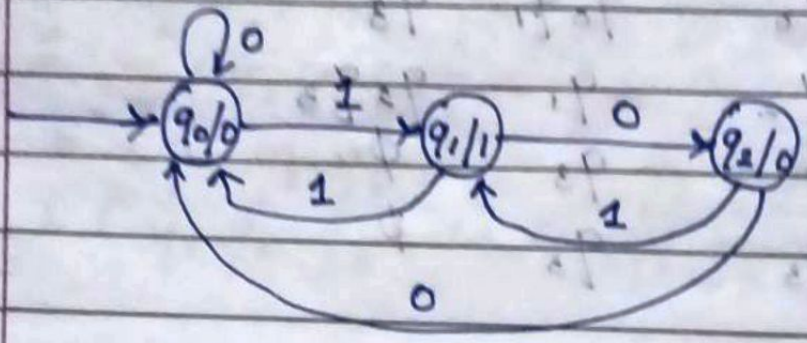
$\delta_1(q_0q_2q_3, 1) = q_0q_1q_2q_3$

q_0q_3, $q_0q_1q_2q_3$ and $q_0q_2q_3$ as A, B and C.

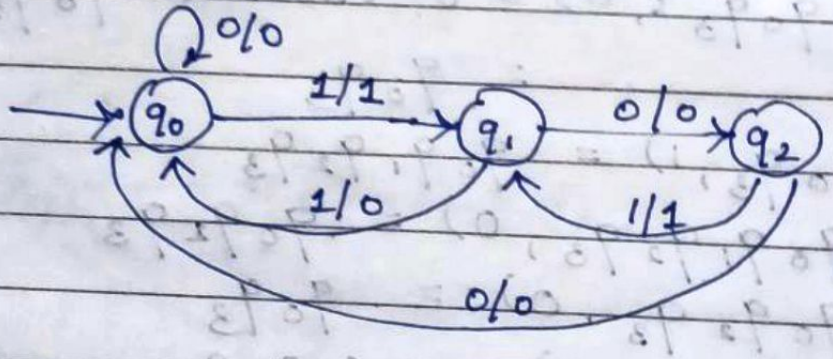
δ_1	0	1
A	A	B
B	C	A
C	A	B



* Moore to Mealy conversion.

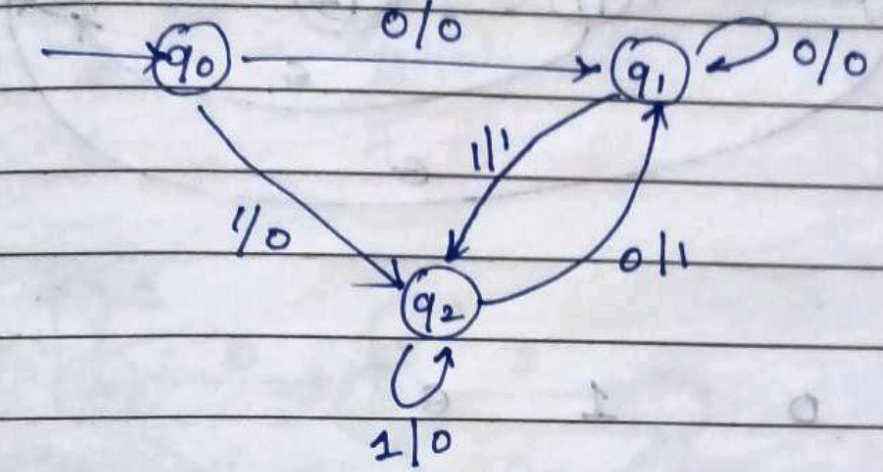


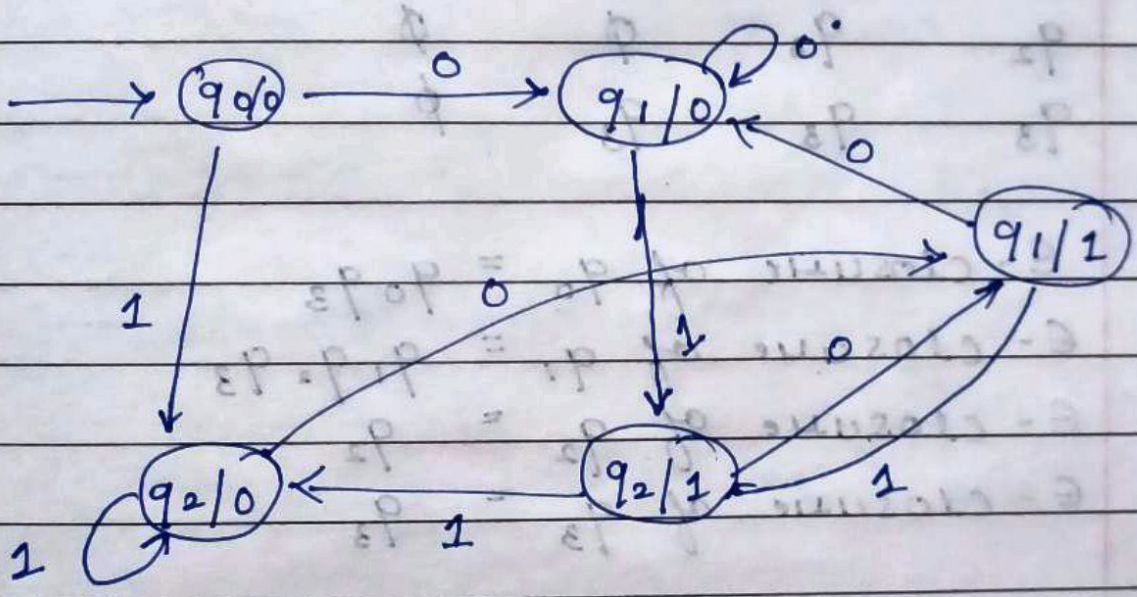
State	0	1	output
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_0	q_1	0



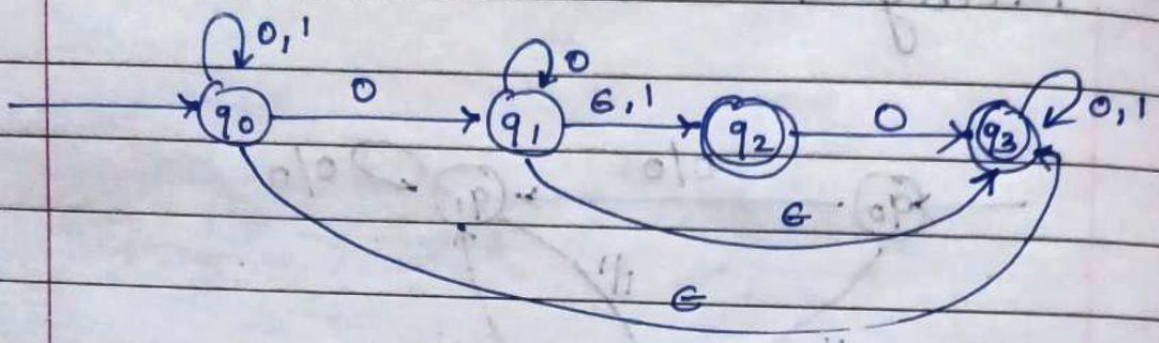
States	0	output	1	output
q_0	q_0	0	q_1	1
q_1	q_2	0	q_0	0
q_2	q_0	0	q_1	1

* Mealy to moore conversion.





Q.



δ	0	1	ϵ
q_0	$q_0 q_1$	q_0	q_3
q_1	q_1	q_2	$q_2 q_3$
q_2	q_3	ϕ	ϕ
q_3	q_3	q_3	ϕ

ϵ-closure of $q_0 = q_0 q_3$
ϵ-closure of $q_1 = q_1 q_2 q_3$
ϵ-closure of $q_2 = q_2$
ϵ-closure of $q_3 = q_3$

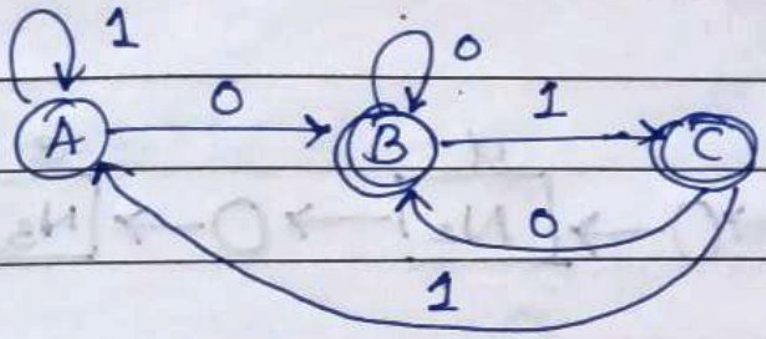
δ_1 of $(q_0 q_3, 0) = q_0 q_1 q_2 q_3$
$\delta_1 (q_0 q_3, 1) = q_0 q_3$
$\delta_1 (q_0 q_1 q_2 q_3, 0) = q_0 q_1 q_2 q_3$
$\delta_1 (q_0 q_1 q_2 q_3, 1) = q_0 q_2 q_3$
$\delta_1 (q_0 q_2 q_3, 0) = q_0 q_1 q_2 q_3$
$\delta_1 (q_0 q_2 q_3, 1) = q_0 q_3$

Considering:- $q_0 q_3 \rightarrow A$
$q_0 q_1 q_2 q_3 \rightarrow B$
$q_0 q_2 q_3 \rightarrow C$

Page No.
Date :

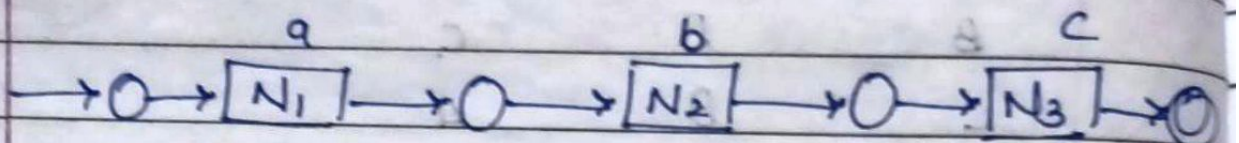
Transition table:-

δ	0	1
A	B	A
B	B	C
C	B	A

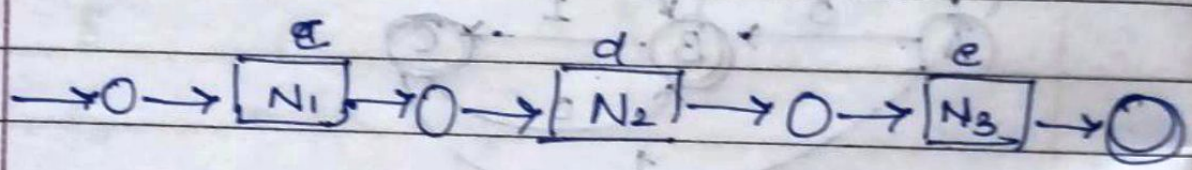


1) (abc/cde)*

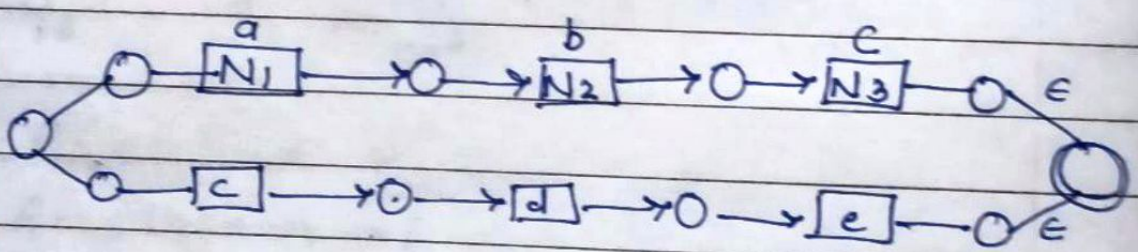
abc :-



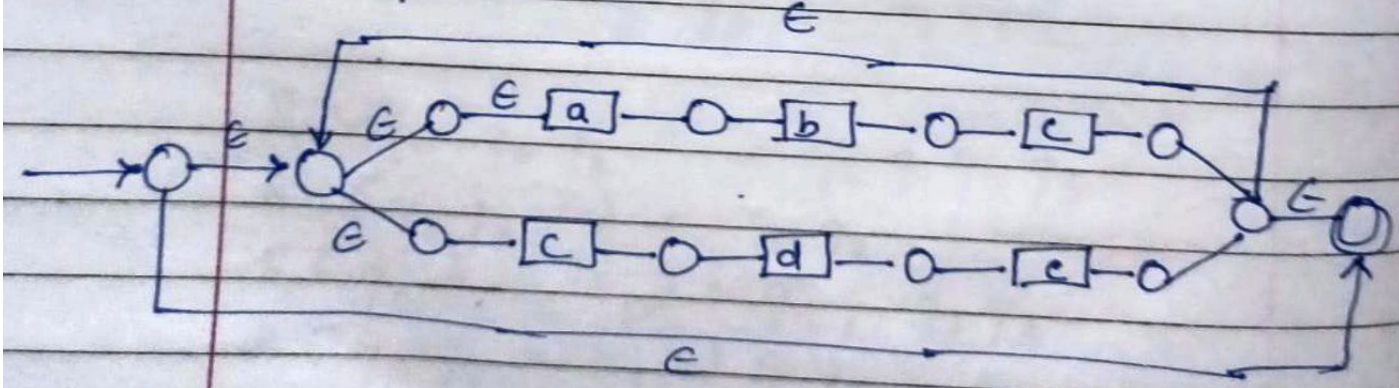
cde :-



abc/cde

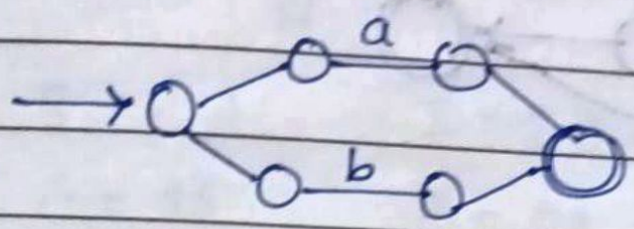


(abc/cde)*

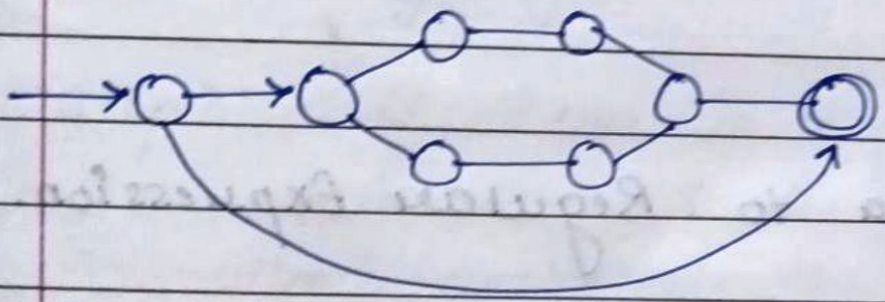


2) $(a+b)^*/d$

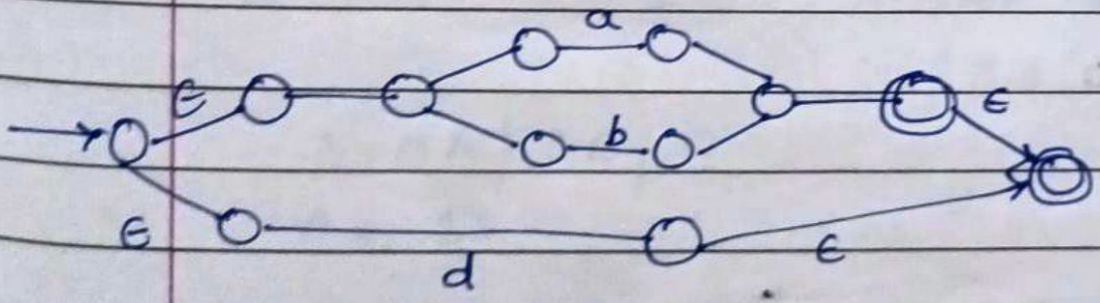
$a+b$



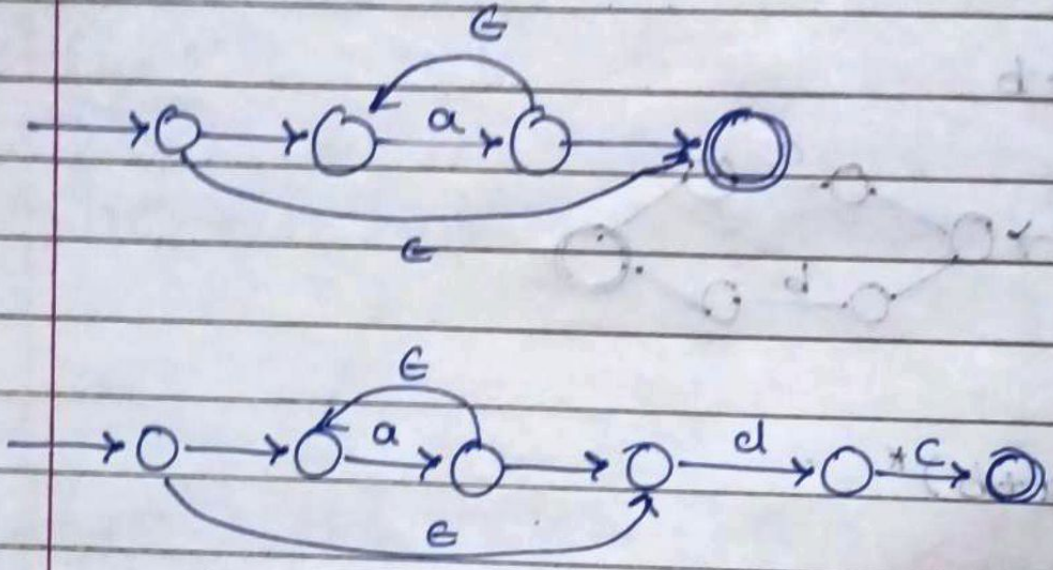
$(a+b)^*$



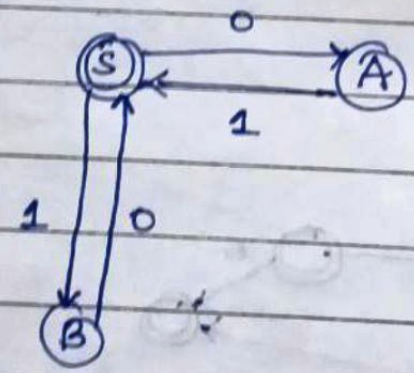
$(a+b)^*/d$



3) $a^* . b . c$



* Automata to Regular Expression.



1) Associate suitable variant with the state of automata

(S) → state (A) → 'A' state

Step 2:-

S is going to A over 0

S is going to B over 1.

$S = 0A$ $S = 1B$

$A = 1S$ $B = 0S$

A → is going to S over 1] outgoing
B → is going to S over 0 transition.

Step 3:- if there is a final state (in our case S is the final state) then associate the equation to final state $S = \epsilon$

$S = 0A$ $S = 1B$ $A = 1S$ $B = 0S$ $S = \epsilon$

$\therefore S = 0A \mid 1B \mid \epsilon$

$A = 1S$

$B = 0S$

Step 4:- Remember the equation as follows

$S = aS + b$

The regular expression for this expression is:-

$S = a^*b$

Date :

$S = 0(1S) \mid 1(0S) \mid \epsilon$ ← means +

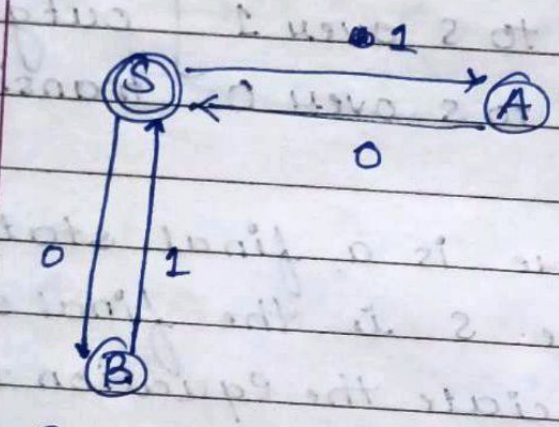
$= 0.1.S \mid 1.0.S \mid \epsilon$

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

$S = (01 + 10)^* \epsilon$

$S = (01 + 10)^*$

Q]



$S = 1A \quad S = 0B$

$A = 0S \quad B = 1S \quad 6 = \epsilon$

$S = 1A \mid 0B \mid \epsilon$

$= 1(0S) \mid 0(1S) \mid \epsilon$

$= 1.0.S \mid 0.1.S \mid \epsilon$

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

$S = (10 + 0.1)^* \epsilon$

$= (1.0 + 0.1)^*$

Page No.
Date :

PUMPING LEMMA.

* Pumping lemma is a theorem.
* Periodicity is characteristic of regular language.
* Pumping lemma is a theorem that tells about the periodicity characteristics.

Background

L → Regular language

M → {q, ∈, δ, q_0, F}

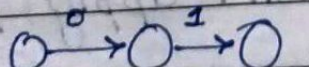
Represented in five tuples

let L be a regular language. There exists a finite automata accepting L. Therefore L = L(M) L language accepted by automata called M.

let N be a no. of state of automata[M] Consider a string of length greater than or equal to n.

(m ≥ n)

Example : length string {0, 1}



string is 3

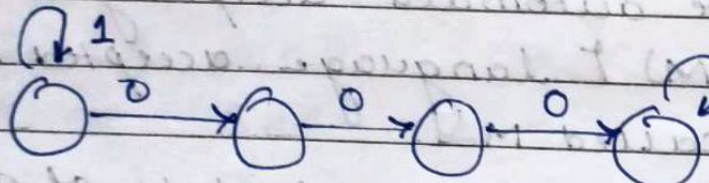
m=2, n=3 n>m.

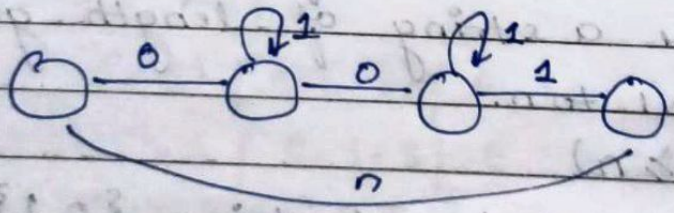
Consider a string (m ≥ n)

let this string be {a_1, a_2, a_3, ... a_m}

Page No.
Date :

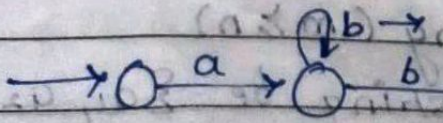
such that $m \geq n$. The path corresponding to string w starting with state q_0 cannot be in a chain. It will be a cyclic because there are maximum n states & length of the string which is m is greater than equal to n ($m \geq n$) and length of this cycle will be minimum '1' & maximum 'n' and this cycle will always start in 1^{st} n transition. That means all the state appearing on the path corresponding to the string can be distinct.





If $n > m$ (we don't apply pumping lemma)

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



pumping lemma is applied on $m \geq n$.

Date :

* statement of pumping lemma.

let L be a regular set that there's a constant of pumping lemma n such that for every word z mod of z

$|z| \geq n$

$z = a_1, a_2, \ldots\ldots a_m$

then $|z| = m$

$|z| = m \geq n$

let $z = uvw$

In such a way that $|uv| \leq n$

$1 \leq |v| \leq n$

Example:- $\underline{ab}\ \underline{b}\ \underline{bb}$

$u\ v\ w$

→ $ab = u$ $\{1 \leq |v| \leq n\}$ here $|v| = 1$

$b = v$

$bb = w$ $uv = abb$

$|uv| = 3$

→ $v = bb$ can be 1 to 3

$\underline{ab}\ \underline{bb}\ \underline{bb}$ here $\{1 \leq |v| \leq n\}$

$u\ v\ w$

$|v| = 2$

for all $i \geq 0$ uv^iw should be in L

$(i = 1, 2, \ldots\ldots \infty)$

Date :

Example :- $\underbrace{ab}_{u}\ \underbrace{bb}_{v}\ \underbrace{b}_{w}$ for v^i

→ $uv^iw = v = b$

→ $uv^2w = v = bb$

$\underbrace{ab}_{u}\ \underbrace{bb}_{v}\ \underbrace{bb}_{w}$

→ $uv^0w = ab\ bb$
$u\ \ w$

If all these strings is in L, then it is periodic

$uv^iw = \{ab, bb, abbb.b, ----\}$
$no.v, \quad no, v^2$

If its part of L then periodic

* Steps to solve the problem of pumping lemma.

Step 1 :- Given a language 'L' you have to prove that it is not regular language.

Step 2 :- To solve this kind of problem using pumping lemma we have to assume that 'L' a regular language

Page No.
Date :

Step 3 :- This is very important step because we have to select z as a string which is a part of 'L'
The selection of 'z' should be in such a way that $|z| \geq n$
where n is constant of pumping lemma.

Step 4 :- Therefore we write $z = uvw$ i.e we split z into 3 parts u, v and w the splitting z into uvw should satisfy following condition.

① $|uv| \leq n$

② $1 \leq |v| \leq n$

Step 5 :- Then for all possible choices of v we have to show that there exist an integer i, such that $i \geq 0$ & uv^iw should not be a part of 'L'.

Q Prove that
$L = \{ 0^{i^2} \mid i \geq 1 \}$ is not regular.

→ Step 1:- Let 'n' be a constant of pumping lemma so we will select $z = 0^{n^2}$ $|z| = n^2$. This will satisfy our condition that $|z| \geq n$.
Therefore z is sufficiently longer string on which you can apply pumping lemma.

Step 2:- Therefore we will write $z = uvw$ such that
① $1 \leq |v| \leq n$
② $|uv| \leq n$

Possible choices of value
$v = 0^p$
$1 \leq p \leq n$

Step 3:- For this choice of v
$|uv^2w|$ will vary from n^2+1 to n^2+n. Since n^2+1, n^2+n is not a perfect square of any integer numbers.

Therefore $|uv^2w| = z_i$, is not a perfect square of any integer number

hence it is proved that language 'L' is not regular.

Example :- $|uv^2w|$

000 000 000

n^2+1 let $n=3$

$3^2+1 = 10$

not a perfect square.

Hence, the language L is not regular.

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

prove that it is not regular.

→ [we want $L = \{01, 0011, 000111\}$ equal to followed by equal 1.

Example :- 000111

if $u=00$ & $w=111$

$v=0$ & if we pump v 1 times then it become 0000111 not equal no. of 0's & 1's]

Step 1 :- let n be a constant of pumping lemma.

∴ we will select $z = 0^n 1^n$

∴ $|z| = 2n$

Page No.
Date :

Therefore $|z| \geq n$

That's why we can apply pumping lemma on z

Step 2 :- we will divide $z = uvw$ such that

① $1 \leq |v| \leq n$

② $|uv| \leq n$

Step 3 :- possible choice of v choice

① if $v = 0^p$ where p is $1 \leq p \leq n$

Therefore uv^0w will be 0^{n-p} and 1^n. In this equation $0^{n-p} < 1^n$ $n-p \neq n$

∴ it means that in uv^0w there will less no. of zeros (0) as compared with (1).

∴ $0^{n-p} \notin 1^n$ will not be part of z.

∴ It proved that L is not regular

{000111 if we take $u = 00$ $v = 01$ & $w = 11$, we pump '01' one time, then 000101011, equal no. of 0 and 1 but 0 don't follow equal no of 1's}

Page No.
Date :

Q. Let L be $L = \{0^m / m \text{ is a prime number}\}$ is not regular.

→ Let m be a constant of pumping lemma.

i.e. we select $z = 0^n$.

$\therefore |z| = m$

$|z| \geq n$

We divide $z = uvw$ such that

a) $1 \leq |u| \leq n$

b) $|uv| \leq n$

Possible choices of v are

Choice ① If $v = 0^p$ where p is $1 \leq p \leq n$

for this choice of v.

$uv^i w = 0^{m+(i+1)p}$

$p = 1 - - - - n$

$i \geq 0 \quad i = 0 - - - - \infty$

$0^{m+(0-i)p} = 0^{m-p}$

i = how many times pump

If the case is such that

$i = m+1 \quad \{m = 3 \text{ zeros}\}$

The equation will be modified will be

$i = m+(m+1-1)p$

$i = m+mp$

$i = (1+p)m$

Page No.
Date :

Therefore $uv^iw = 0^{m*(1+p)}$

since $m \geq 1$ and $p \geq 1$

$1+p \geq 1$

that's why multiplication $m(1+p) \geq 1$ and this multiplication will not be a prime no.

$\therefore uv^iw$ = no. of zeros which are not prime no and therefore uv^iw will not be a part of L.

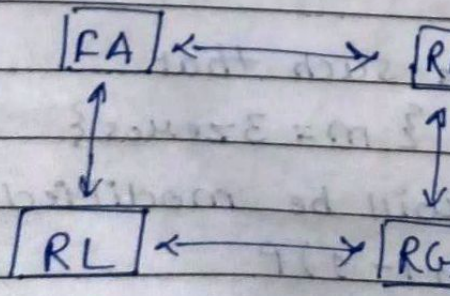
Hence proved.

Q// Prove that $L = \{0^n 1^m 0^{m+n} \mid n \geq 1$ and $m \geq 1\}$ is not regular.

→ $L = \{0^n 1^m \mid 1 \leq n \leq m\}$

{ 0's is equal or greater than 1's 00111 part of L

000111 not part of L.



Grammer has a tuple G.

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

V = Set of non-terminal (Capital Symbol which are written in capital letter)

T = Set of terminal (Symbol which are written in small letter)

$V = \{S\}$ = non-terminal

$T = \{a, b\}$ = terminal

P = production $P = \{S \to ab, S \to a\}$

P = production $P = \{S \to ab, S \to a\}$

P_1 – ⟶ –
P_2 – ⟶ –
P_3 – ⟶ – } Grammer

Left side start with capital symbol
On right side we have combination of terminal and non-terminal

P_1 $S \to abBc$
P_2 $A \to Bc$
P_3 $B \to aBA$ } G

$V = \{S, A, B\}$ — non terminals

$T = \{a, b, c, d\}$ — terminals

$P = \{P_1, P_2, P_3\}$

22/02/2022.

Example

$P_1 \longrightarrow S \longrightarrow abA$

$P_2 \longrightarrow A \longrightarrow cd$

S is a start symbol

Here $S \longrightarrow abA$

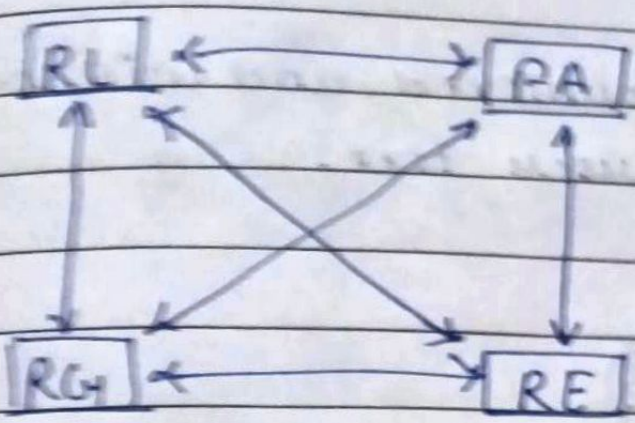
$A \longrightarrow cd$

then $S \longrightarrow abcd$

Start symbol will be firstmost process

Non-terminals is in the right side of the process. So Here S and A are non-terminals

Terminals are in the left side of the process. Here terminals are abcd. It is written in small letters



(b) G :- $S \longrightarrow aA$

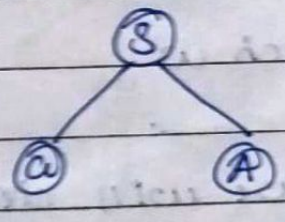
$A \longrightarrow b/\epsilon$

Step 1 :-

Start with S because its a start symbol

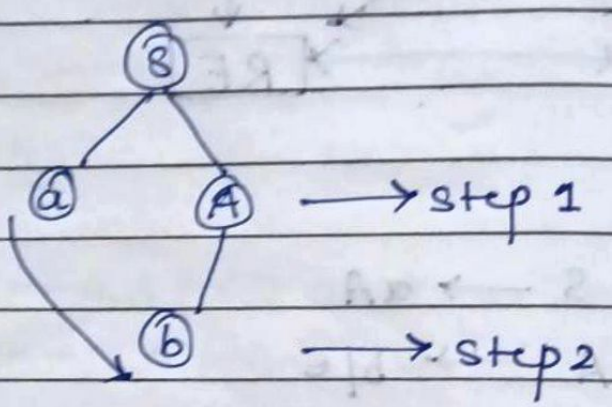
$\longrightarrow aA$

Now pausey tree will be made



Step 2 :- Pausey tree always scans from left to right. It is first scan a as it comes to the right side of the symbol a will be encountered. It is terminal Then A will be encountered A is non terminal. Now A will scanned b

will be encountered and will be added to the parsers tree.



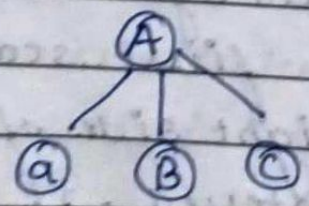
$L = \{ab\}$

Q $A \rightarrow aBc$

$D \rightarrow d \mid \epsilon$

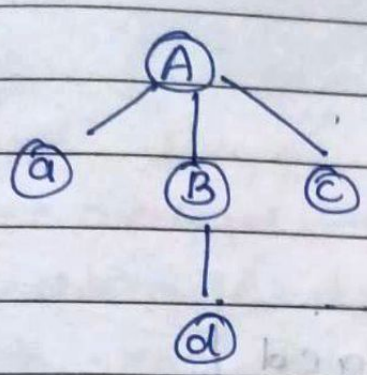
$w = adc$, then write the derivative step to reach w.

→ Step 1:- First we will derive A. Parser tree will be made



Step 2:- Now B will be derived as it is non-terminal

$B \rightarrow d$



step 1

step 2

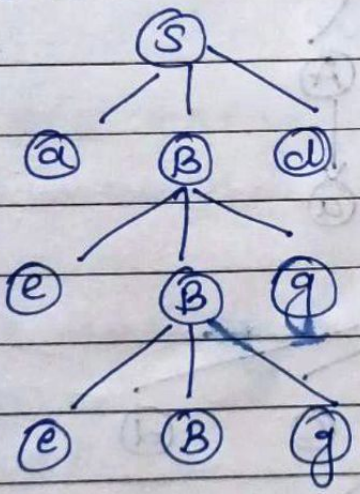
We have reached $w = adc$ through $B \rightarrow d$.

$L = \{adc\}$

Q. G_1:- $S \rightarrow aBd$

$B \rightarrow eBg \mid \epsilon$

step 1 :-



$L = \{ad, aegd, aeegggd, aeeeggggd, \ldots\}$

Date :

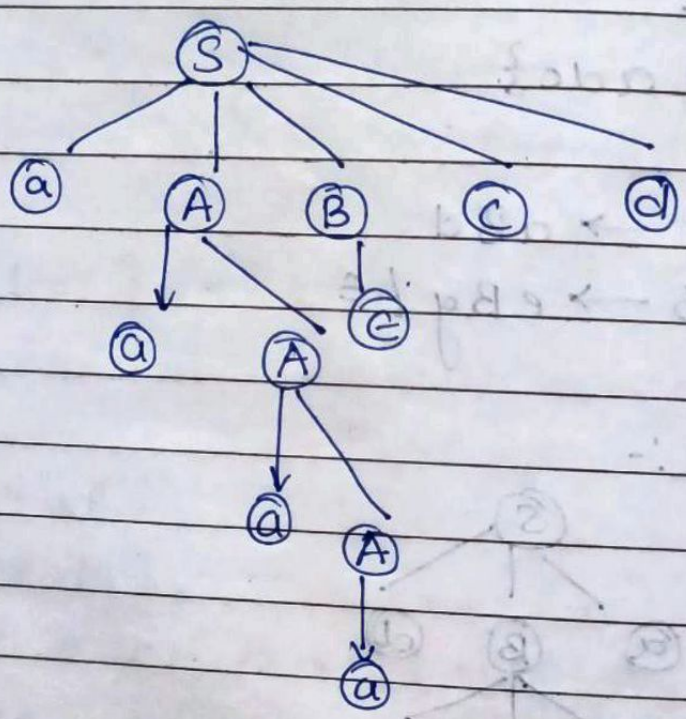
Q. $S \rightarrow aABcd$

$B \rightarrow eBg \mid \epsilon$

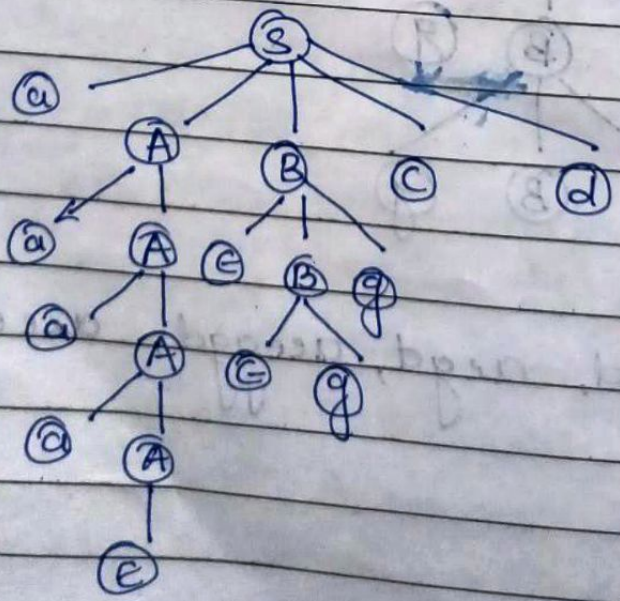
$A \rightarrow aA \mid \epsilon$

$\omega = aaaaeegg cd$

step (1) :-



step 2



Date :

Steps

$S \rightarrow aABcd \rightarrow aaABcd \rightarrow aaaABcd \rightarrow aaaaABcd \rightarrow aaaadeBcd \rightarrow aaaaBCD \rightarrow aaaaeBgcd \rightarrow aaaaeegg cd$

8 steps are involved.

* Right linear Grammer.

$A \rightarrow xB$
or
$A \rightarrow x$

} - condition to check Regd Right linear grammer.

x is a string of terminals. It can be one terminal one more than one.

Q: $A \rightarrow abC$
$B \rightarrow cd$
$C \rightarrow ef$

Check whether the grammer is right linear grammer or not

→ It should imply the condition $\boxed{A \rightarrow xB}$ or $\boxed{A \rightarrow x}$

Page No.
Date :

$A \rightarrow abBC \longrightarrow$ implies $A \rightarrow xB$
$B \rightarrow cd$ implies $B \rightarrow x$
$C \rightarrow ed$ implies $C \rightarrow x$
Hence this is linear grammar

* Regular Grammar

Regular Grammar is that grammar in which every production of regular grammar falls under one of these categories.

$A \rightarrow aB$

or

$A \rightarrow a$

where 'a' is a single terminal.

As in regular grammar there is a single terminal ('a') on the right side of the production. Therefore every regular grammar is right linear grammar. But every right linear grammar is not regular grammer.

$$\boxed{\begin{array}{l} A \rightarrow xB \\ A \rightarrow x \end{array}}$$

x is on the right and x is a string of terminals so x can be one terminal or more than one terminal. Therefore every right linear grammar will not be regular grammar.

* Q. Consider the following grammar
G:- S → aaB / ab
B → bB / bb

Represent the above grammar into regular grammar

→ S → aaB
↳ (non-terminal ∴ right linear grammar)

S → aC (aB is Replaced by C)
C → aB ↵ write
(language should not change)
I is regular grammar we have one single terminal]
For S → ab
(S → a will be wrong because it will change language]
S → aD
D → b

For $B \rightarrow bB$

Here we have single terminal 'b' and one non-terminal ($A \rightarrow aB$)

$\therefore$ It is regular Grammar.

For,

$B \rightarrow bb$

$B \rightarrow bD$

(we have already taken $D \rightarrow b$ so here we replace b to D)

$S \rightarrow aC$ $\quad B \rightarrow bB$

$C \rightarrow aB$ $\quad B \rightarrow bD$

$S \rightarrow aD$

$D \rightarrow b$

Page No.
Date :

① Prove that, $L = \{ww / w \text{ is in } (0/1)^*\}$ is not regular language.

→ $L = \{\epsilon, 0, 1, 00, 11, 01, 10, 000, 111, 011, ----\}$

Here length of $w = n$

$z = |www| = n$

$|z| > n$

let, $z = \underbrace{011}_{u}\underbrace{01}_{v}\underbrace{011}_{w}\underbrace{01}_{w}$

let $01 = v$

taking $v' = 01$

v is pumped once

then $w = \underbrace{011}_{w}\underbrace{0101}_{w}$, $\underbrace{011}_{w}\underbrace{01}_{w}$

$w \neq w$

Hence it is not regular.

② $S \rightarrow cdE / ef$
$B \rightarrow fgbL / uvw$ } convert into regular grammar

→ For,

$S \rightarrow cdE$

we replace dE with A, so,

$S \rightarrow cA$
$A \rightarrow dE$

Now, for

$S \rightarrow ef$

Now, we replace f will W, so,

$S \rightarrow eW$

$W \rightarrow f$

For,

$B \rightarrow fgbL$

$B \rightarrow fV$ (replacing gbL with V)

$V \rightarrow gbL$

Now V is not in regular form so,

$V \rightarrow gX$ (replacing bL with X)

$X \rightarrow bL$

For,

$B \rightarrow uvw$

$B \rightarrow uM$ (vw replacing with M)

$M \rightarrow vw$

$M \rightarrow vN$ (replacing w with N)

$N \rightarrow w$

So, the final process is:-

$S \rightarrow CA$ $A \rightarrow dE$

$S \rightarrow eW$ $W \rightarrow f$

$B \rightarrow fV$ $V \rightarrow gX$

$B \rightarrow uM$ $X \rightarrow bL$

$M \rightarrow vN$ $N \rightarrow w$

Date :

Left Linear
$A \rightarrow Bx$
$A \rightarrow x$

Right Linear
$A \rightarrow xB$
$A \rightarrow x$

* Context Free Grammar (CFG)

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

$V = \{S, A, B\}$ → Non-terminals

$T = \{a, b\}$ → Terminals

$P = \{P_1, P_2\}$ → Production

$S \rightarrow \{SS\}$ → start string.

$A \rightarrow (V \cup T)^*$ → All possible combination

① 0 PC $(\cup VT)$

② 1 PC of $(\cup VT)$ of length 1

③ 2 PC of $(\cup VT)$ of length 2.

Parse Tree

$G = A \rightarrow aA$

$A \rightarrow ab/a$

$\omega \rightarrow aab$

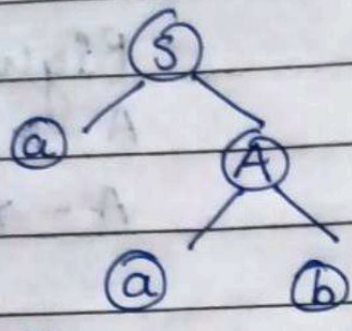
$L(G) \rightarrow CFL = \{aab, aa\}$

Q. $G: S \rightarrow aA$

$A \rightarrow ab/a$

$\omega = aab$

$S \rightarrow aA \rightarrow aab$ leftmost derivation



→ Top down parser

→ left to right

travelling from left to right we get aab similar to w.

* Bottom up Parser

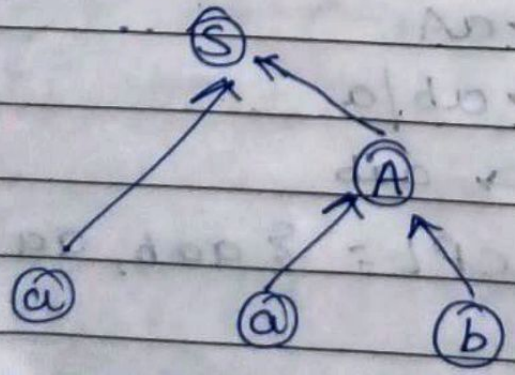
This will decrement w to s

Ex:- $S \rightarrow aA$

$A \rightarrow ab \mid a$

$w \rightarrow aab$

$w \rightarrow aab \rightarrow aA \rightarrow S$ → Reduction



→ bottom up parser.

Left Most Derivation

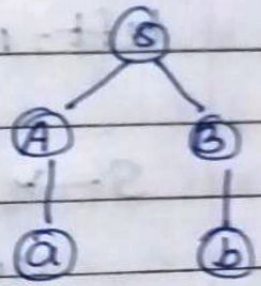
In this parser will first expand the left down non-terminal to the terminal then move to the left most non-terminal. This is used in top down Parser derivation

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow \underline{A}B \rightarrow a\underline{B} \rightarrow ab$



Right most derivation

Right most non-terminal is derived first.

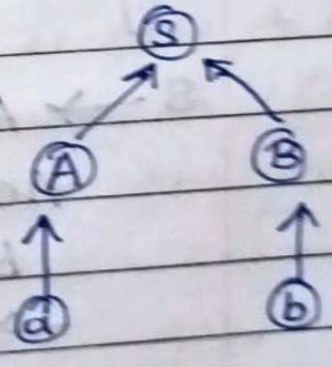
$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A\underline{B} \rightarrow \underline{A}b \rightarrow ab$

$\leftarrow$ bottom up parser



To do bottom-up parser it will find right most derivation and reverse it.

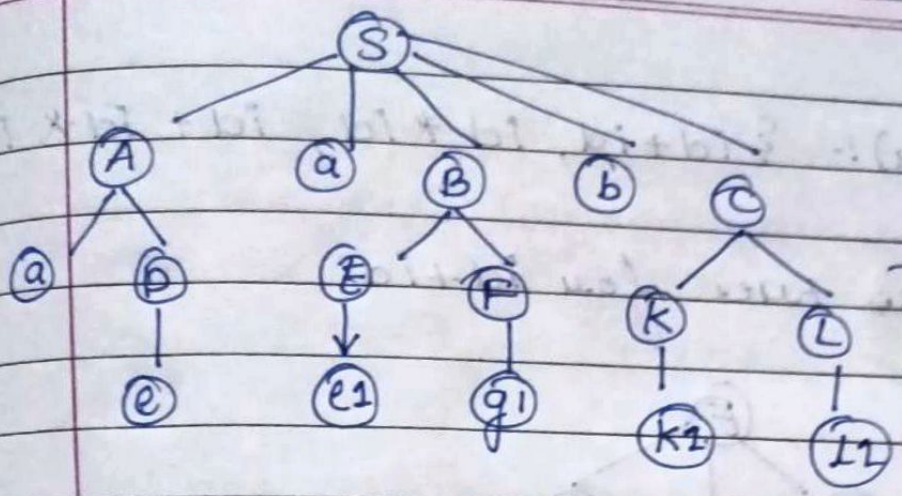
Q. G: S → AaBbC F → g1
A → aD K → k1
B → Ef L → l1
C → kL
D → e
E → e1

→ left-most derivation

S → AaBbC → ADaBbC → aeaBbC →
aeaEfbC → aeae1fbC → aeae1g1bC →
aeae1g1bkL → aeae1g1bk1L →
aeae1g1bk1l1

Right Most derivation

S → AaBbC → AaBbkL → AaBbkl1 →
AaBbk1l1 → AaEfbk1l1 → AaEg1
bk1l1 → Aae1g1bk1l1 → aDae1g1bk1l1
→ aeae1g1bk1l1



→ left to right.

For any grammar G, if we write down left-most and rightmost derivation and if we draw a parse tree from top to bottom then both the parse tree will be same.

* Ambiguous Grammar

A Grammar G is said to be ambigious, if for some w in L(G) there can be more than one parse tree.

① Example:-

G :- $E \rightarrow E+E$

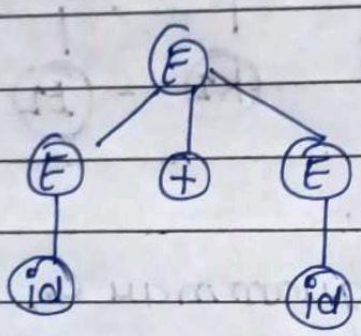
$E \rightarrow E*E$

$E \rightarrow id$

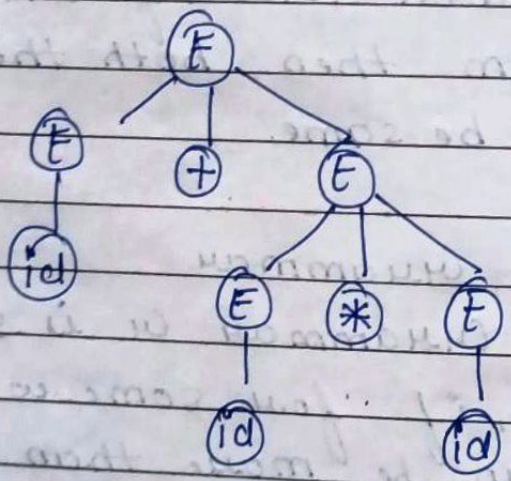
Prove that the grammar is ambiguous or not.

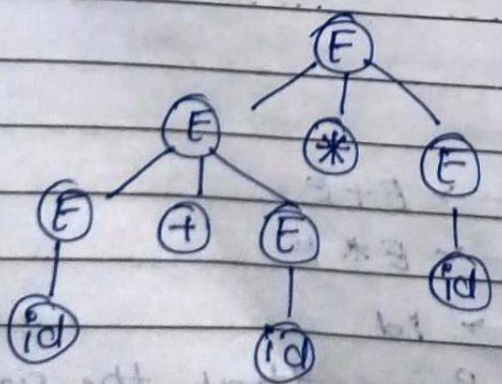
→ L(G1) :- {id+id, id*id, id+id*id}

Parse tree for id+id



Parse tree for id+id*id



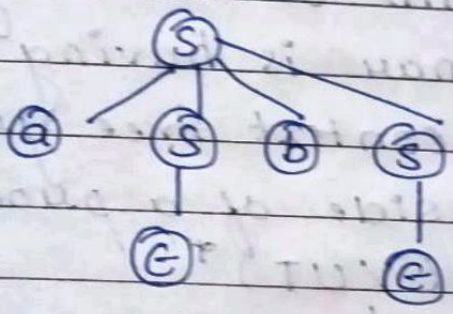


W = id+id*id has two different parser tree. So, the Grammar G1 is ambiguous.

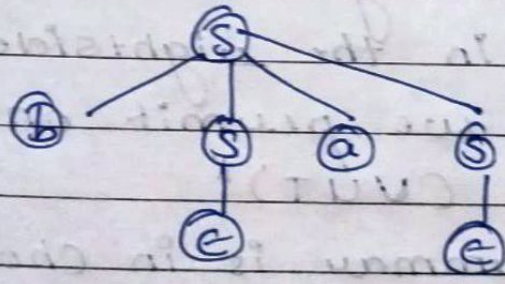
(2) G1 → S → aSbS | bSaS | ε . prove whether the grammer is ambigious or not.

→ L(G1) :- {ab, ba, abab, baba, abba

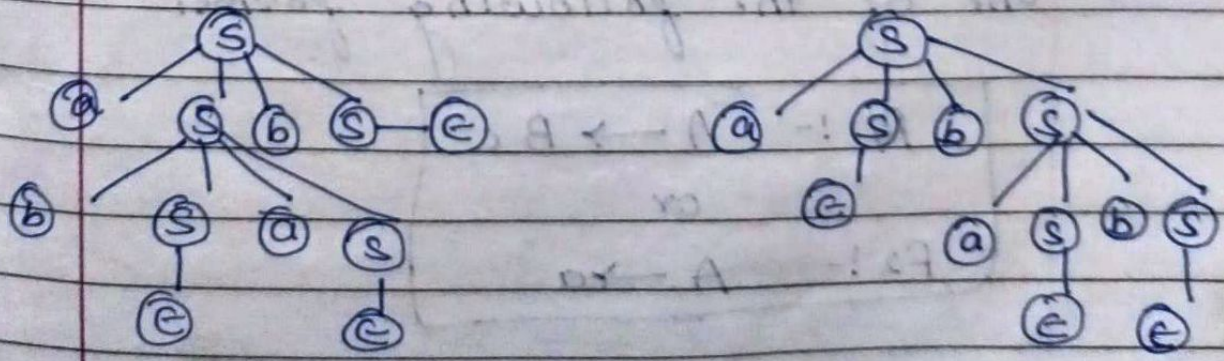
Parse tree for ab



Parse tree for ba



Parse tree for abab



Date :

There is two parse tree for abab so the grammar is ambiguous.

* Chomsky Normal Form (CNF)

A grammar is said to be in normal form when every production of a grammar is having specific form. In context free grammar (CFG) the right side of a production consist of $(V \cup T)^*$

↳ V union T

$V \rightarrow (V \cup T)^*$

when a grammar is in normal form then in the rightside of the production we permit only specific member of $(V \cup T)^*$

A grammar is in Chomsky Normal Form when every production of the Grammar will be either in one of the following forms:

F1 :- $A \rightarrow BC$

or

F2 :- $A \rightarrow a$

where a is a single terminal

Q) Consider the following CFG

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

Convert the following grammar into Chomsky Normal form.

→ $S \rightarrow aB \rightarrow CB$

$(C \rightarrow a)$

$S \rightarrow bA$

$S \rightarrow DA \quad (D \rightarrow b)$

$A \rightarrow a$

$A \rightarrow CS \mid DA$

14/03/202.

Page No.
Date :

* Imp

* Left Recursion :-

A grammar is said to be left recursive grammer if any production in the grammer is of the form

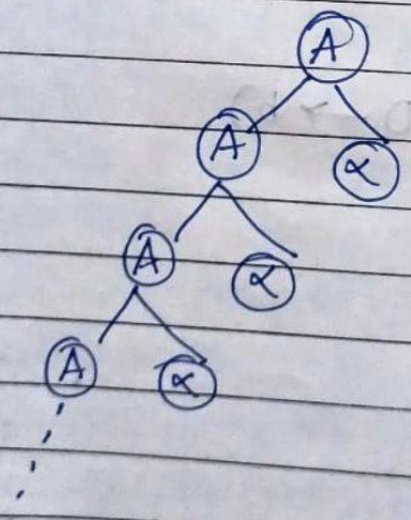
$$A \longrightarrow A\alpha \mid \beta$$

A → Non-terminal

α → single terminal

β → combination of terminals and non-terminals.

left recursion parse tree



The above scenerio is called left recursion because always parser will select leftmost non-terminal A and try to expand A to Aα. Therefore

as seen from the diagram parser will go into a infinite loop and the passing process will be slowed down.

$A \longrightarrow A\alpha \mid \beta$

To remove left recursion, it is replaced by

① $A \longrightarrow \beta B$ (B is new non-terminal)

② $B \longrightarrow \alpha B \mid \epsilon$

① Example:- $S \longrightarrow AB$

$A \longrightarrow Aa \mid c$

$B \longrightarrow d$

→ $A \longrightarrow Aa \mid c$ is left recursive, so to remove this

$A \longrightarrow \underset{A\alpha}{Aa} \mid \underset{\beta}{c}$ will be replaced by

$A \longrightarrow cD$

$D \longrightarrow aD \mid \epsilon$

② Consider the grammer: $S \longrightarrow aBDh$

$B \longrightarrow Bb \mid c$

$D \longrightarrow EF$

$E \longrightarrow g \mid \epsilon$

$F \longrightarrow f \mid \epsilon$

Remove the left recursion from the above.

→ Step 1:- From the above grammar left recursive production is as follows

$B \rightarrow Bb/c$

Step 2:- We will treat above production in the format

$A \rightarrow A\alpha/\beta$

Step 3:- If $A \rightarrow A\alpha/\beta$ production is replaced by following productions

① $A \rightarrow \beta B$

② $B \rightarrow \alpha B/\epsilon$

then,

$B \rightarrow Bb/c$ will be replaced by

① $B \rightarrow cM$

② $M \rightarrow bM/\epsilon$

So, the reduced grammar (after left recursion is) is:-

$S \rightarrow aBDh$

$B \rightarrow cM$

$M \rightarrow bM/\epsilon$

$D \rightarrow EP$

$E \rightarrow g/\epsilon$

$P \rightarrow f/\epsilon$

③ Consider the following grammar

$S \rightarrow A$

$A \rightarrow Ad/Ae/aB/ac$

$B \rightarrow bBc/f$

$C \rightarrow g$

→ From the above the left recursive production is

$A \rightarrow Ad/Ae/aB/ac$

This production is replaced by,

$A \rightarrow aBM$

$A \rightarrow acM$

$M \rightarrow dM/\epsilon$

$M \rightarrow eM/\epsilon$

This can be written as

① $A \rightarrow aBM/acM$

② $M \rightarrow dM/eM/\epsilon$

So, the reduced grammar is:-

$S \rightarrow A$

$A \rightarrow aBM/acM$

$M \rightarrow dM/eM/\epsilon$

$B \rightarrow bBC/f$

$C \rightarrow g$

④ GNB :- S → A

A → Al / Am / eB / bc

B → bBC / f

C → g

→ So, here left recursive is:

$A \rightarrow \underset{\alpha}{Al} / \underset{\alpha}{Am} / \underset{\beta}{eB} / \underset{\beta}{bc}$

It can be replaced by :-

A → eBM

A → bCM

M → lM / ϵ

M → mM / ϵ

Can be written as,

A → eBM / bCM

M → lM / mM / ϵ

So, the grammar will be.

S → A

A → eBM / bCM

M → lM / mM / ϵ

B → bBc / f

C → g

* Greibach Normal Form:- (GNF)

Any grammar is in greibach Normal form (GNF) if, any production belongs to following category:

$[A \rightarrow a\alpha]$

where,

A → Non-terminal

a → single-terminal

α → string of Non-terminals (possibly empty)

① Consider the following grammar

G_1 :- $S \rightarrow aSa \mid bSb \mid \epsilon$

Convert it into GNF.

→ $S \rightarrow aSa \rightarrow aSM$

$M \rightarrow a$

$S \rightarrow bSb \rightarrow bSN$

$N \rightarrow b$

② G_{11} :- $S \rightarrow bcD$

$D \rightarrow e$

Convert it into GNF

→ $S \rightarrow bcD \rightarrow bMD$

$M \rightarrow c$

$D \rightarrow e$

Date :

* ∈-productions :-

A grammer G is having ∈ productions, it means that, the grammer G is having minimum 1 production of following type

$$\boxed{A \to \epsilon}$$

Example :- In the following grammar

G :- $S \to aABdh$

$A \to d/e/\epsilon$

$B \to f/\epsilon$

There are 2 ∈ production

$A \to \epsilon$

$B \to \epsilon$

So the grammar is ∈-production.

∈-production in the above grammar are $A \to \epsilon$ and $B \to \epsilon$, which are useless, thats why, $A \to \epsilon$ and $B \to \epsilon$ need to be removed from the grammar G.

① Consider the following grammar

G: S → aA

A → b/ε

eliminate the ε-productions from the grammar

→ Step 1:- To remove ε-production A → ε from the grammar G, we will see that in the above grammar G, at which point A is coming on the right side.

A is occuring in the right side of following production

S → aA

In the above production we will replace A on the right side by ε.

S → aA (A → ε)

S → a

So our final grammar will be:-

S → aA/a

A → b

(2) Consider the following grammar

G:- $S \rightarrow aABA$

$A \rightarrow b \mid \epsilon$

Remove the ϵ-production from the grammar.

→ Step 1:- To eliminate $A \rightarrow \epsilon$ from grammar G we will identify the production from G in which A is occuring in the right side of production.

$S \rightarrow aABA$

Replacing $A \rightarrow \epsilon$

$S \rightarrow aABA$

$S \rightarrow aBA$ (Replacing first $A \rightarrow \epsilon$)

$S \rightarrow aABA$

$S \rightarrow aAB$ ϵ

$S \rightarrow aABA$

$S \rightarrow aB$

So, our final grammar will be

$S \rightarrow aABA \mid aBA \mid aAB \mid aB$

$A \rightarrow b$

Page No.
Date :

8 Ans :- $S \to ABAC$

$A \to aA \mid \epsilon$ | $A \to aA$, $A \to \epsilon$

$B \to bB \mid \epsilon$ | $B \to bB$, $B \to \epsilon$

$C \to c$

Remove the ϵ-production from the grammar

→ $S \to ABAC$

$S \to BAC \to AC$

$S \to ABAC$ $\quad S \to ABAC$

$S \to ABC \quad S \to BC$

$S \to ABAC \quad S \to ABAC$

$S \to AAC \quad S \to C$

So, our grammar will be

$S \to ABAC \mid BAC \mid ABC \mid AAC \mid BC \mid AC \mid C$

Now, To remove ϵ-production from,

① $A \to aA$

$A \to a$ ($A \to \epsilon$)

② $B \to bB$

$B \to b$ ($B \to \epsilon$)

So, our final production will be,

$S \to ABAC \mid BAC \mid ABC \mid AAC \mid BC \mid AC \mid C$

$A \to aA \mid a$

$B \to bB \mid b$

$C \to c$

Date:

(2) The grammar G is ~~is~~ H

(3) $S \rightarrow ABAC$

$A \rightarrow aA \mid \epsilon$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow cC \mid \epsilon$

→ Step 1:- First we will remove $A \rightarrow aA \mid \epsilon$,

So,

$S \rightarrow \underline{A}BAC$

$S \rightarrow BAC$ $(A \rightarrow \epsilon)$

$S \rightarrow AB\underline{A}C$

$S \rightarrow ABC$ $(A \rightarrow \epsilon)$

$S \rightarrow \underline{A}B\underline{A}C$, $A \rightarrow aA$

$S \rightarrow BC$ $A \rightarrow a$

G1:- $S \rightarrow ABAC \mid BAC \mid ABC \mid BC$

$A \rightarrow aA \mid a$

Step 2 :-

Now, we will remove,

$B \rightarrow bB \mid \epsilon$

$S \rightarrow ABAC$ $S \rightarrow BAC$

$S \rightarrow AAC$ $S \rightarrow AC$

$S \rightarrow BC$

$S \rightarrow a$, $b \rightarrow bB$

$b \rightarrow b$

Ex:- $S \to ABAC \mid BAC \mid ABC \mid BC \mid AAC \mid AC \mid C$

$A \to aAa \mid a$

$B \to bB \mid b$

Step 3 Now, remove

$C \to cC \mid \epsilon$

$S \to ABAC$ $S \to BAC$

$S \to ABA$ $S \to BA$

$S \to BC$ $S \to AAC$

$S \to B$ $S \to A\bar{A}$

$S \to AC$

$S \to A$

$C \to cC \mid \epsilon \to C$

So, our final productions will be,

$S \to ABAC \mid BAC \mid ABC \mid BC \mid AAC \mid AC \mid C \mid AAB \mid BA \mid B \mid AA \mid A$

$A \to aA \mid a$

$B \to bB \mid b$

$C \to cC \mid c$

21/03/2

Page No.
Date :

* UNIT PRODUCTION:-

A grammar is said to be have unit production if there is minimum 1 production of the form

$A \rightarrow B$

The above production $A \rightarrow B$ is useless. Therefore, should be removed from the grammar.

Q Consider the following

G: $S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow C \mid b$

$C \rightarrow D$

$D \rightarrow E$

$E \rightarrow a$

Remove the unit production

→ In the above grammare, there are 3 unit production.

$B \rightarrow C$

$C \rightarrow D$

$D \rightarrow E$

Rule: To remove unit production from the grammar, we have to replace by non-unit production. In the above grammar there is a unit production

$B \rightarrow C$

But there is non-unit production starting with C. Therefore we select

$C \rightarrow D$

In this production, there is no non-unit production starting with D. Therefore we will select

$D \rightarrow E$

In this production, there is a non-unit production starting with E which is

$E \rightarrow a$

Therefore, we will add $E \rightarrow a$ in the grammar, and remove $D \rightarrow E$

Therefore, the grammar will be

G_1: $S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow \epsilon \mid b$

$C \rightarrow D$

$D \rightarrow a$

$E \rightarrow a$

Date :

In, the above grammar G1, there are following two unit production,

$B \rightarrow C$

$C \rightarrow D$

To remove $B \rightarrow C$ we have to add a non-unit production starting with C.

There is no such non-unit production starting with C. Therefore, we will move to next production $C \rightarrow D$.

To remove this production we have to add a non-unit production starting with D.

Since, there is non-unit production starting with D, $D \rightarrow a$

∴ In G1, $C \rightarrow a$ is added and therefore, the modified grammar is as follows.

G2 :- $S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow C \mid b$

$C \rightarrow a$

$D \rightarrow a$

$E \rightarrow a$

In the above grammar G2, there is only non-unit production

$B \rightarrow C$

AND there is non-unit production starting with C, $C \rightarrow a$

To remove $B \rightarrow C$, we have to add a non-unit production called as

$B \rightarrow a$

So, the G2 is modified to G3 as follows

G3: $S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow a \mid b$

$C \rightarrow a$

$D \rightarrow a$

$E \rightarrow a$

Q $S \rightarrow CD$

$A \rightarrow a$

$B \rightarrow E \mid b$

$E \rightarrow F$

$P \rightarrow f$

Remove the unit production from the grammar.

→ There is two unit production in the grammar,

$B \rightarrow E$

$E \rightarrow F$

We will consider, the unit production, $B \rightarrow E$, but there is no, non-unit production starting with E.

Now, we will consider, the unit production, $E \rightarrow F$, there is a non-unit production starting with F

$F \rightarrow f$

so we will replace $f \rightarrow F$ with $E \rightarrow F$. so the grammar will be

$S \rightarrow CD$

$A \rightarrow a$

$B \rightarrow E \mid b$

$E \rightarrow f$

$F \rightarrow f$

Now, there is only one unit production

$B \rightarrow E$

there is a non-unit production starting with E.

So we will replace $B \rightarrow E$ and

Page No.
Date :

the modified grammar would be

$S \rightarrow CD$

$A \rightarrow a$

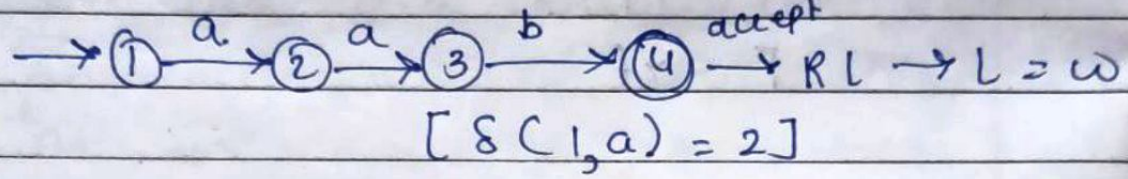
$B \rightarrow f \mid b$

$E \rightarrow f$

$F \rightarrow f$

* PDA - Pushdown Automata:

ω = aab → (Input string)

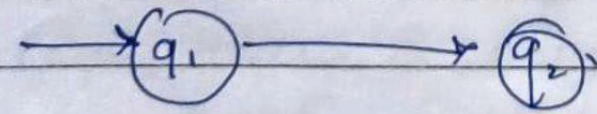


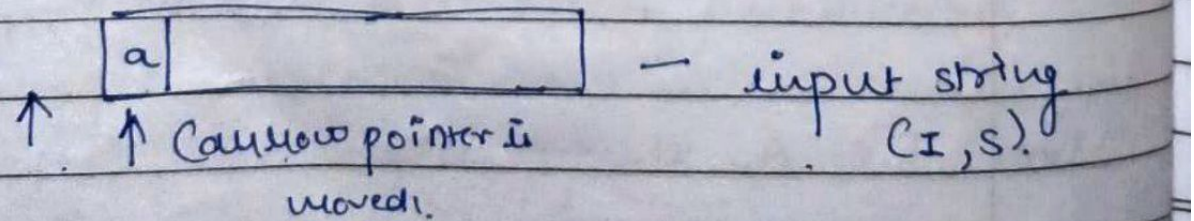
Finite automata uses input string (memory)
In pushdown Automata it uses input string & stack

→ If we want to go from 1 state to another state in (PDA) three things are responsible.

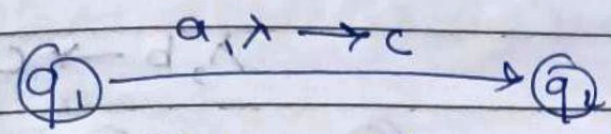
1) Current state
2) Input symbol (string)
3) Pop/push symbol.

a, b → c → action for reaching q_1 to q_2.



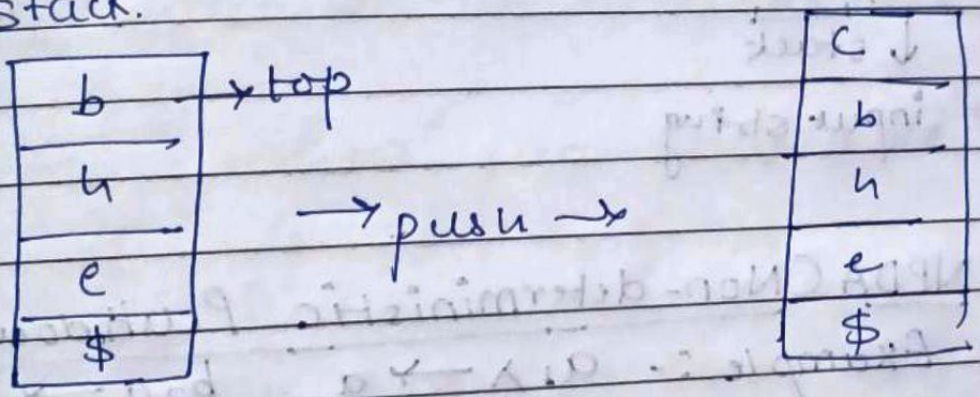


To pop b, b should be on top of stack.



[λ = ϵ]

stack.



Since above b is empty so we push c on top of stack.

for q_1 

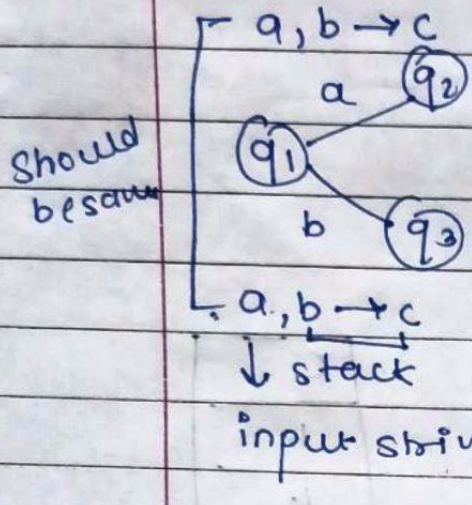
for q_2 

To move from q_1 to q_2 we need (a, λ, c)

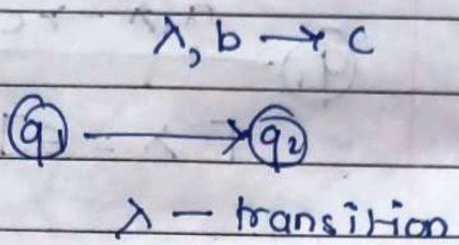
λ is assumed to be present at the top of stack

Page No.
Date :

* Non - Deterministic



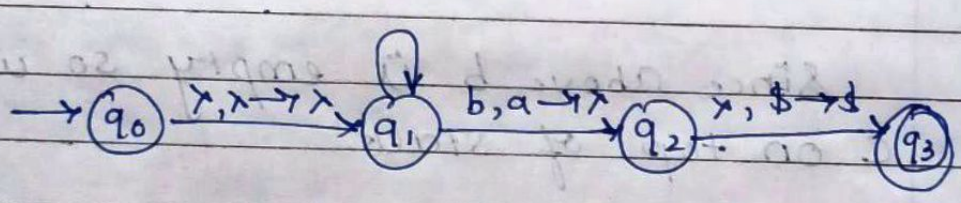
Should be same



λ - transition

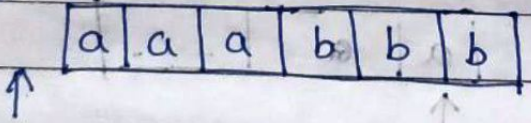
* NPDA (Non-deterministic Pushdown automa

Example :- $a, \lambda \rightarrow a$ $b, a \rightarrow \lambda$



For acceptance of string

Input



The pointer should reach the end of input string.

Two condition are accepted.

1) Pointer should reach on the end of I.S & the automata should reach at final state (state may or may not be empty

2) Pointer should reach on the end of IS & the automata doesn't reach to the final state but the stack should be empty.

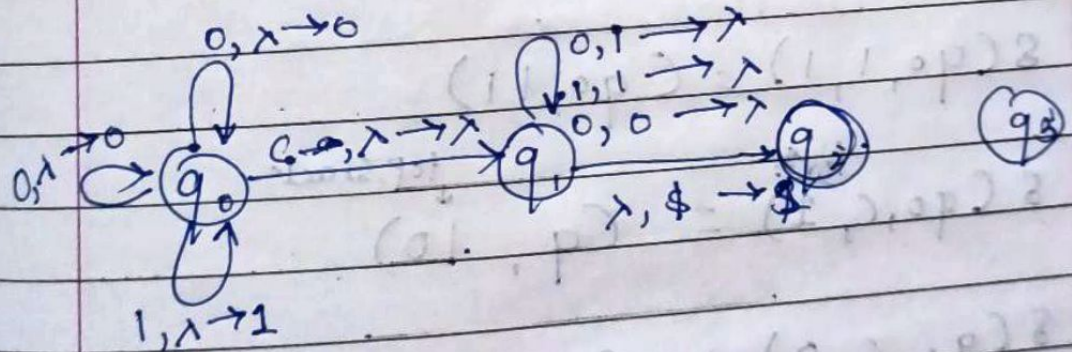
Q. Construct a PDA accepting :-

$L = \{ w c w^R / w$ is in $(0/1)^*$ and w^R is reverse of $w \}$

→ Consider a w

$w = \{010 c 010\}$

0	1	0	0	1	0



$\delta(q_0, 0, Z_0) = (q_0, Z_0)$

State IS stack) stack top.

For those strings starting with '0' or '1'

Date :

$\delta(q_0, 0, Z_0) = (q_0, 0Z_0)$

$\delta(q_0, 1, Z_0) = (q_0, 1Z_0)$

Next symbol (if I.S is starting with '0'

$\delta(q_0, 1, 0) = (q_0, 10)$.

0	1	C	1	0

$\delta(q_0, 0, 0) = (q_0, 00)$

For R.S

1	0	C	0	1

$\delta(q_0, 0, 1) = (q_0, 01)$.

$\delta(q_0, 1, 1) = (q_0, 11)$

for 1 ↓ top stack

$\delta(q_0, C, 1) = (q, 10)$

$\delta(q_0, C, 0) = (q_1, 01)$

$\delta(q_1, 1, 1) = (q_1, \epsilon)$

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

Page No.
Date :

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

q_2 is final state

$[\delta(q_0, \in, z_0) = (q_2, z_0)]$

5/09/202

Page No.
Date :

* Conversion of CFG to PDA.

Mathematical representation of PDA

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

$Q \rightarrow$ set of states $= \{q_0, q_1, q_2\}$
$\Sigma \rightarrow$ set of input string symbols
$\Gamma \rightarrow$ set of stack symbols
$\delta \rightarrow$ transition function which will map $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma^* \rightarrow Q \times \Gamma^*$
$q_0 \rightarrow$ start symbol / initial state
$z_0 \rightarrow$ initial stack symbol
$F \rightarrow$ set of final states.

CFG to PDA

(1) Construct a PDA for the following CFG

G :- $S \rightarrow aAA$
$A \rightarrow aS \mid bS \mid a$

Page No.
Date :

→ Step 1:- Mathematical representation of PDA is as follows.

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

Mathematical representation of PDA for above CFG will be as follows

$M = (\{q\}, \{a, b\}, \{S, A\}, \delta, \{q\}, S, \{q\})$

Step 2:- The transition function will be as follows.

Consider the first production of CFG

$S \rightarrow aAA$ ---(1)

$\delta(q, a, S) \rightarrow (q, AA)$

$A \rightarrow aS$ ---(2)

$\delta(q, a, A) \rightarrow (q, S)$

$A \rightarrow bS$ ---(3)

$\delta(q, b, A) \rightarrow (q, S)$

$A \rightarrow a$ ---(4)

$\delta(q, a, A) \rightarrow (q, \epsilon)$

Date :

$\delta(q, a, S) \rightarrow (q, AA)$

$\delta(q, a, A) \rightarrow (q, S)$

(Stubby) $\delta(q, b, A) \rightarrow (q, S)$

$\delta(q, a, A) \rightarrow (q, \epsilon)$

$\delta(q, a, A) \rightarrow \{(q, S), (q, \epsilon)\}$

∴ Final transition would be.

$\delta(q, a, S) \rightarrow (q, AA)$

$\delta(q, a, A) \rightarrow \{(q, S), (q, \epsilon)\}$

$\delta(q, b, A) \rightarrow (q, S)$

(2) Consider the following CFG

$S \rightarrow bBD$

$B \rightarrow eB \mid fB \mid g$

→ Step 1:- Mathematical representation is

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

Mathematical representation of following CFG will be.

$M = (\{q\}, \{b, e, f, g\}, \{S, B, D\}, \delta, \{q\}, \{S\}, \{\phi\})$

Page No.
Date :

Step 2:- The transition function will be

$S \rightarrow bBD$

$\delta(q, b, S) \rightarrow (q, BD)$ — (1).

$B \rightarrow eB$

$\delta(q, e, B) \rightarrow (q, B)$ — (2).

$B \rightarrow fB$

$\delta(q, f, B) \rightarrow (q, B)$ — (3)

$B \rightarrow g.$

$\delta(q, g, B) \rightarrow (q, \in)$ — (4).

Final transition will be.

$\delta(q, b, S) \rightarrow (q, BD)$

$\delta(q, e, B) \rightarrow (q, B)$

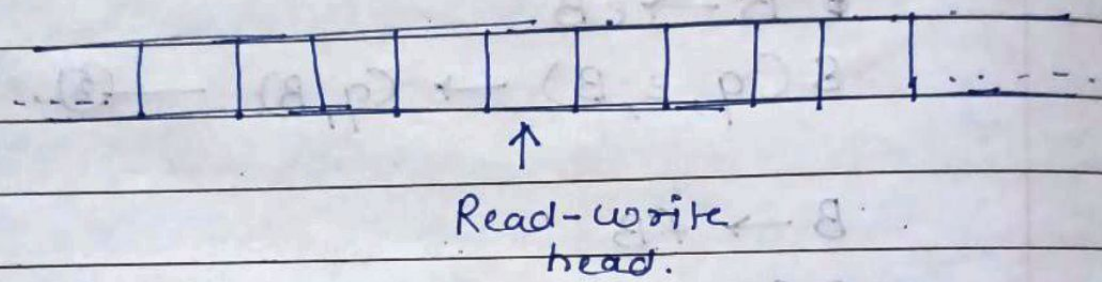
$\delta(q, f, B) \rightarrow (q, B)$

$\delta(q, g, B) \rightarrow (q, \in)$

Turing Machine

Tape

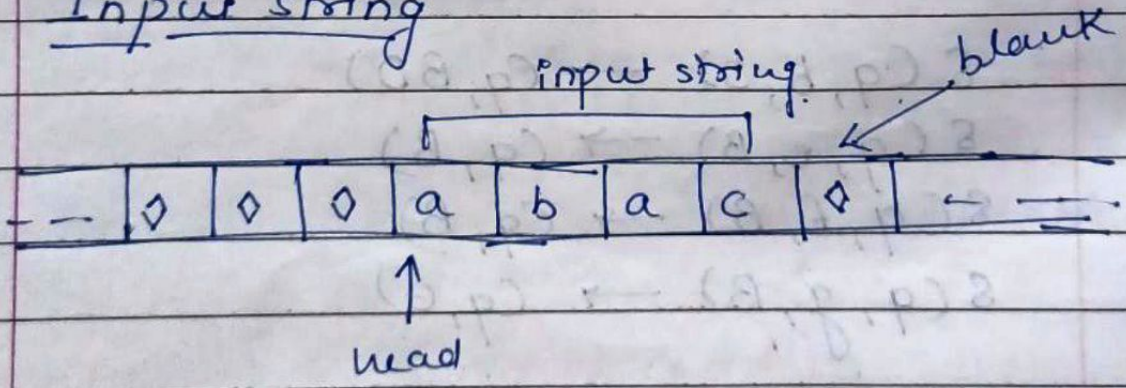
Infinite ~~tape~~ boundary.



The head at each step.

1. Read a symbol
2. Write a symbol
3. Moves left or right.

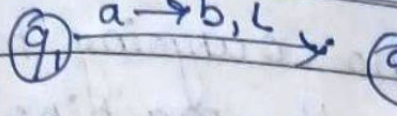
Input string

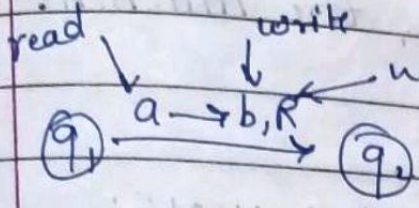


1. Input string is never blank.
2. Head at leftmost symbol.

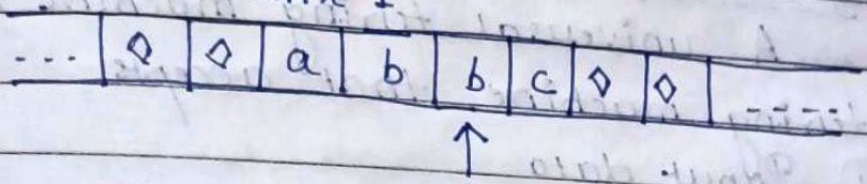
Page No.
Date :

State & Transition.

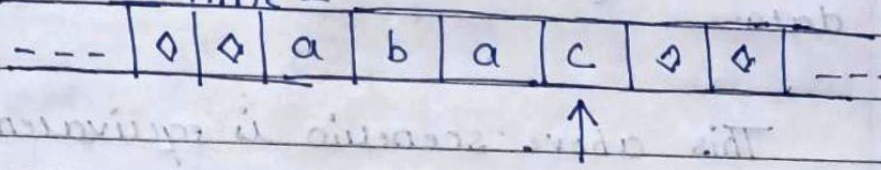




Time 1



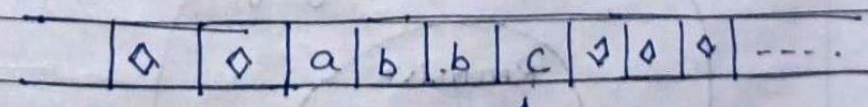
Time 2

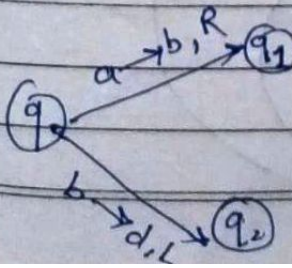


Turing Machine are deterministic.

Halting :-

- When there is no transition.





Page No.
Date :

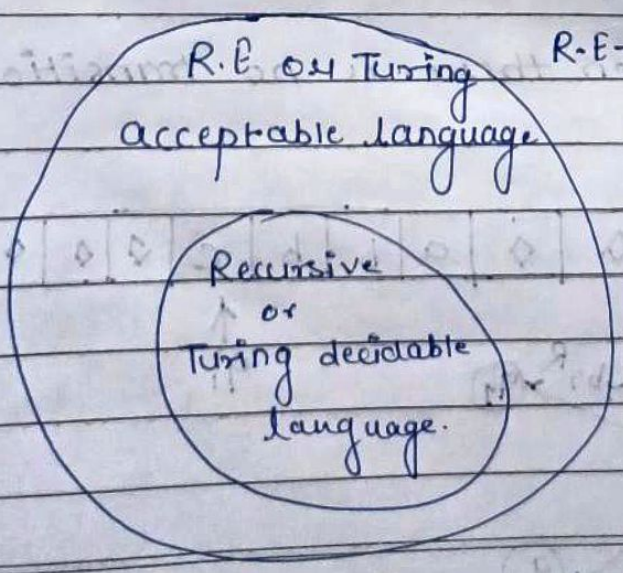
① Final state have no outgoing transition.
② In the final state, the machine halts

4/2022

* UNIVERSAL TURING MACHINE :-

A universal turing machine is a turing machine that accepts
① Input data
② A discription of computation (ie algorithm) to work on input data.

This above scenario is equivalent to the general purpose computer. This Therefore, the universal turing machine is powerful enough to stimulate the behaviour of the arbitrary turing machine.



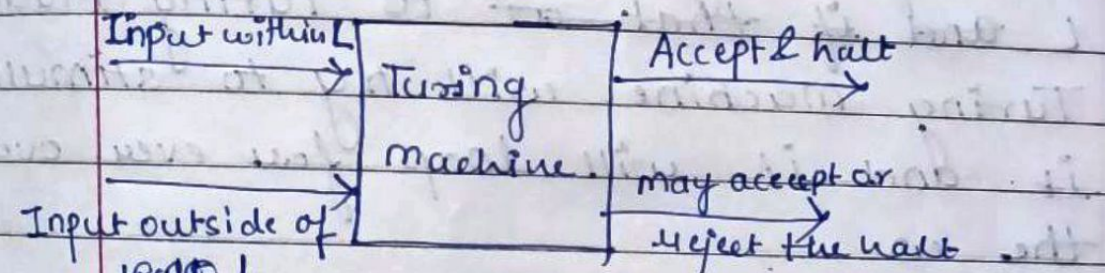
R.E → Recursive Enumerable)

Date :

① Recursive Enumerable (Outerset)
② Recursive Language (Innerset)

* Recursive Enumerable Language :

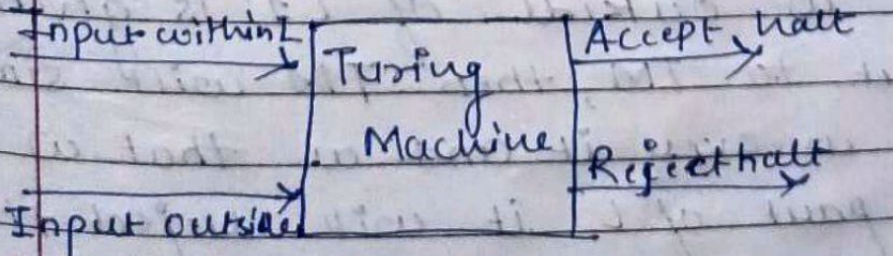
Let, $L = \{ w_1, w_2, w_3, w_4 \dots \}$



↳ enters into loop forever.

* Recursive Language

$L = \{ w_1, w_2, w_3, w_4, w_5, \dots \}$



Recursive enumerable languages are those language for which we can design turing Machine in such a way that if we give any w from L the turing machine will stimulate the w over it and accept the w and reach to final state and halt. But if we give any w outside of L ~~and if that w~~ to Turing Machine, Turing machine will try to stimulate it and it will loop for ever over the w.

Recursive languages are those language for which we can design Turing machine in such a way that if we give and w which is a part of L1, TM will stimulate over that w, reach to final state and halt (accept halt or YES). If any w outside of L1 is given as input to TM, then TM will simulate the w over it and as that w is not part of L it will reject that w and halt.