BEIT403T **THEORY OF COMPUTATION**

(Theory Credit: 05)

Teaching Scheme:
Lecture: 4 Hours/week
Tutorial: 1 Hour/week

Examination Scheme:
Theory: T (U): 80 Marks T (I): 20 Marks
Duration of University Exam. : 03 Hours

==========

UNIT I:

Strings, Alphabet, Language operations, Finite state machine definitions, Finite automation model, Acceptance of strings and language, Non deterministic finite automation, Deterministic finite automation, Equivalence between NFA and DFA, Conversion of NFA into DFA, Minimization of FSM, Equivalence between two FSM's Moore and Mealy machines

UNIT II:

Regular sets, Regular expressions, Identity rules, Manipulation rules, Manipulation of regular expressions, Equivalence between RE and FA, Inter conversion, Pumping lemma, Closure properties of regular sets(proofs not required), Chomsky hierarchy of languages, Regular grammars, Right linear and left linear grammars, Equivalence between regular linear programming and FA, Inter conversion between RE and RG.

UNIT III:

Context free grammar, Derivation trees, Chomsky normal form, Greibach normal form, Push down automata, Definition, Model acceptance of CFL, Equivalence of CFL and PDA, Inter conversion, Closure properties of CFL(Proofs omitted),Pumping Lemma of CFL, Introduction of DCFL and DPDA

UNIT IV:

Turing Machine: Definition, Model of TM, Design of TM, Universal Turing Machine, Computable function, Recursive enumerable language, Types of TM's (proofs not required), Linear bounded automata and Context sensitive language, Counter machine

UNIT V:

Decidability and Undecidability of problems, Properties of recursive & recursively enumerable languages, Halting problems, Post correspondence problem, Ackerman function, and Church's hypothesis.

UNIT VI:

Recursive Function: Basic functions and operations on them, Bounded Minimalization, Primitive recursive function, μ-recursive function, Primitive recursive predicates, Mod and Div functions, Unbounded Minimalization, Equivalence of Turing Computable function and μ-recursive function.

Text Books:

1. Introduction to Automata Theory, Languages and Computation by J. E. Hopcraft,R. Motwani, J. D Ullman, second Edition, Pearson Education, Aisa
2. An Introduction to Formal Languages and Automata by Peter Linz

Scanned by CamScanner

Unit - 1

1) String :-

A string is a finite sequence of symbol & we use a letter w to denote a string.

2) Length of the string :-

If w is the string, then length of the string is denoted by mod of w i.e. |w| & it is a count of number of symbols in a string.

e.g. If $w = XYZ$
$\therefore |w| = 3$

3) Empty String :-

If $|w| = 0$, then string is called as an empty string & we use ϵ (epsilon) to denote empty string

4) Prefix :-

If x is some string then any string formed by discarding 0 or more trailing symbols of x is called as prefix of x.

e.g :- $x = abcde$
then $x = abc$ is a prefix of abcde

5) Suffix :-

A suffix of x is a string formed

Scanned by CamScanner

by deleting 0 or more of the leading symbols of x.

e.x :-

cde is a suffix of abcde.

6) Substring :-

A substring of x is any string obtained by deleting a prefix & a suffix from x. Moreover, any prefix or suffix of x is a substring of x but a substring didn't be a prefix or suffix.

Ex. cd is a substring of abcde but not a prefix or suffix.

7) Proper prefix :-

Let, w = xy be a string then x is a proper prefix if y is not empty.

8) Proper suffix :-

Let, w = xy be a string then y is proper trail or suffix. If x is non-empty.

9) α - bit :-

It is a finite set of symbol denoted by letter Σ (summation)

Scanned by CamScanner

10) Concatenation :-

If x & y are strings then the concatenation of x & y written $x \cdot y$ or xy. is the string formed by following symbol of x by the symbol of y.

ex. If $x = abc$ & $y = de$
where, a, b, c, d, e are symbols then

$$xy = abcde$$

The concatenation of the empty string with any string is ϵ $x = x \cdot \epsilon = x$.

11) Language :-

It is a set of string formed from some specific alphabets. Simple sets such as ϕ. The empty sets having no members, the set containing only the empty string are languages under this defination.

If L & M are languages then LM is a language consisting of all string xy which can be formed by selecting a string M & concatenating them in that order.

$LM = \{ xy \mid x \text{ is in } L \text{ \& } y \text{ is in } M \}$

ex. $L = \{a, b\}$ $M = \{c, d\}$

$LM = \{ac, ad, bc, bd\}$

Scanned by CamScanner

* FINITE AUTOMATA

A finite automata consists of finite number of states & finite number of transition & these transition are defined on certain specific symbols. called as input symbols. One of the states defined as initial state & one of the state is defined as final state in finite automata.

The 5 tuples form of finite automata is given below

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

where, Q - Number of states
Σ = Input symbols
δ = transitions specifying from which state on which input symbol where the transition goes.
q_0 = Initial state
F = final state

Scanned by CamScanner

* Deterministic Finite Automata (DFA)

Q.1) Construct a DFA that is formed by 0 & 1 and the DFA should start with 0.

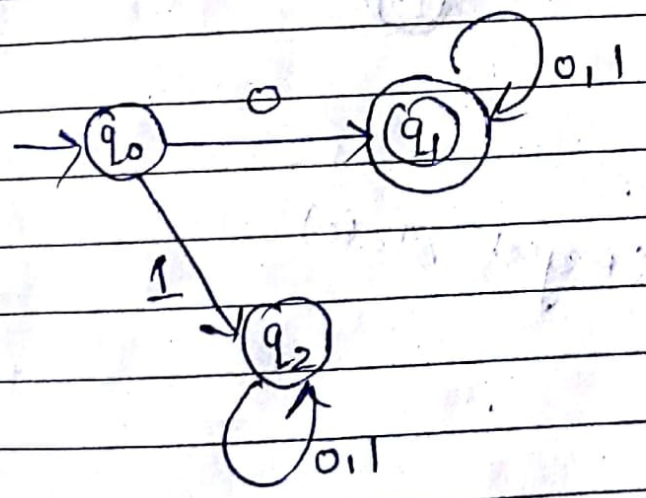
⇒ Transition table

Q \ Σ	0	1
Σ → q_0	q_1	q_2
0 * q_1	q_1	q_1
(Reject State) q_2	q_2	q_2

→ Reject state

Σ0 ✓
Σ1 ✗
00 ✗
01 ✗
} but it should start with 0

DFA



① String → 0101000

$= \delta(\delta(q_0, 0101000))$
$= \delta(\delta(q_0, 0)\, 101000)$
$= \delta(\delta(q_1, 1)\, 01000)$
$= \delta(\delta(q_1, 0)\, 1000)$
$= \delta(\delta(q_1, 1)\, 000)$
$= \delta(\delta(q_1, 0)\, 00)$
$= \delta(\delta(q_1, 0)\, 0)$
$= \delta(\delta(q_1, 0))$
$= q_1$

↳ got final state (string accepted)

② String → 10101

$= \delta(\delta(q_0, 10101))$
$= \delta(\delta(q_0, 1)\, 0101)$
$= \delta(\delta(q_2, 0)\, 101)$
$= \delta(\delta(q_2, 1)\, 01)$
$= \delta(\delta(q_2, 0))$
$= \delta(\delta(q_2, 1))$
$= q_2$ → non-final state (not accepted)

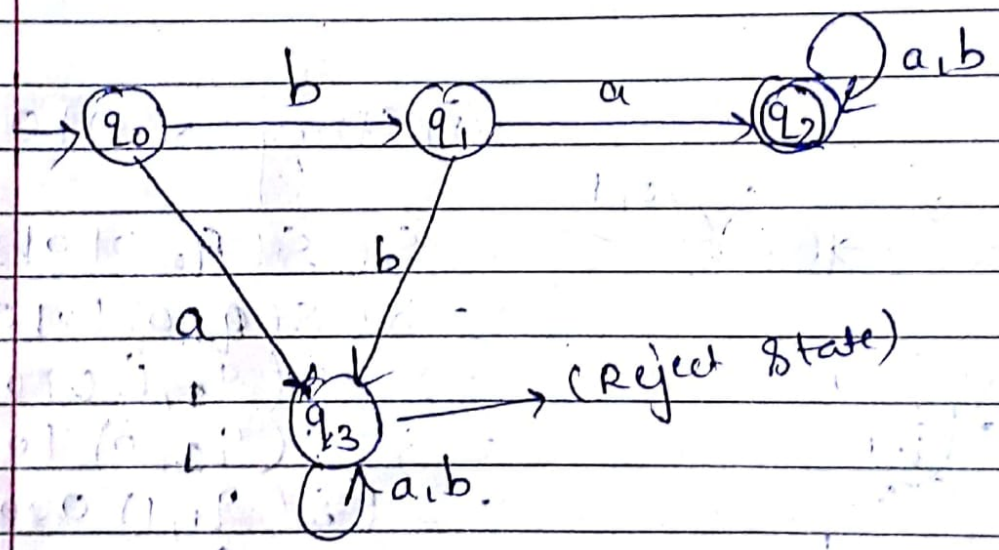
Scanned by CamScanner

Q. 2) Construct a DFA that is formed by a & b and the DFA should start with ba.

→ q_0 contain → ε
q_1 —"— → b
* q_2 —"— → ba

→ Transition table

	Q \ Σ	a	b
ε →	q_0	q_3	q_1
b	q_1	q_2	q_3
ba *	q_2	q_2	q_2
(Reject state)	q_3	q_3	q_3



$= \delta(\delta(q_0, baaba)$
$= \delta(\delta(q_0, b)\ aaba)$
$= \delta(\delta(q_1, a)\ aba)$
$= \delta(\delta(q_2, a)\ ba)$
$= \delta(\delta(q_2, b)\ a)$
$= \delta(\delta(q_2, a)$
$= \delta(q_2)$
↳ final state

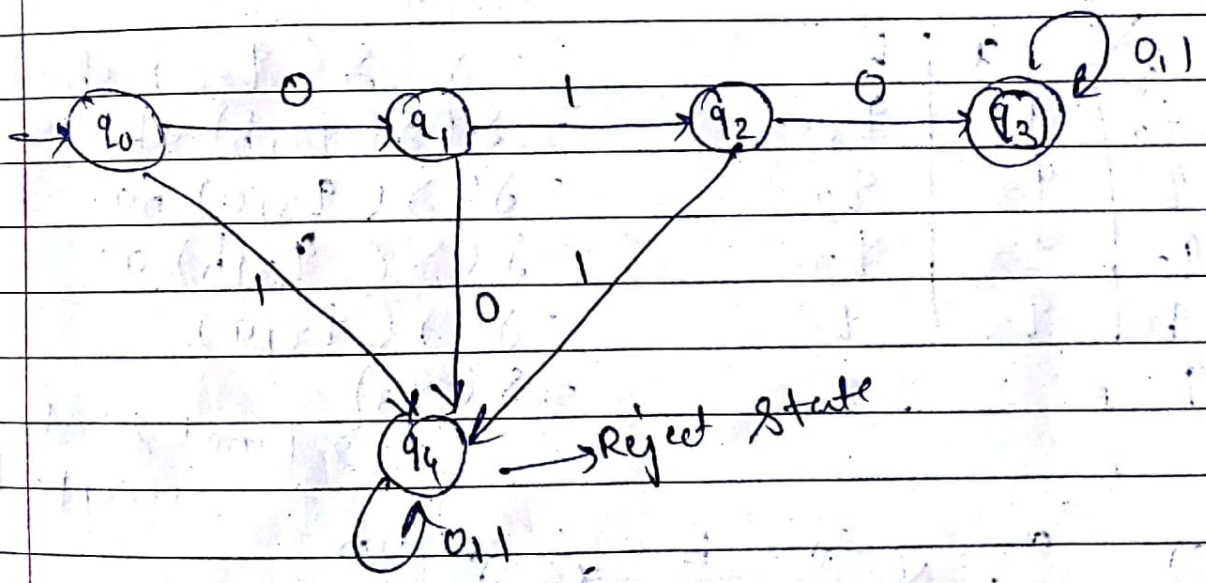
Scanned by CamScanner

Q.3) String start with 010.

⇒ Transition table

Q \ Σ	0	1
Σ → q_0	q_1	q_4
0 q_1	q_4	q_2
01 q_2	q_3	q_4
010 * q_3	q_3	q_3
(Reject state) q_4	q_4	q_4

→ q_0 = Σ
q_1 = 0
q_2 = 01
* q_3 = 010



Q. 4) String not and start with ab.

⇒ 1) Construct for string start with ab

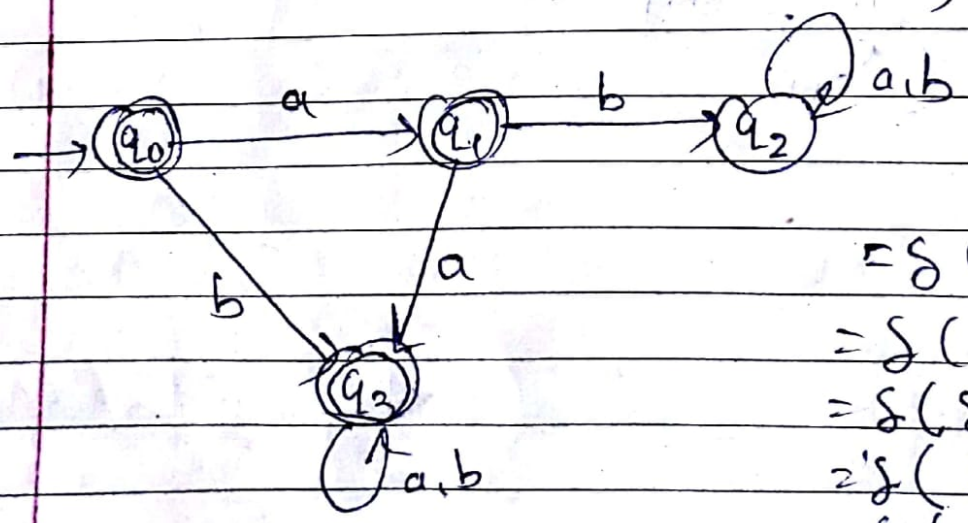
Q	ε	a	b
ε → q_0	q_1	q_3
a q_1	q_3	q_2
ab * q_2	q_2	q_2
(Rej state) q_3	q_3	q_3

$q_0 = \varepsilon$
$q_1 = a$
$q_2 = ab$

2) Change non-final state as final state & final state as non-final state.

Q	ε	a	b
ε * q_0	q_1	q_3
a * q_1	q_3	q_2
ab q_2	q_2	q_2
(Rej state) * q_3	q_3	q_3

$= \delta(\delta(q_0, baba))$
$= \delta(\delta(q_0, b)\, aba)$
$= \delta(\delta(q_3, a)\, ba)$
$= \delta(\delta(q_3, b)\, a)$
$= \delta(\delta(q_3, a))$
$= \delta(q_3)$
↳ Final state accepted



$= \delta(\delta(q_0, abb))$
$= \delta(\delta(q_0, a)\, bb)$
$= \delta(\delta(q_1, b)\, b)$
$= \delta(\delta(q_2, b))$
$= \delta(q_2)$
↳ non-final state (not accepted)

Scanned by CamScanner

Q.5) String not Start with 1101

⇒

1) Construct for String Start with 1101.

Q	ε	0	1
ε → q_0	q_5	q_1	
1 q_1	q_5	q_2	
11 q_2	q_3	q_5	
110 q_3	q_5	q_4	
1101 * q_4	q_4	q_4	
(R.S) q_5	q_5	q_5	

→ $q_0 = \varepsilon$

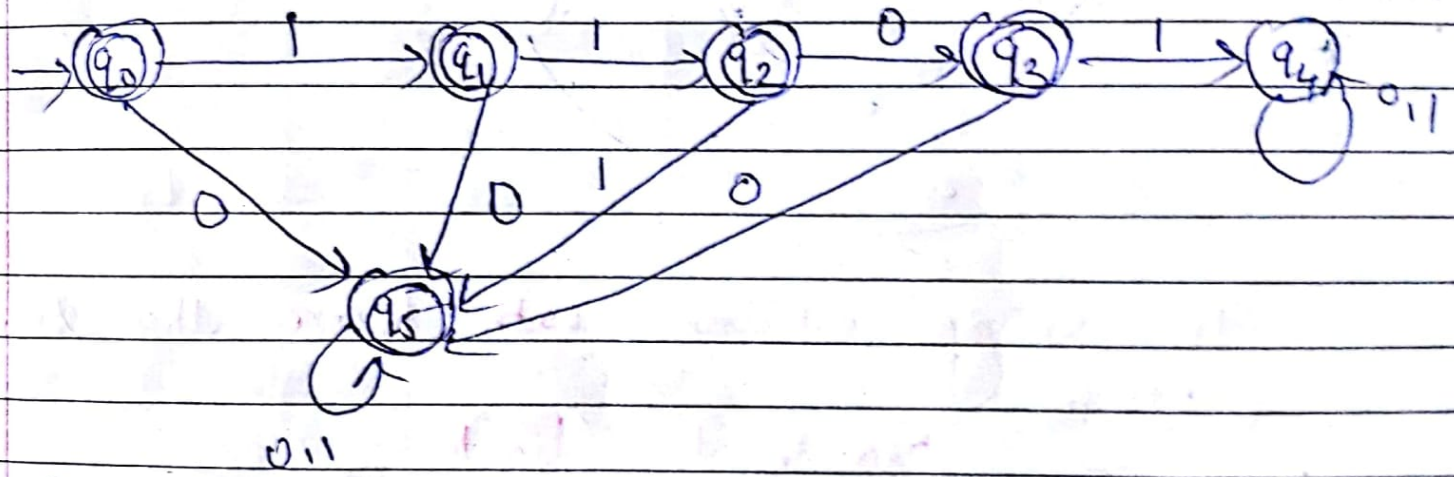
$q_1 = 1$

$q_2 = 11$

$q_3 = 110$

* $q_4 = 1101$

2) for not Start with 1101 — make nonfinal = final, final = non-final



Q.6) String end with 1

⇒ Transition table

→ $q_0 = \varepsilon$

* $q_1 = 1$

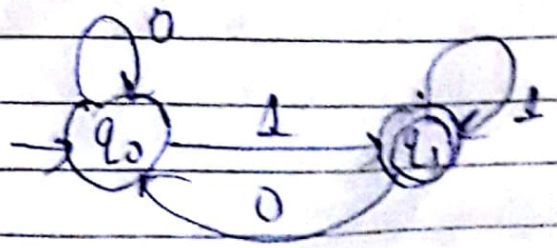
Q	ε	0	1
ε → q_0	q_0	q_1	
1 * q_1	q_0	q_1	

~~ε~~ ~~0~~ → q_0

~~ε~~ ~~1~~ → q_1

~~1~~ ~~0~~ → q_0

~~1~~ ~~1~~ →



Scanned by CamScanner

Q. 7) String end with baa.

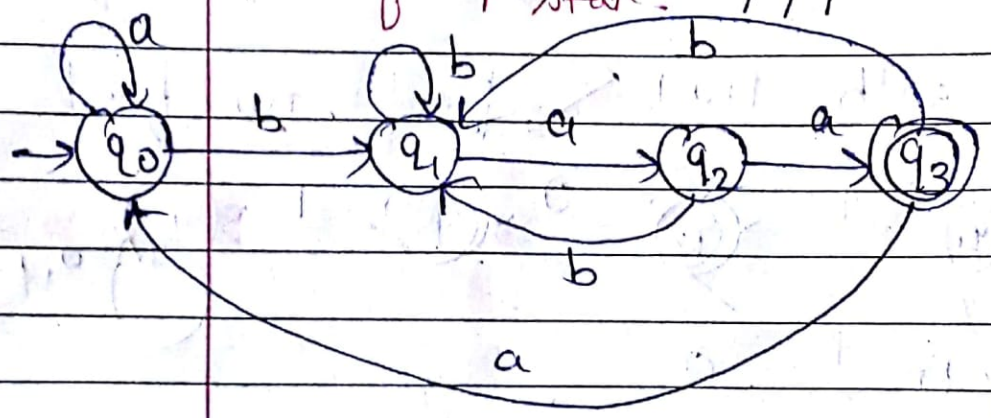
→ Transition table

Q \ Σ	a	b
Σ → q_0	q_0	q_1
b q_1	q_2	q_1
ba q_2	q_3	q_1
baa q_3	q_0	q_1

In final state check for previous state.

~~Σ a~~ → q_0
~~Σ b~~ → q_1
ba → q_2
~~b~~b → q_1
baa → q_3
~~b~~~~a~~b → q_1
~~b~~~~a~~~~a~~~~a~~ → q_0
~~b~~~~a~~~~a~~b → q_1

If not contain then goes to Initial state only.



If string not end with baa then simply change

non-final = final
final = non-final

all states should be same

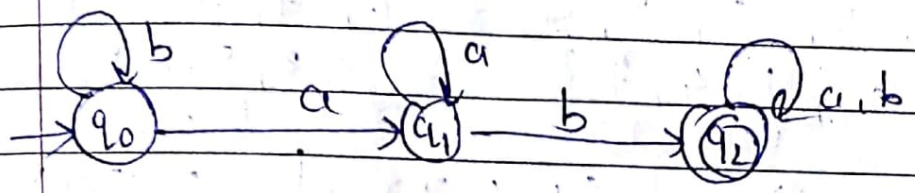
Q.8) String contain "ab" as substring

→ Transition table

Q \ Σ	a	b
→ ε q_0	q_1	q_0
a q_1	q_1	q_2
* ab q_2	q_2	q_2

$\cancel{\varepsilon}\, a = q_1$
$\cancel{\varepsilon}\, b = q_0$
$\cancel{a}\, a \rightarrow q_1$
$ab = q_2$

→ In final state it goes to final state only



Q.9) Construct a DFA that accepting decimal number divisible by 2.

→ $\Sigma^* = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

If you divide any number by 2 it will give remainder either 0 or 1.

$= \{0, 2, 4, 6, 8\}$ (Rem = 0) $= \{1, 3, 5, 7, 9\}$ (Rem = 1)

Q \ Σ	0 (0, 2, 4, 6, 8)	1 (1, 3, 5, 7, 9)
→ * (Rem = 0) q_0	q_0	q_1
(Rem = 1) q_1	q_0	q_1

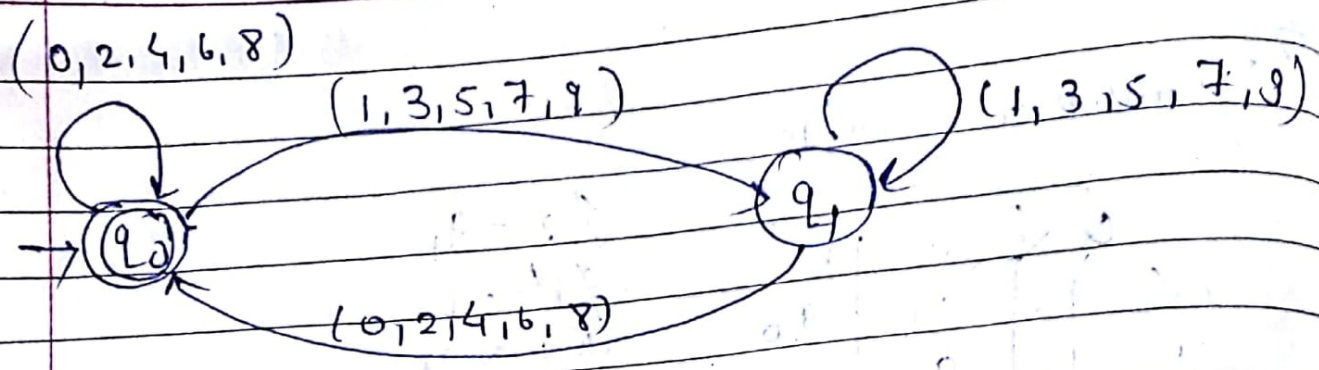
$\frac{00}{2} = 0$

$\frac{02}{2} = 0$

$\frac{10}{2} = 0$ $\frac{11}{2} = 1$

$\frac{14}{2} = 0$ $\frac{13}{2} = 1$

Scanned by CamScanner

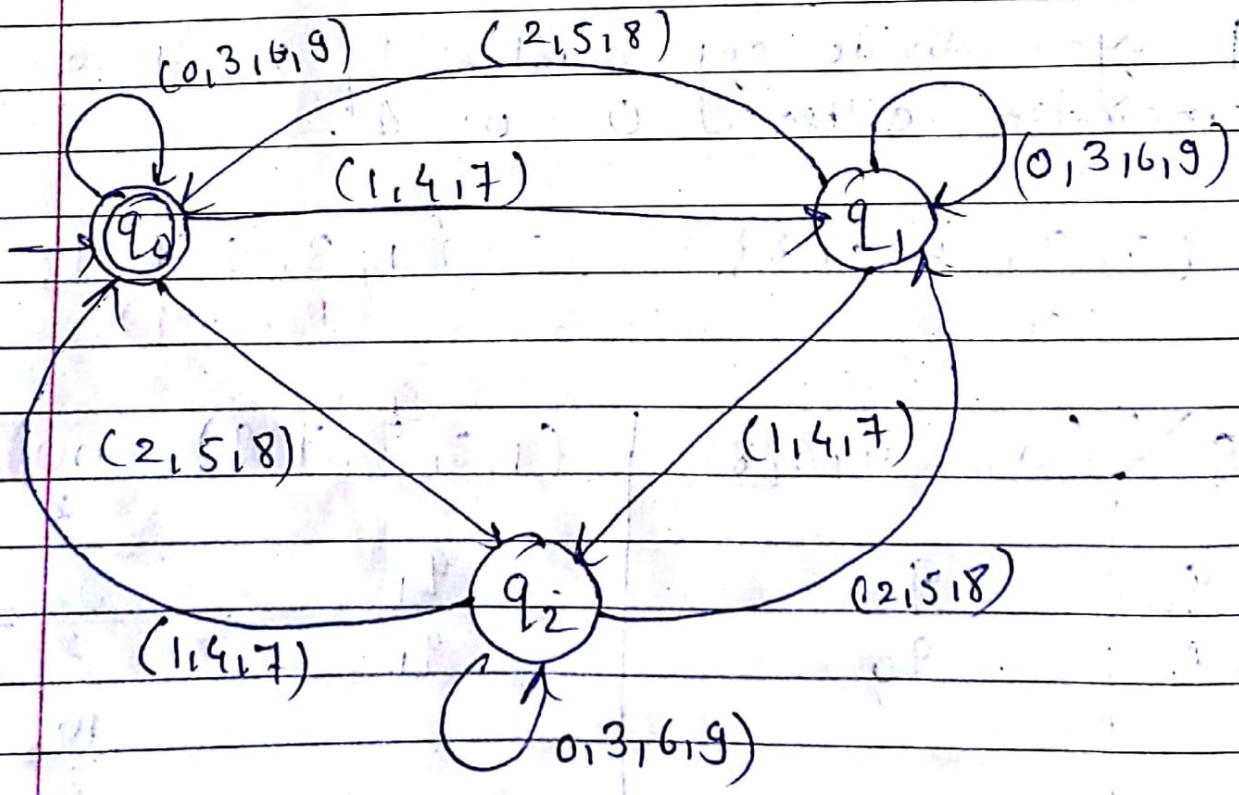


Q. 10) decimal number is divisible by 3.

→ $\Sigma^* = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$= \{0, 3, 6, 9\}$ (Rem = 0) $= \{1, 4, 7\}$ (Rem = 1) $= \{2, 5, 8\}$ (Rem 2)

$Q \backslash \Sigma$	(0,3,6,9) (Rem 0)	(1,4,7) (Rem 1)	(2,5,8) (Rem 2)
(0) q_0	q_0	q_1	q_2
(1) q_1	q_1	q_2	q_0
(2) q_2	q_2	q_0	q_1



Scanned by CamScanner

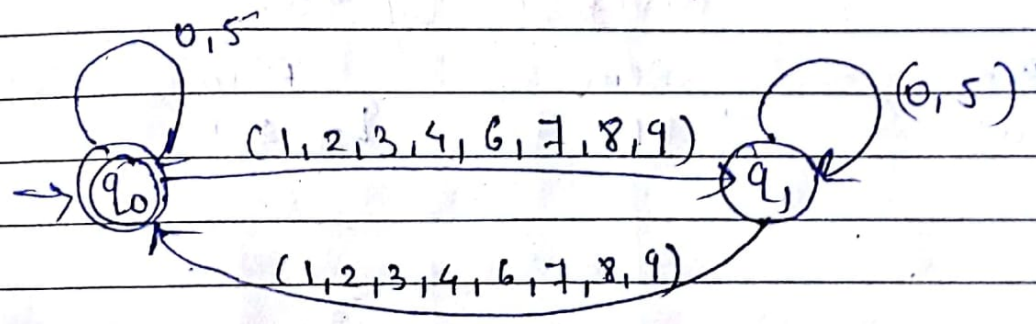
Q.11) decimal no. is divisible by 5

→ $\Sigma^* = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$= \{0, 5\}$ $\{1, 2, 3, 4, 6, 7, 8, 9\}$

Rem = 0 Rem = 1

	Q	Σ (0, 5)	(1, 2, 3, 4, 6, 7, 8, 9}
(0)	q_0	q_0	q_1
(1)	q_1	q_0	q_1



Q.12) Construct a DFA formed that 0 & 1 & string should (binary Number) is divisible by 2.

→ $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

0 0 0 → 0
0 0 1 → 1
0 1 0 → 2
0 1 1 → 3
1 0 0 → 4
1 0 1 → 5
1 1 0 → 6
1 1 1 → 7

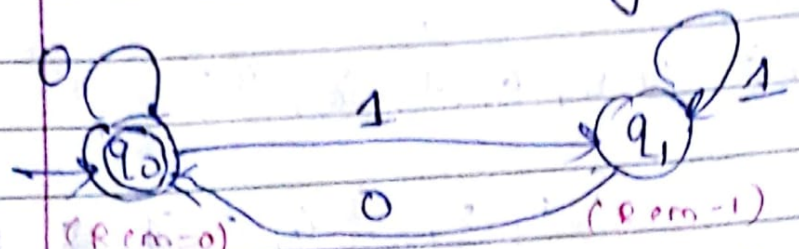
① If any number is divisible by 2 it give rem = 0 & rem = 1

② So, it has 2 states (q_0, q_1).

Scanned by CamScanner

If remainder 0 then no. is divisible.

If any number is divisible by 2, 3, 4, 5, 6, ...
then it has always remainder 0.



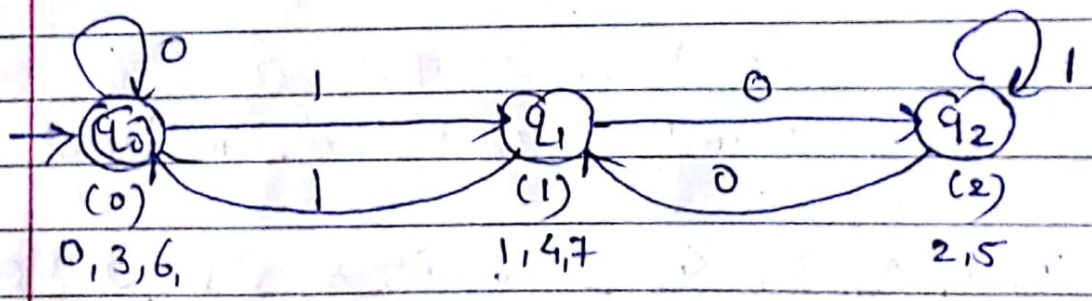
Q \ Σ	0	1
* → (0) q_0	q_0	q_1
(1) q_1	q_0	q_1

0 → 000
2 → 010
4 → 100
6 → 110

1 → 001
3 → 011
5 → 101
7 → 111

Q.13) Construct a DFA formed by 0 & 1 & binary number divisible by 3.

⇒ q_0 (0) Rem, q_1 (1) Rem, q_2 (2) Rem



Q \ Σ	0	1
* → (0) q_0	0	q_1
(1) q_1	q_2	q_0
(2) q_2	q_1	q_2

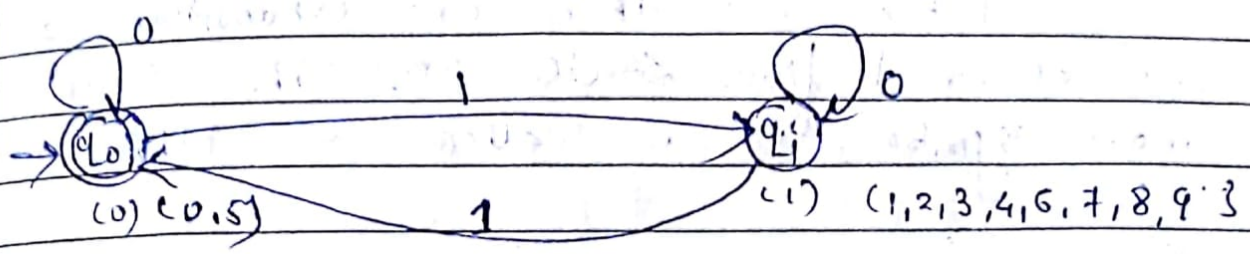
Scanned by CamScanner

(6M)

5. ~~Non-DFA~~

Q.14) Design DFA to check whether the given binary number is divisible by 5.

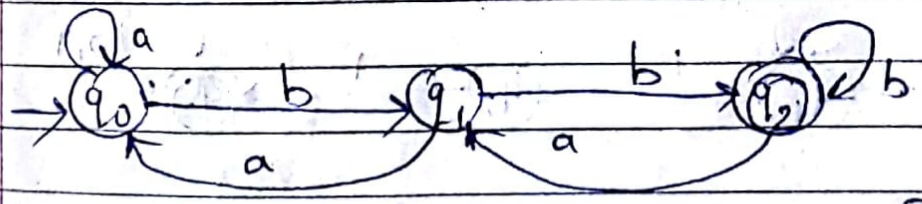
⇒



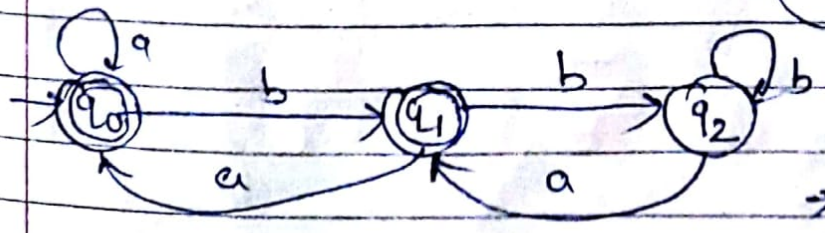
Q	Σ	0	1
→ * (0)	q_0	q_0	q_1
(1)	q_1	q_1	q_0

Q.15) Consider the language L of all strings of a's & b's that do not end with "b" & do not contain the substring "bb". Find a finite language S so that L = S*.

⇒ Construct for contain 'bb' as substring & end with 'b'.



• Not contain "bb" as substring & not end with "b"



Q	Σ	a	b
→ *	q_0	q_0	q_1
*	q_1	q_0	q_2
	q_2	q_1	q_2

Scanned by CamScanner

Non - Deterministic Finite Automata

NFA -

When more than one transition are transformed from states on the same input symbol, it is called as NFA.

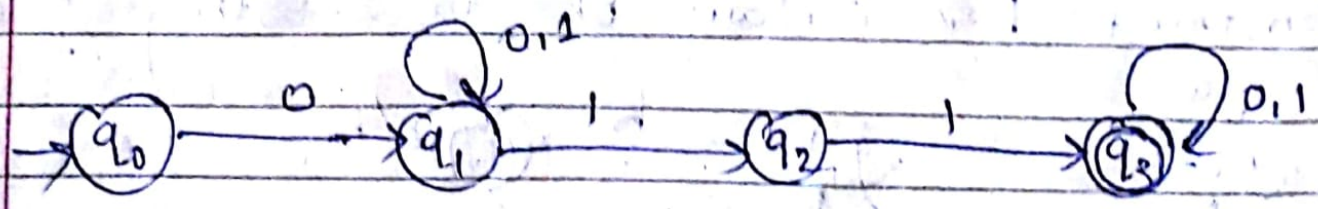
Ex

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

where δ,

States	0	1
$\rightarrow q_0$	q_1	ϕ
q_1	q_1	q_1, q_2
q_2	ϕ	q_3
$* q_3$	q_3	q_3

$\Rightarrow$



Scanned by CamScanner

Q.1) Construct DFA for following NFA

States	0	1
→ q_0	q_1	ϕ
q_1	q_1	q_1, q_2
q_2	ϕ	q_3
* q_3	q_3	q_3

→

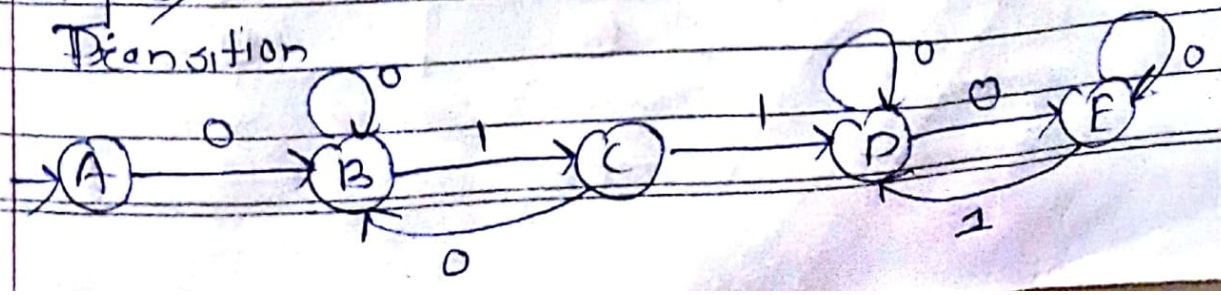
Step 1)

States	0	1
A q_0	B q_1	ϕ
B q_1	B q_1	C q_1, q_2
C q_1, q_2	B q_1	D q_1, q_2, q_3
* D q_1, q_2, q_3	E q_1, q_3	D q_1, q_2, q_3
* E q_1, q_3	E q_1, q_3	D q_1, q_2, q_3

Step 2)
Transition table

States	0	1
A	B	ϕ
B	B	C
C	B	D
* D	E	D
* E	E	D

Step 3)
Transition



Scanned by CamScanner

9.15 (i)

Q. 21 Convert the following NFA to DFA & informally describe the language it accepts.

	0	1
→ p	q, s	q
* q	r	q, r
r	s	p
* s	–	p

State p be the initial state & state q & r be the final state.

⇒ Step 1>

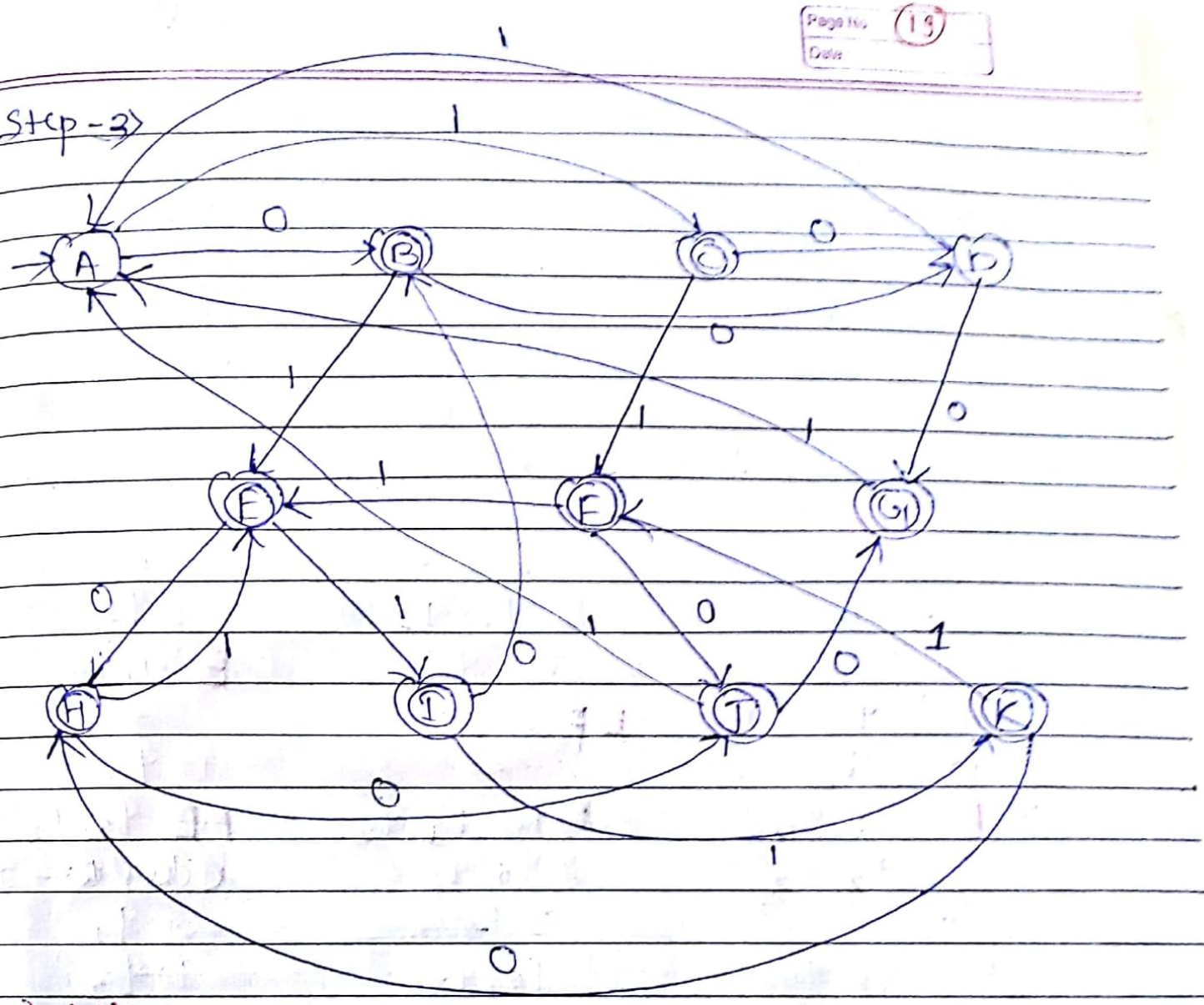
States	0	1
→ A p	B q, s	C q
* B q, s	D r	E p, q, r
* C q	D r	F q, r
D r	G s	A p
* E p, q, r	H q, r, s	I p, r, s
* F q, r	J r, s	E q, r, p
* G s	–	A p
* H q, r, s	J r, s	E p, q, r
* I p, r, s	B q, s	K q, p
* J r, s	G s	A p
* K p, q	H q, s, r	F q, r

Step 2>

Transition table

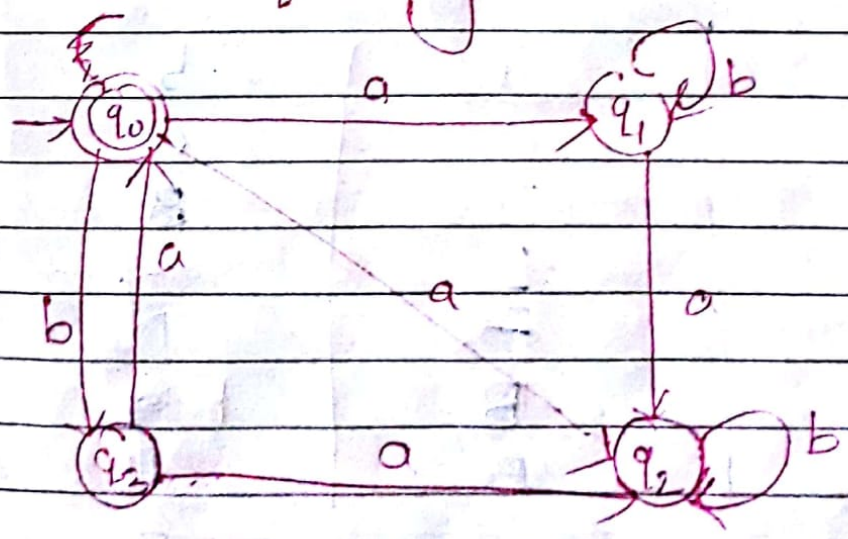
Scanned by CamScanner

Step-3)



(S-17) (7)

Q.3) Convert the following NFA into equivalent DFA



Scanned by CamScanner

⇒ Step 1)

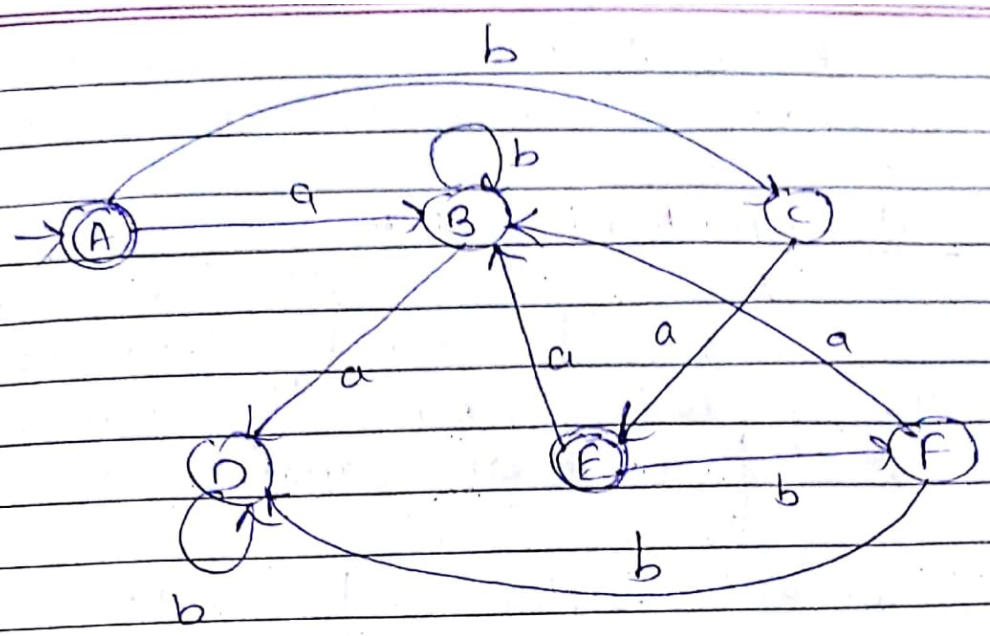
States	a	b
* → q_0	q_1, q_2	q_3
q_1	q_2	q_1
q_2	–	q_2
q_3	q_0, q_2	–

Step 2)

States	a	b
* → A q_0	B q_1, q_2	C q_3
B q_1, q_2	D q_2	B q_1, q_2
C q_3	E q_0, q_2	–
D q_2	–	D q_2
* F q_0, q_2	B q_1, q_2	F q_2, q_3
* F q_0, q_3	B q_0, q_2	– D q_2
~~G q_1~~	~~D q_2~~	~~G q_1~~
~~H q_2, q_3~~	~~F q_0, q_2~~	~~D q_2~~

Step 3) Transition Diagram

States	a	b
* → A	B	C
B	D	B
C	E	–
D	F	D
* E	B	F
* F	B	D



* Optimization or Minimization of DFA

Optimization is referred detecting those of DFA whose presence or absence does not affect the DFA such as unreachable state, dead state, non-distinguishable states.

For optimization or minimization of DFA-

(1) Unreachable state -

These are those states of DFA which are non-reachable from the initial state of DFA on any possible input sequence

(2) Dead state -

A dead state is that non-fin

Scanned by CamScanner

State of DFA whose transition on every input symbol terminates on itself.

(3) Non-distinguishable State -

These are those states of DFA for which their exists no distinguishing string hence they can not be distinguish from one another.

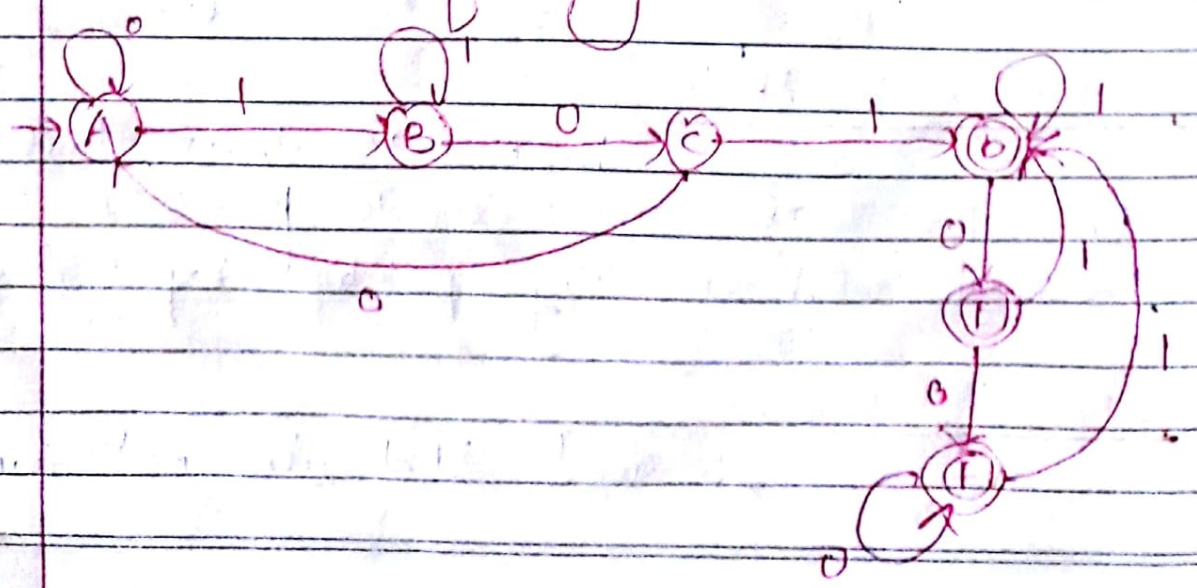
Therefore, optimization refers to -

(1) Detection of unreachable states & eliminiting them from DFA.

(2) Identification of non-distinguishing states & merging them together.

(3) Detecting the dead state & eliminating them from DFA.

Q.1) Convert the following into minimized DFA.



Scanned by CamScanner

⇒ Step 1>
Transition table

States	0	1
→ A	A	B
B	C	B
C	A	D
* D	E	D
* E	F	D
* F	F	D

Step 2> Using partition method

Non-final state final state

[A B C] [D E F] 0 equivalence

[A B] [C] [D E F] 1 equivalence

[A] [B] [C] [D E F] 2 equivalence

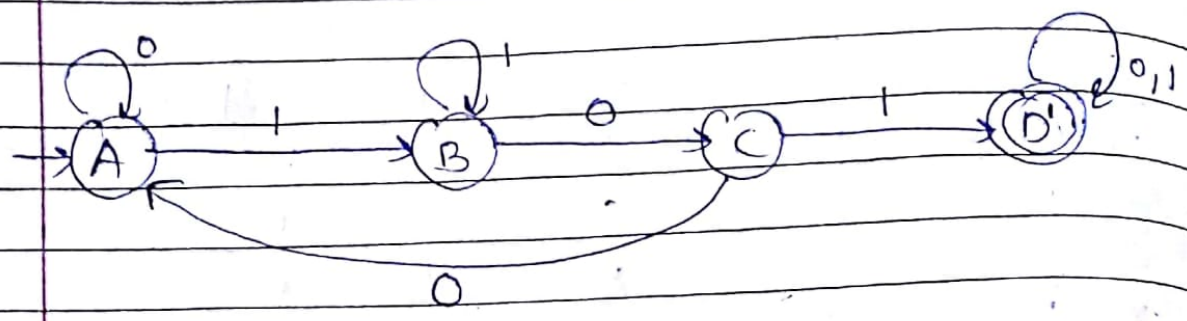
D = E = F = D'

3> Optimized Transition table

States	0	1
→ A	A	B
B	C	B
C	A	D'
* D'	D'	D'

Scanned by CamScanner

Step 4) Optimized DFA



Q. 2) Convert the following NFA into minimize DFA.

States	0	1
→ p	p, q	p
q	r	r
r	s	ϕ
* s	s	s

⇒ (1) Transition table

	State	0		1	
A	p	B	pq	A	p
B	pq	C	pqr	D	pr
C	pqr	E	pqrs	D	pr
D	pr	F	pqs	A	p
E	pqrs	E	pqrs	G	prs
F	pqs	E	pqrs	G	prs
G	prs	F	pqs	H	ps
H	ps	F	pqs	H	ps

Scanned by CamScanner

Step 2

States	0	1
→ A	B	A
B	C	D
C	E	D
D	F	A
* E	E	G
* F	E	G
* G	F	H
* H	F	H

Step 3 Using Partition Method

Non-final final

[A B C D] [E F G H] 0 equivalence

[A.B] [C] [D] [E F G H] 1 equivalence

[A] [B] [C] [D] [E F G H] 2 equivalence

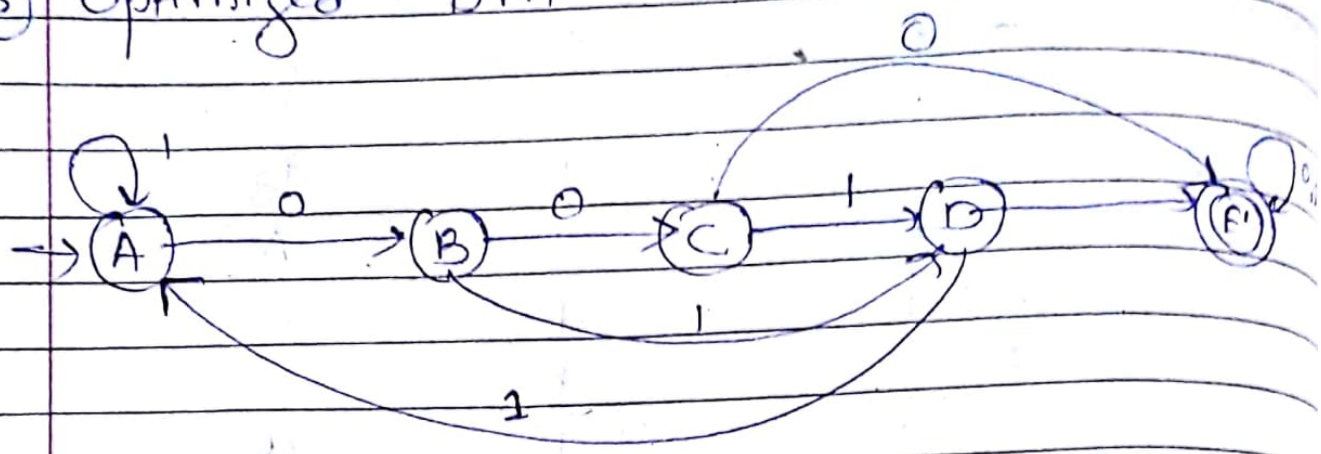
E = F = G = H = E'

4

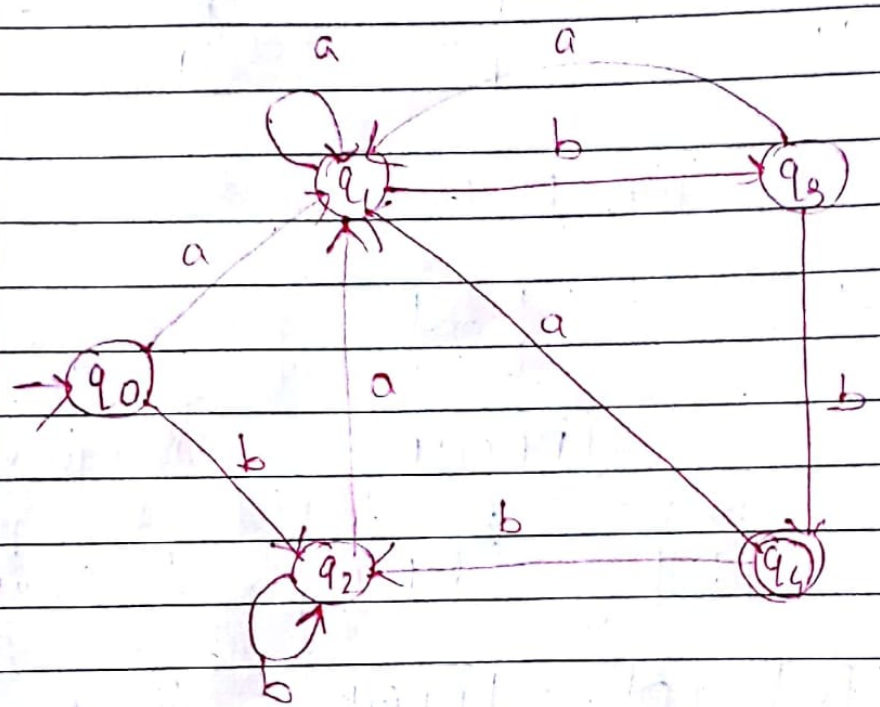
states	0	1
A	B	A
B	C	D
C	E'	D
D	E'	A
E'	E'	E'

Scanned by CamScanner

(5) Optimized DFA



Q. 3)



⇒

States	a	b
→ q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
* q_4	q_1	q_2

Scanned by CamScanner

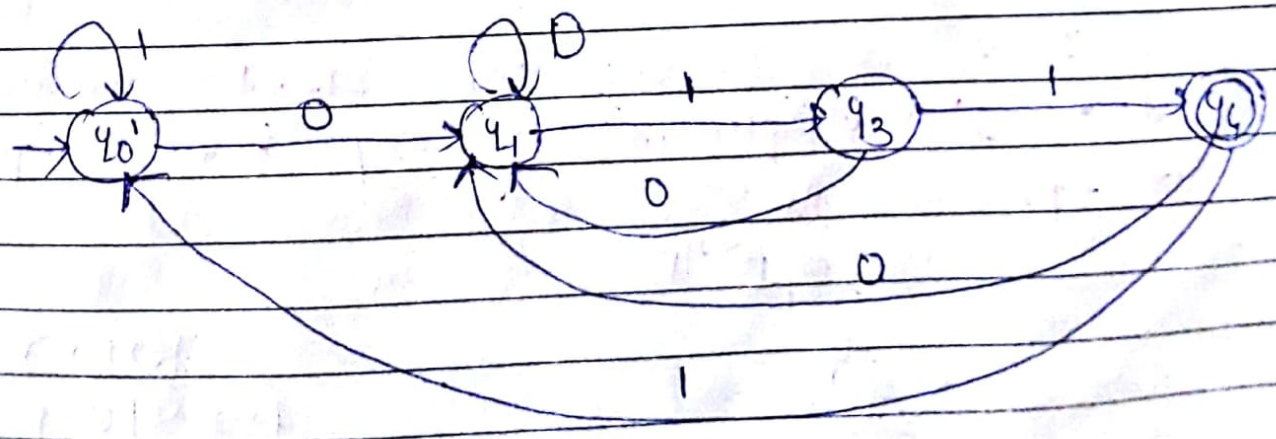
(2) Non-final state | final state

$q_0\ q_1\ q_2\ q_3$			q_4	0 equivalence
q_0, q_1, q_2	q_3		q_4	1 equivalence
q_0, q_2	q_1	q_3	q_4	2 equivalence

check eq. 1

$q_0 = q_2 = q_0'$

States	0	1
→ q_0'	q_1	q_0'
q_1	q_1	q_3
q_3	q_1	q_4
* q_4	q_1	q_0'



Scanned by CamScanner

$\Sigma \rightarrow$ Input $\quad \Delta \rightarrow$ output

* Mealy Machine (o/p is associated with inputs)

In the case of finite automata with output when output is associated with transition then finite automata is called mealy m/c.

Mathematically mealy m/c can be defined as 6 tuples as -

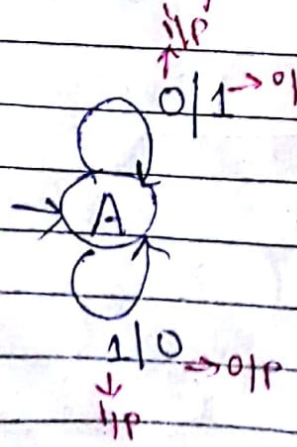
$$M = \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$$

where,

Q = finite set of states
Σ = finite set of i/p symbol
Δ = —"— o/p symbol
δ = Mapping $Q \times \Sigma \Rightarrow Q$
λ = mapping $Q \times \Sigma \Rightarrow \Delta$
q_0 = Initial State

Q. 1) Construct a Mealy m/c that produces the 1's complement of any binary input string.

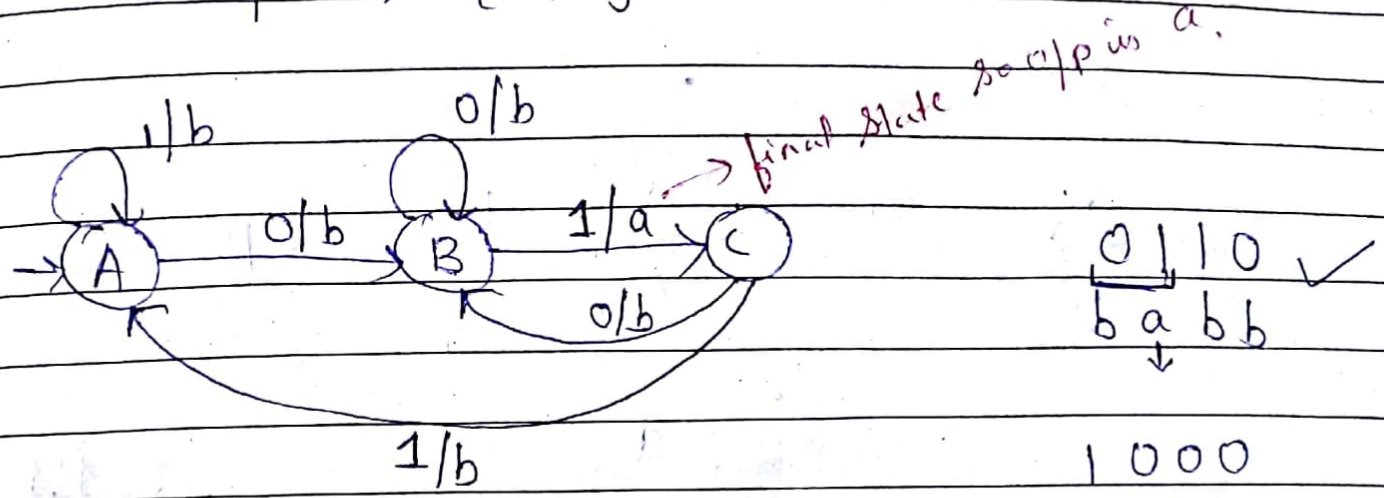
⇒



10100
1's → 01011

Q.2) Construct a Mealy m/c that prints 'a' whenever the sequence "01" is encountered in any input binary string.
↳ end

→ Input ⇒ $\{0, 1\} \rightarrow \Sigma$
output ⇒ $\{a, b\} \rightarrow \Delta$



0 1 1 0 ✓
b a b b

1 0 0 0
b b b b

* Moore Machine (o/p is associated with states)

In the case of finite Automata with output when output is associated with state then finite automata is called moore m/c.

$M = \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$

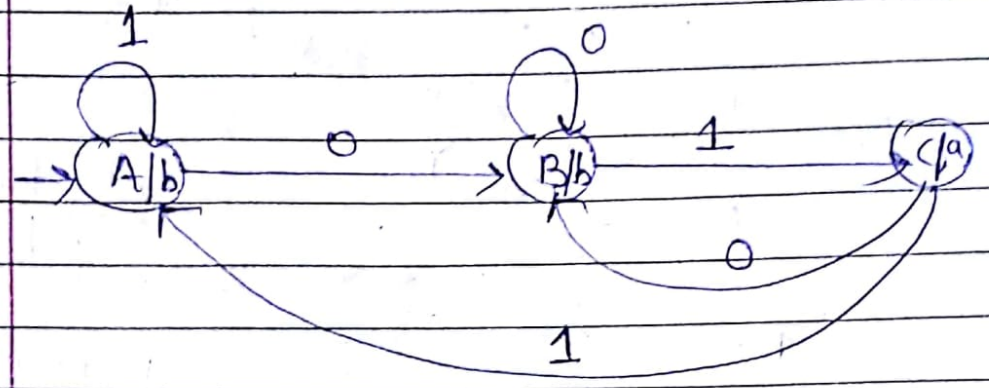
where,

Q = finite set of states
Σ = —"— input symbol
Δ = —"— output symbol
δ = mapping $Q \times \Sigma \rightarrow Q$
λ = —"— $Q \rightarrow \Delta$
q_0 = Initial State

Scanned by CamScanner

Q.1) Construct a Moore m/c that prints 'a' when the sequence '01' is encountered in any input binary string.

⇒ $\Sigma = \{0, 1\}$ $\Delta = \{a, b\}$



0 1 0 1
A B C B C
b b a b a

0 1 1 0
A B C A B
b b a b b

* Conversion of Mealy M/C to Moore M/C

Q.1) Convert the following given Mealy m/c to its equivalent Moore m/c.

State	a	b
→ q_0	$q_3, 0$	$q_1, 1$
q_1	$q_0, 1$	$q_3, 0$
q_2	$q_2, 1$	$q_2, 0$
q_3	$q_1, 0$	$q_0, 1$

Scanned by CamScanner

q_1 q_2

$q_1, 0 (q_{10})$ $q_1, 1 (q_{11})$ $q_2, 0 (q_{20})$ $q_2, 1 (q_{21})$

output 0 output 1 output 0 output 1

State	a	b	Output
→ q_0	$q_3, 0$	$q_{11}, 1$	1
q_{10}	$q_0, 1$	$q_3, 0$	0
q_{11}	$q_0, 1$	$q_3, 0$	1
q_{20}	$q_{21}, 1$	$q_{20}, 0$	0
q_{21}	$q_{21}, 1$	$q_{20}, 0$	1
q_3	$q_{10}, 0$	$q_0, 1$	0

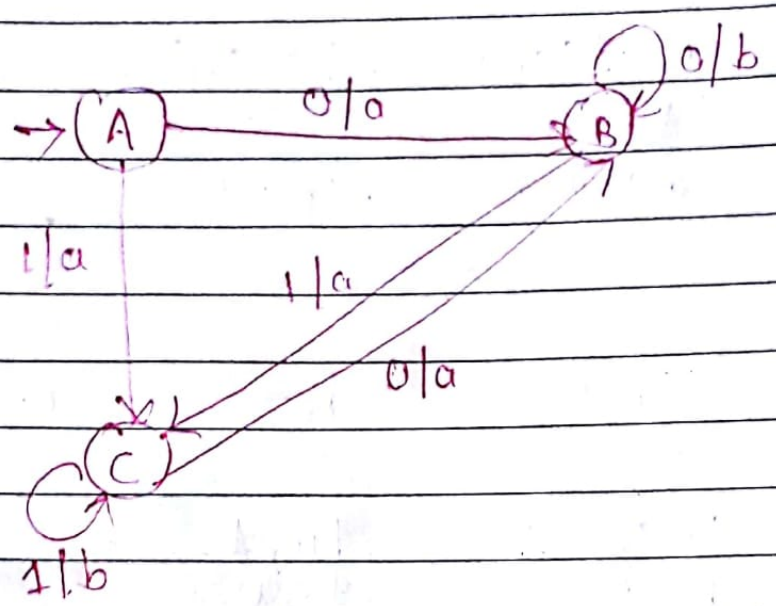
States	a	b	Output
q_0	q_3	q_{11}	1
q_{10}	q_0	q_3	0
q_{11}	q_0	q_3	1
q_{20}	q_2	q_{20}	0
q_{21}	q_{21}	q_{20}	1
q_3	q_{10}	q_0	0

Scanned by CamScanner

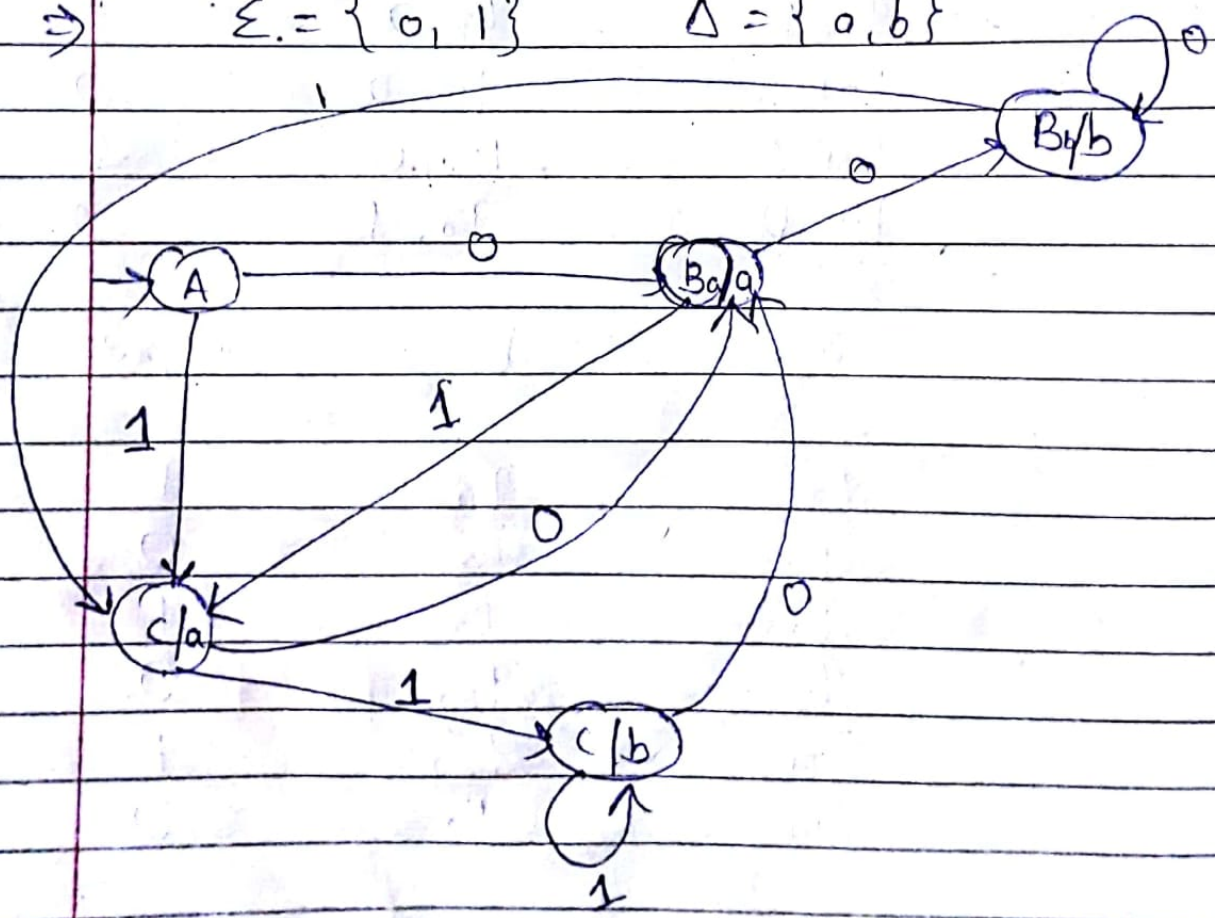
Moore → Mealy ⇒ No. of States were same

Mealy → Moore ⇒ No. of States increased

Q.2) Convert the following Mealy m/c to its equivalent Moore m/c.



⇒ $\Sigma = \{0, 1\}$ $\Delta = \{a, b\}$



Scanned by CamScanner

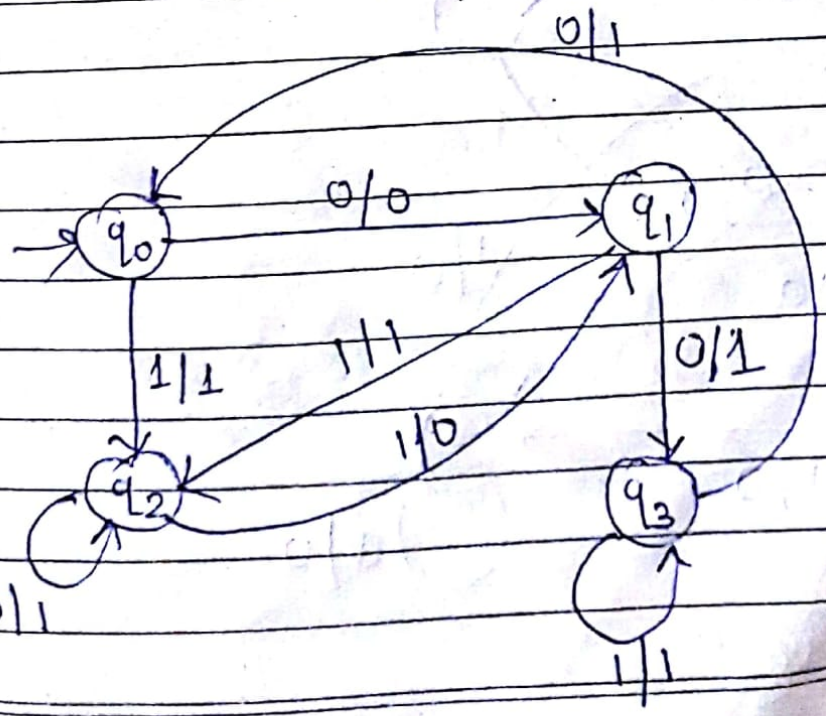
* Conversion of Moore m/c to Mealy m/c

Q.1) Convert the given moore m/c to its equivalent mealy m/c.

States	0	1	output
→ q_0	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

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

State	0	1 → o/p
→ q_0	q_1, 0 output	q_2, 1
q_1	q_3, 1	q_2, 0
q_2	q_2, 1	q_1, 0
q_3	q_0, 1	q_3, 1



Scanned by CamScanner

15-16 (8M)

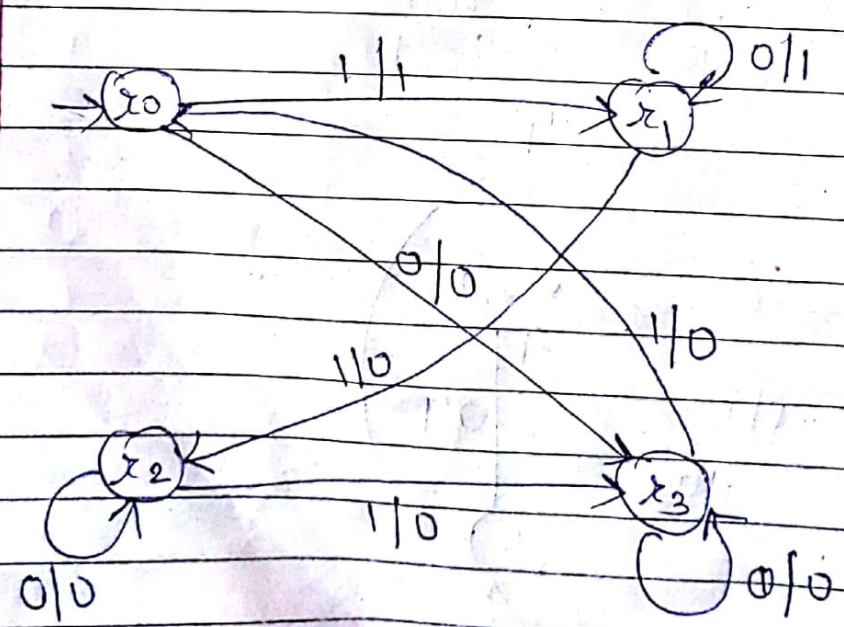
Q. :- Compare Moore m/c with mealy m/c. Construct mealy m/c equivalent to the Moore m/c given below.

Present state	Next state b=0	Next state b=1	Output
r_0	r_3	r_1	0
r_1	r_1	r_2	1
r_2	r_2	r_3	0
r_3	r_3	r_0	0

⇒

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

State	0	1
r_0	$r_3, 0$	$r_1, 1$
r_1	$r_1, 1$	$r_2, 0$
r_2	$r_2, 0$	$r_3, 0$
r_3	$r_3, 0$	$r_0, 0$



Scanned by CamScanner

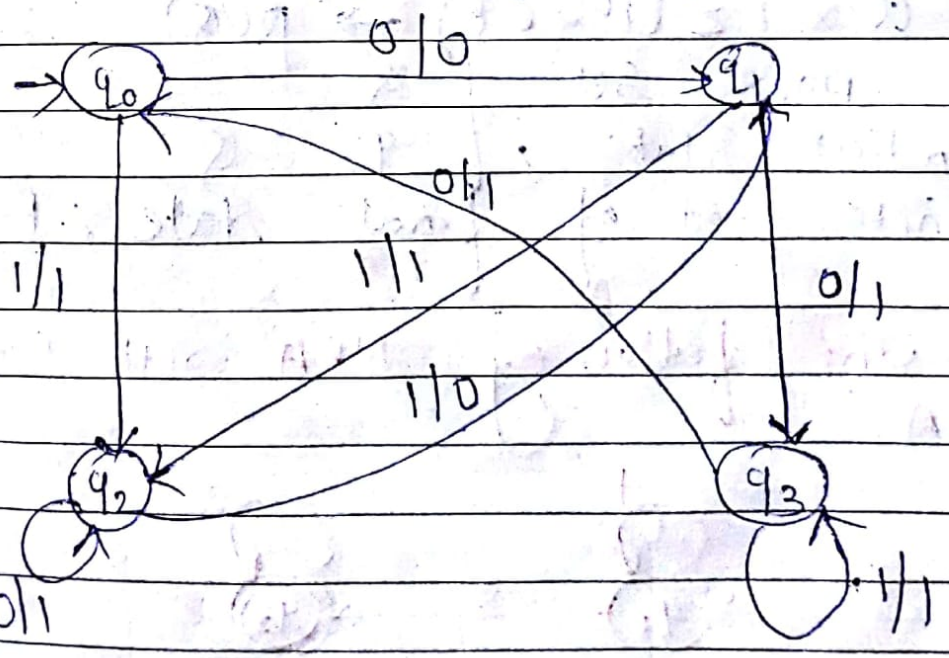
Date :

Q.3) Construct a Mealy m/c which is equivalent to moore m/c.

	a=0	b=1	O/P
→ q_0	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

→

State	0	1
q_0	q_1, 0	q_2, 1
q_1	q_3, 1	q_2, 1
q_2	q_2, 1	q_1, 0
q_3	q_0, 1	q_3, 1



Scanned by CamScanner

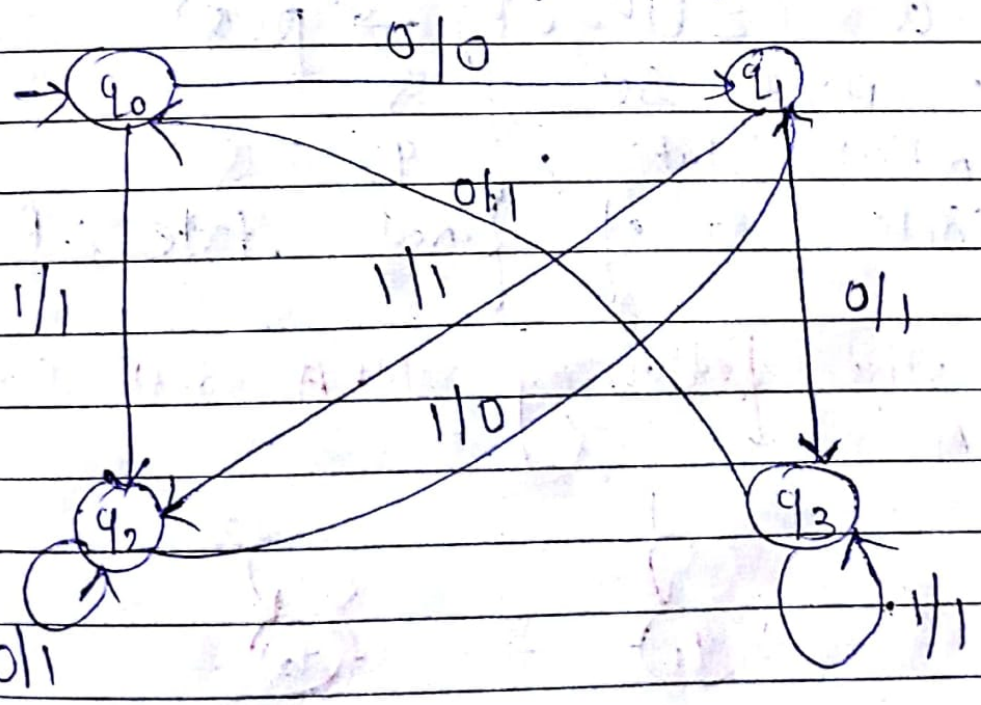
Date:

Q.3) Construct a Mealy m/c which is equivalent to moore m/c.

	a = 0	b = 1	O/P
→ q_0	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

→

State	0	1
q_0	q_1, 0	q_2, 1
q_1	q_3, 1	q_2, 1
q_2	q_2, 1	q_1, 0
q_3	q_0, 1	q_3, 1



Scanned by CamScanner

Non-Deterministic Finite Automata (NFA) with ϵ-transition

If any finite automata containing ϵ (null) move or transaction, then that finite automata is called NFA with ϵ-moves.

Mathematically NFA with ϵ-move can be defined as five tuples as -

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

where,

Q = finite set of states

Σ = " ——— input symbol

δ = transition fun mapping as

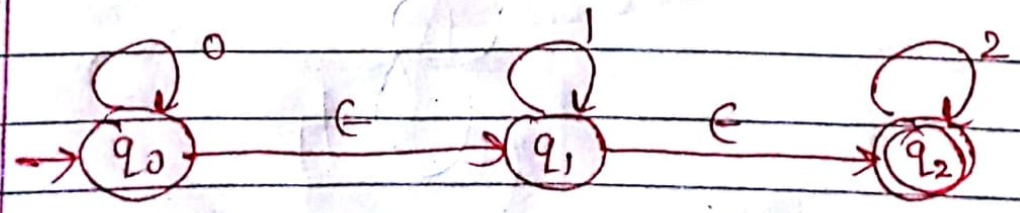
$$Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$$

i.e. power set of Q.

q_0 = Initial state, $q_0 \in Q$

F = Finite set of final state; $F \subseteq Q$.

Q. 1) Convert the following NFA with ϵ-moves to NFA.



Scanned by CamScanner

1) Transition table

Q \ Σ	0	1	2	ε
→ q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
* q_2	ϕ	ϕ	q_2	ϕ

2) Find ε-closure of each state

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

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

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

3) Find transition function δ for each state

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

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

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

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

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

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

Scanned by CamScanner

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

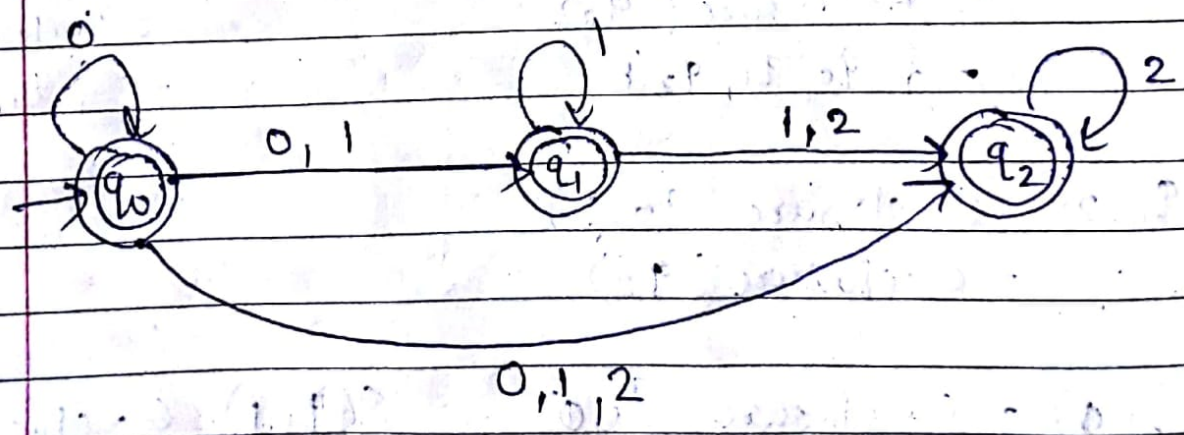
$\delta(q_2, 1) = \epsilon\text{-closure}(q_2, 1)$
$= \phi$

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

4) Hence, Required transition table is -

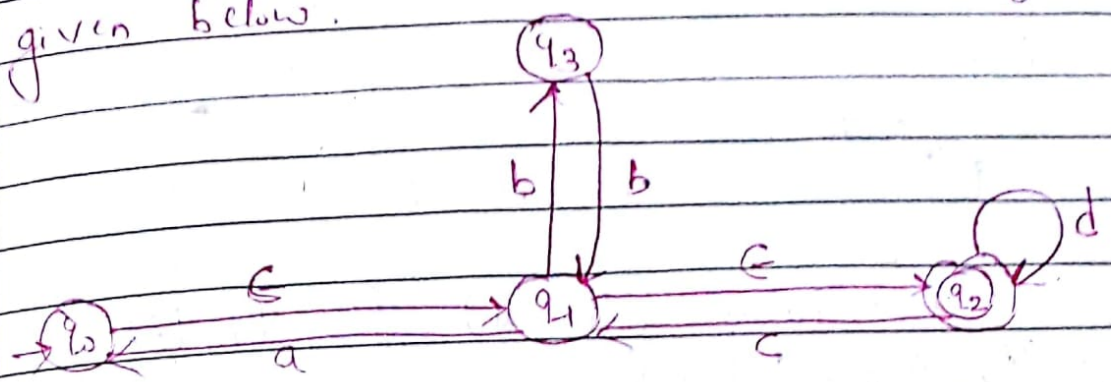
Q \ Σ	0	1	2
→ * q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
* q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
* q_2	ϕ	ϕ	$\{q_2\}$

Final state $\Rightarrow \{q_0, q_1, q_2\}$



Scanned by CamScanner

Q.2: Construct NFA without ε-move from the m/c given below.



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

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

ϵ^* a ϵ^*

$q_0 \to q_0 \to \phi$ $\nearrow q_0$
$q_1 \to q_0 \to q_1$
$q_2 \to \phi$ $\searrow q_2$

$\delta(q_0, b) = \epsilon\text{-closure}(q_0, b) = \{q_3\}$

ϵ^* b ϵ^*

$q_0 \to q_0 \to \phi$
$q_1 \to q_3 \to q_3$
$q_2 \to \phi$

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

ϵ^* c ϵ^*

$q_0 \to q_0 \to \phi$
$q_1 \to \phi$ $\nearrow q_1$
$q_2 \to q_1$ $\searrow q_2$

Scanned by CamScanner

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

$\epsilon^* \quad d \quad \epsilon^*$

$q_0 \to q_0 \to \phi$

$q_1 \to \phi \to q_2$

$q_2 \to q_2$

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

$\epsilon^* \quad a \quad \epsilon^*$

$q_1 \to q_1 \to q_0 \to q_0$

$q_2 \to \phi \quad q_1$

q_2

$\delta(q_1, b) = \epsilon\text{-closure}(q_1, b) = \{q_3\}$

$\epsilon^* \quad b \quad \epsilon^*$

$q_1 \to q_1 \to q_3 \to q_3$

$q_2 \to \phi$

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

$\epsilon^* \quad c \quad \epsilon^*$

$q_1 \to q_1 \to \phi \to q_1$

$q_2 \to q_1 \quad q_2$

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

$\epsilon^* \quad d \quad \epsilon^*$

$q_1 \to q_1 \to \phi$

$q_2 \to q_2 \to q_2$

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

$\epsilon^* \quad a \quad \epsilon^*$

$q_2 \to q_2 \to \phi \to \phi$

Scanned by CamScanner

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

ϵ^* b ϵ^*

$q_2 \to q_2 \to \phi \to \phi$

(4) $\delta(q_3, a) = \epsilon\text{-closure}(q_3, a) = \{\phi\}$

ϵ^* a ϵ^*

$q_3 \to q_3 \to \phi \to \phi$

$\delta(q_3, b) = \epsilon\text{-closure}(q_3, b) = \{q_1, q_2\}$

ϵ^* b ϵ^*

$q_3 = q_3 \to q_1 \to q_1, q_2$

$\delta(q_3, c) = \epsilon\text{-closure}(q_3, c) = \phi$

ϵ^* c ϵ^*

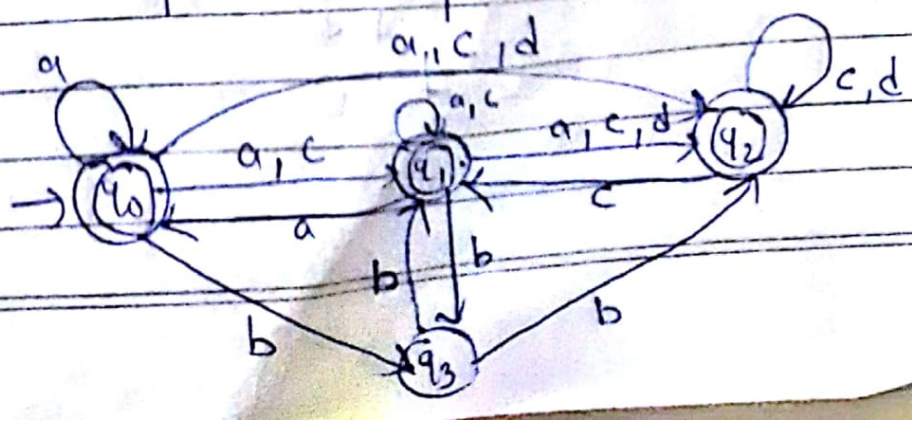
$q_3 \to q_3 \to \phi \to \phi$

$\delta(q_3, d) = \epsilon\text{-closure}(q_3, d) = \phi$

Transistion table -

Q \ Σ	a	b	c	d
→ q_0	$\{q_0, q_1, q_2\}$	$\{q_3\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	$\{q_0, q_1, q_2\}$	$\{q_3\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_3	ϕ	$\{q_1, q_2\}$	ϕ	ϕ

The transition dia →



Scanned by CamScanner

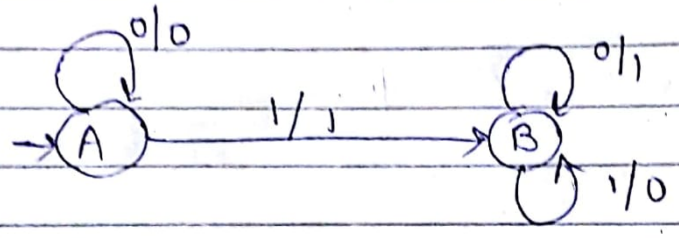
Q) Construct a Mealy m/c that gives 2's complement of any binary i/p (Assume that the last carry bit is neglected).

→

2's complement = 1's complement + 1

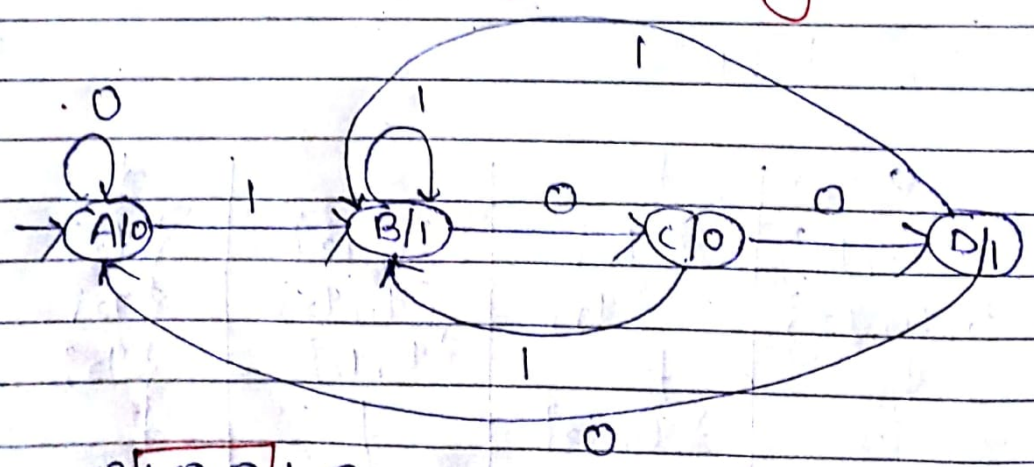
MSB ← LSB

e.g →	10100	11100	1111
1's →	01011	00011	0000
+	1	+ 1	+ 1
2's	01100	00100	0001



Q) Construct the moore m/c to replace each occurance of substring 100 by 101

→



0 1 0 0 1 0

↓ ↓ ↓ ↓ ↓ ↓

0 1 0 1 1 0

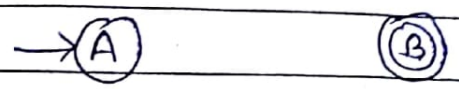
Scanned by CamScanner

Unit - II

* Regular Sets and Regular Expressions (NFA with ∈ moves)

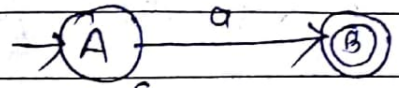
Regular Expression — finite Automata

1) ϕ

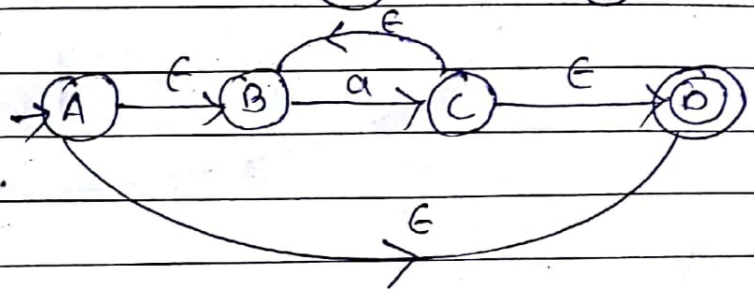


2) ϵ

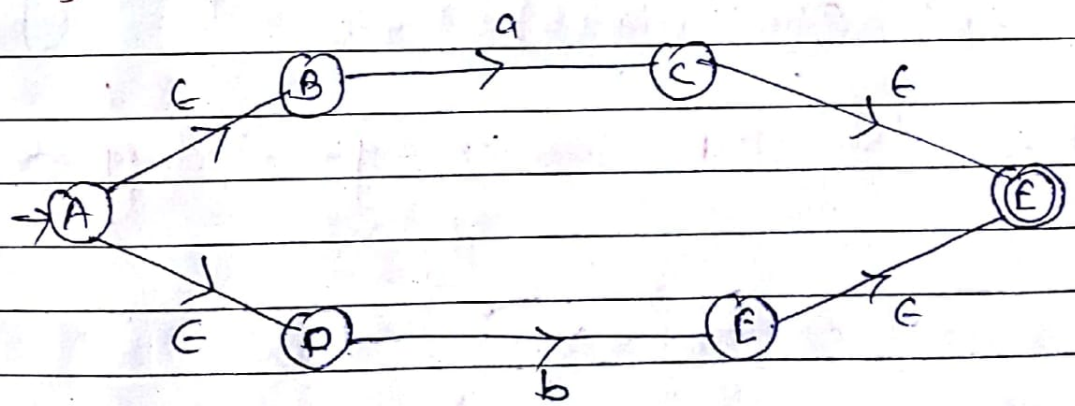
3) a



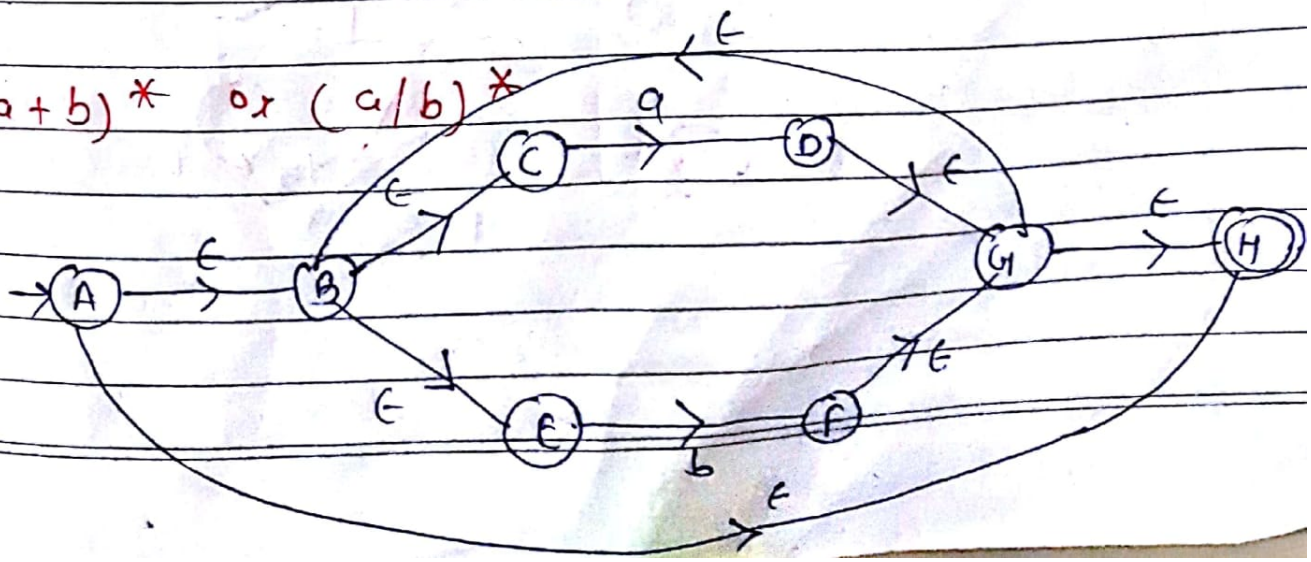
4) a*



5) (a+b) or (a/b)



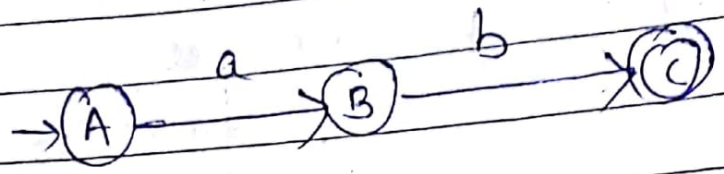
6) (a+b)* or (a/b)*



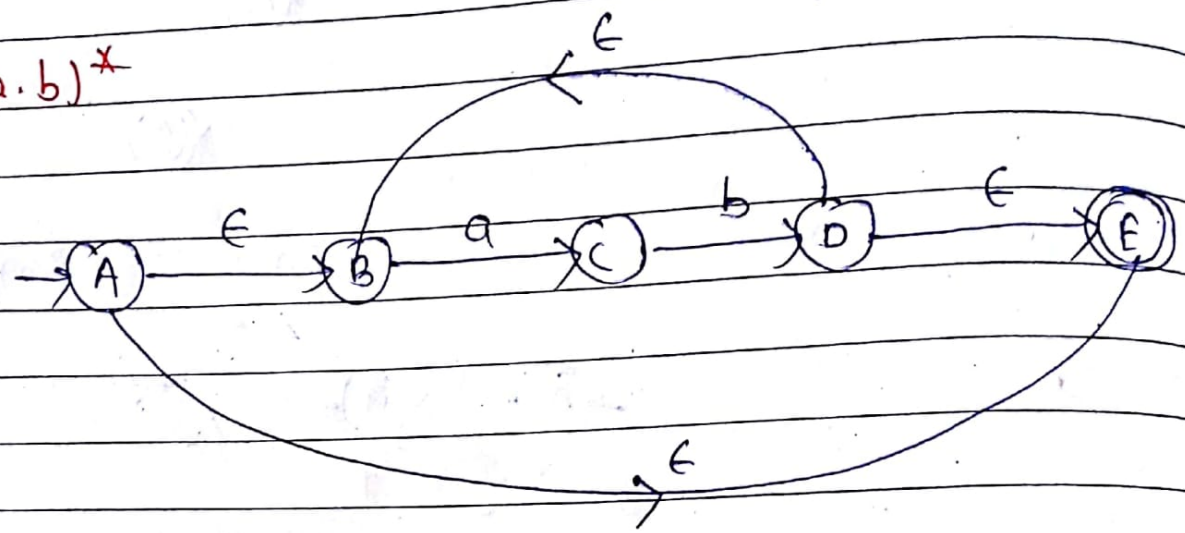
Scanned by CamScanner

Date :

7) a.b



8) $(a.b)^*$



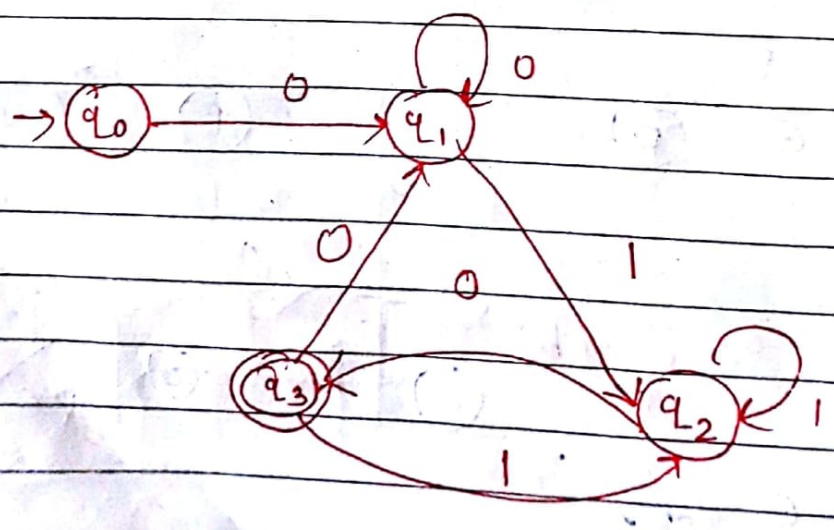
W-18 (7)

Q) 1) Construct a DFA from the following R.E

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

S-18 (6)

Q. 2) Find the R.E for the given diagram



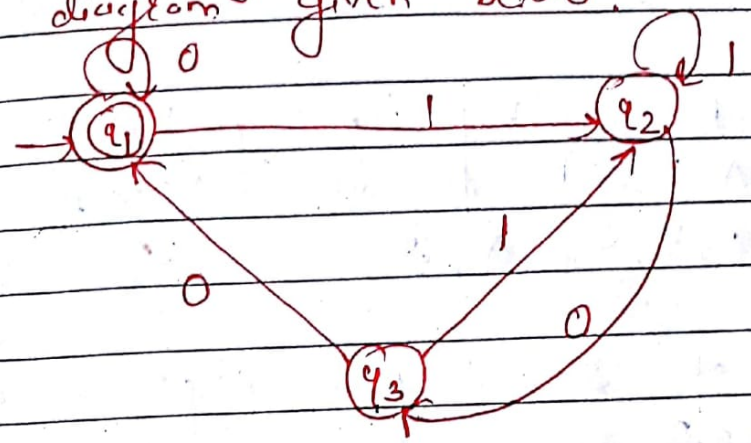
Scanned by CamScanner

W-17 (7)

Q.3) Construct the Finite Automata for the following R.E.
$ab + (a+b)^* aab + bab$

W-16 (7)

Q.4) Construct a regular expression corresponding to the state diagram given below:



S-16 (7)

Q.5) Construct transition diagram for regular expression
i) $R = (ab^*)(a+b)^* (b + ab)$
ii) $R = (0+1)^* (010 + 101)(0+1)^*$

S-16 (7)

Q.6) Consider the following production of the regular grammar G.

$S \rightarrow aA \mid a$
$A \rightarrow aA \mid aB \mid a$
$B \rightarrow bB \mid c$

Find the regular expression corresponding to regular grammar G.

Q.7) Obtain the regular expn from foll. grammar.

S-17 (3)

$S \rightarrow 1A \mid 0C \mid 0$
$A \rightarrow 1B$
$B \rightarrow 1B \mid 1D$
$C \rightarrow 0$
$D \rightarrow 0D \mid 01$

Scanned by CamScanner

* Regular Expression *

Regular expⁿ are used for representing certain sets of strings in an algebric fashion.

1) Any terminal symbol i.e symbol $\in \Sigma$ including Λ and ϕ are regular expressions
e.x. a, b, c, Λ, ϕ

2) The union of two regular expⁿ is also a regular expⁿ
e.x. $R_1, R_2 \quad (R_1 + R_2)$

3) The Concatenation of two regular expⁿ is also a regular expⁿ.
ex. $R_1, R_2 \rightarrow (R_1 \cdot R_2)$

4) The iteration (or closure) of a regular expⁿ is also a regular expⁿ.

$R \rightarrow R^*$, $a^* = \Lambda, a, aa, aaa$.

5) The regular expⁿ over Σ are precisely those obtained recursively by the application of the above rules or several times.

Scanned by CamScanner

Regular Expression Examples

Q) Describe the following sets as Regular expⁿ

1) $\{0, 1, 2\}$ i.e. 0 or 1 or 2

R.E = $0 + 1 + 2$

2) $\{\Lambda, ab\}$

i.e.

R.E = Λ ab

3) $\{abb, a, b, bba\}$

R.E = $abb + a + b + bba$

4) $\{\Lambda, 0, 00, 000, \ldots\}$ i.e. closure of 0.

R.E = 0^*

5) $\{1, 11, 111, 1111, \ldots\}$

R.E = 1^+

Scanned by CamScanner

* Identities of Regular Expression

1) $\phi + R = R$

2) $\phi R + R\phi = \phi$

3) $\epsilon R = R\epsilon = R$

4) $\epsilon^* = \epsilon$ and $\phi^* = \epsilon$

5) $R^* R^* = R^*$

6) $RR^* = R^* R$

7) $R + R = R$

8) $(R^*)^* = R^*$

9) $\epsilon + RR^* = \epsilon + R^* R = R^*$

10) $(PQ)^* P = P(QP)^*$

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

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

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

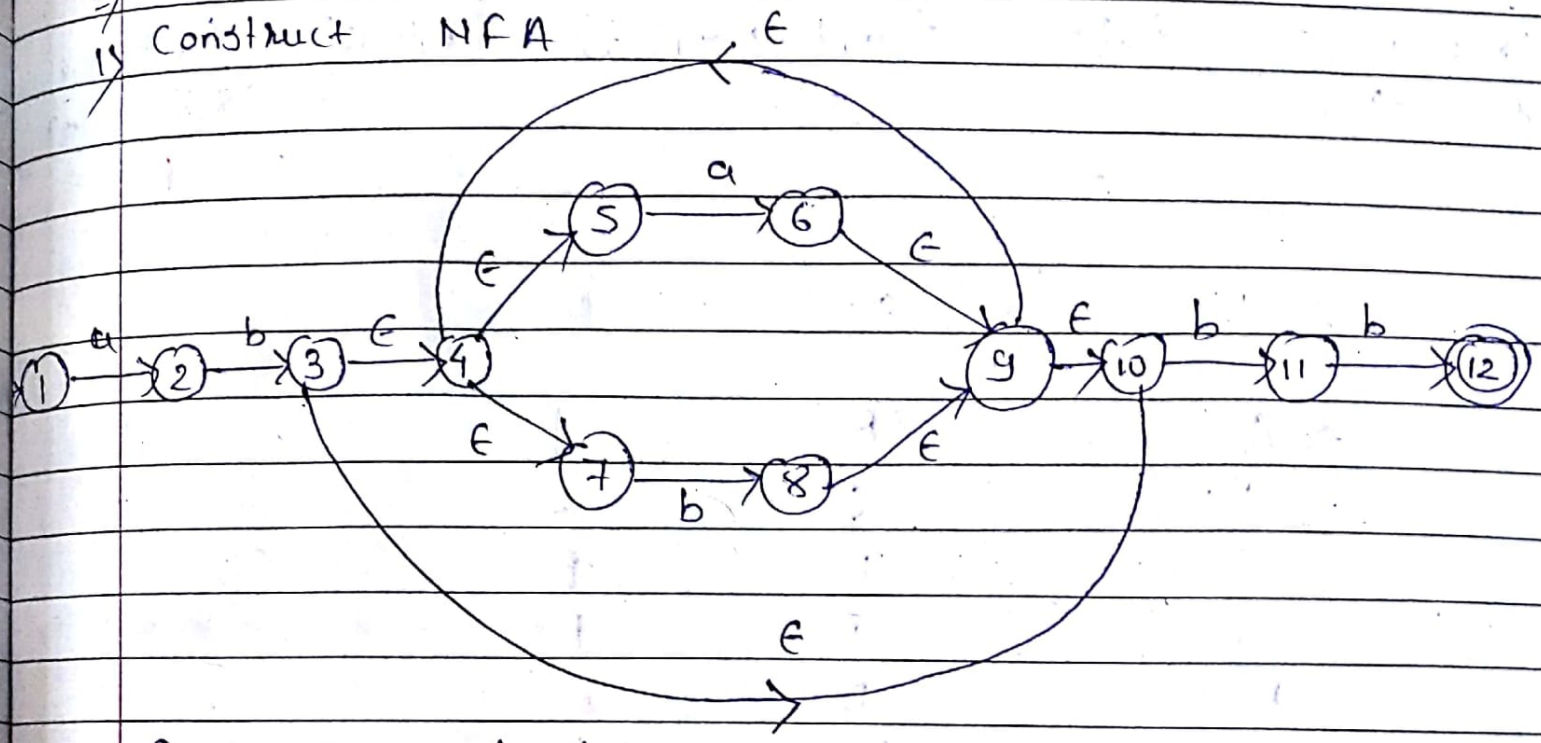
Scanned by CamScanner

Q) Construct a finite automata for the following regular expⁿ.

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

⇒

1) Construct NFA



2) Construction of DFA

ϵ-closure of 1 = {1}

States	δ(S,a)	δ'(S,a)	δ(S,b)	δ'(S,b)
{1} A	{2}	{2} B	ϕ	ϕ
{2} B	ϕ	ϕ	{3}	{3,4,5,7,10} C
{3,4,5,7,10} C	{6}	{6,9,10,4,5,7} D	{8,11}	{8,9,10,4,5,7,11} E
{6,9,10,4,5,7} D	{6}	{6,9,10,4,5,7} D	{11,8}	{8,9,10,4,5,7,11} E

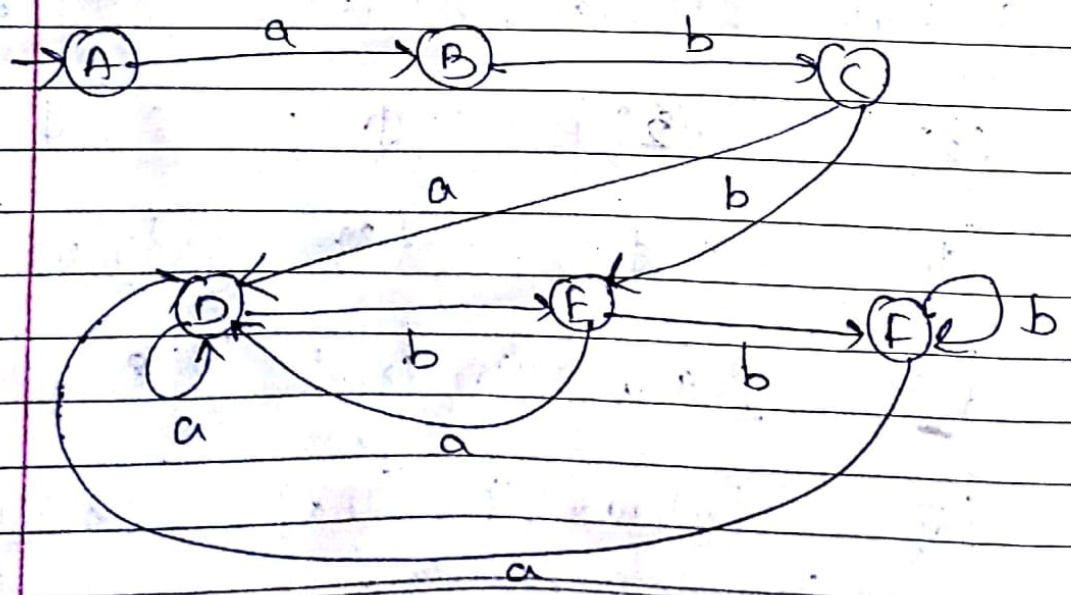
Scanned by CamScanner

Page No.
Date :

States	$\delta(S,a)$	$\delta'(S,a)$	$\delta(S,b)$	$\delta'(S,b)$
{8,9,10,4,5,7,11} F	{6}	{6,9,10,4,5,7} D	{11,8,12}	{8,9,10,4,5,7,11,12} F
{8,9,10,4,5,7,11,12} F	{6}	{6,9,10,4,5,7} D	{8,11,12}	{8,9,10,4,5,7,11,12} F

States	a	b
→ A	B	ϕ
B	ϕ	C
C	D	E
D	D	F
E	D	F
* F	D	F

DFA



Scanned by CamScanner

Page No.
Date :

2) $(0+1)^* (00+11) (0+1)^*$

→ Let, $R = (0+1)^* (00+11) (0+1)^*$

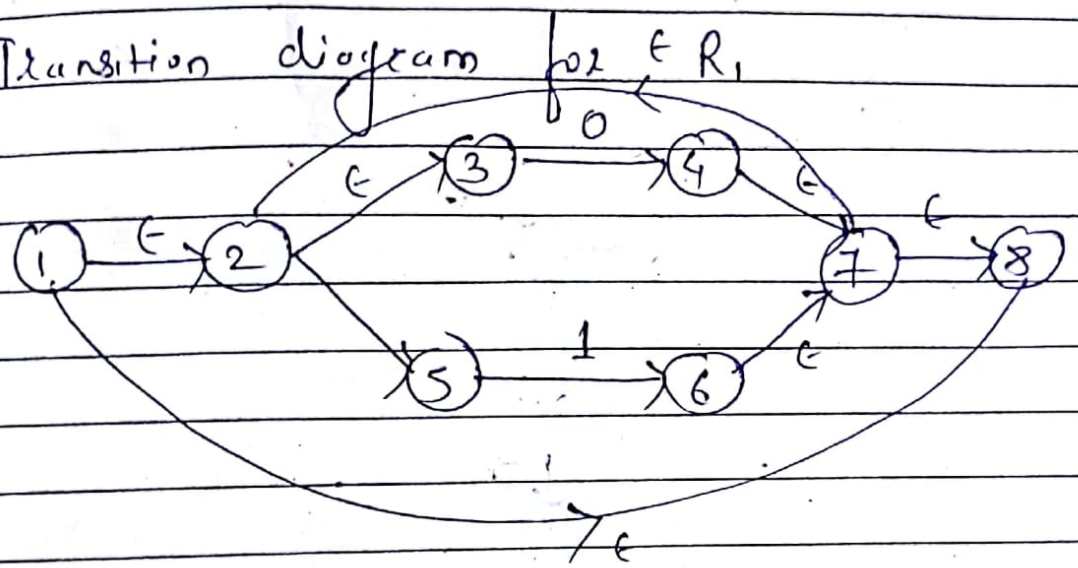
$R = R_1 R_2 R_3$ where

$R_1 = (0+1)^*$

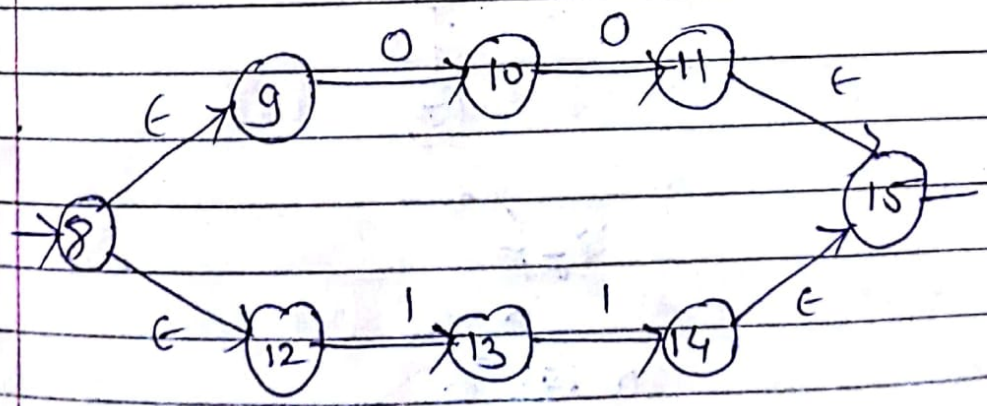
$R_2 = (00+11)$

$R_3 = (0+1)^* = R_1$

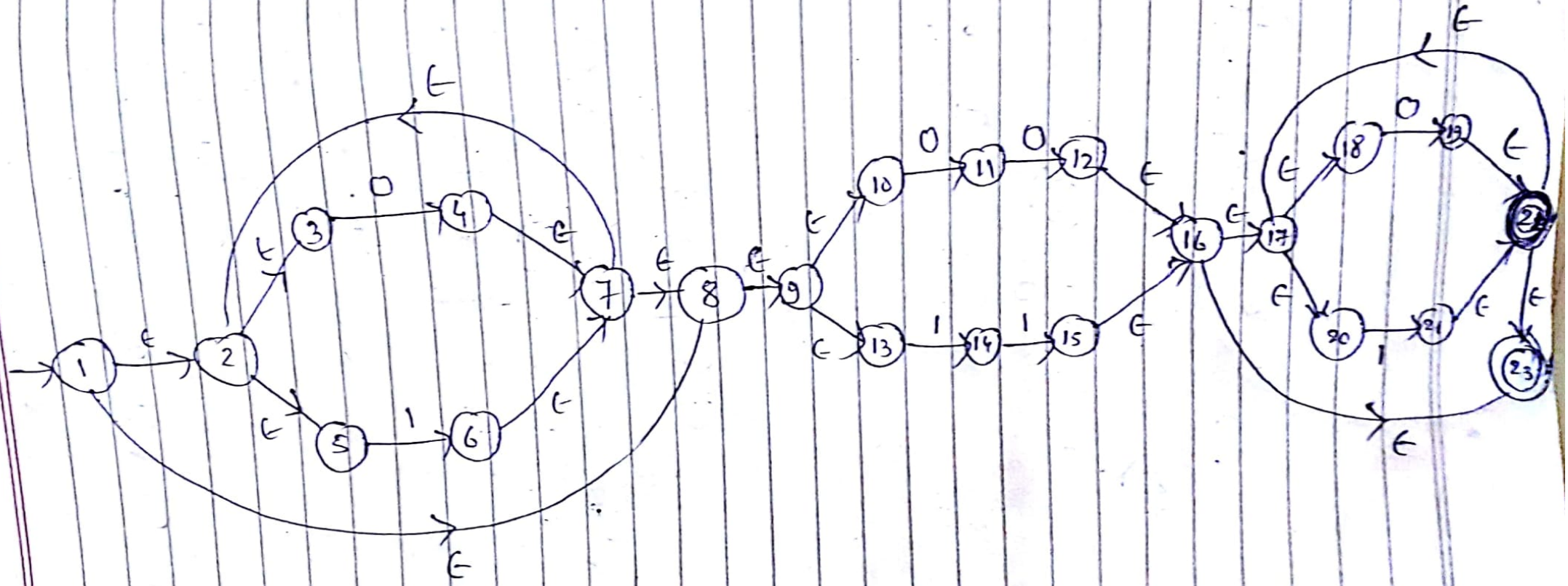
Transition diagram for ϵR_1



$R_2 = (\underset{a}{00} + \underset{b}{11})$



Scanned by CamScanner



Scanned by CamScanner

Page No.
Date :

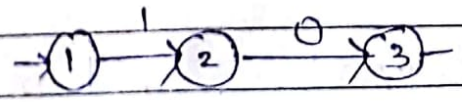
Q.3) Construct a Transition Diagram from the following Regular expression

$10 + (0+11)0^*1$

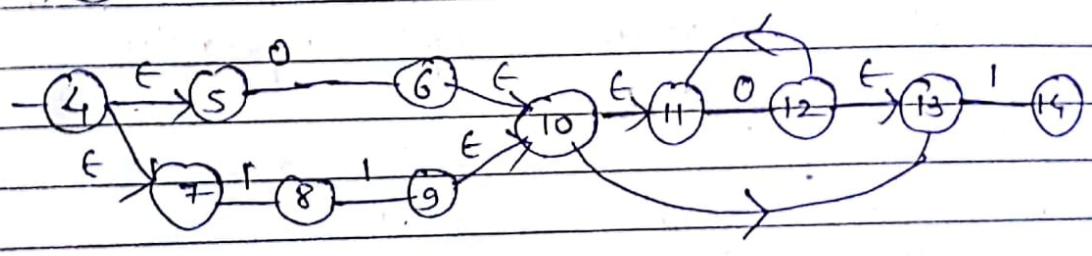
$\Rightarrow$ $\underbrace{10}_{a} + \underbrace{(0+11)0^*1}_{b}$

$R_1 = 10$ $\qquad$ $R_2 = (0+11)0^*1$

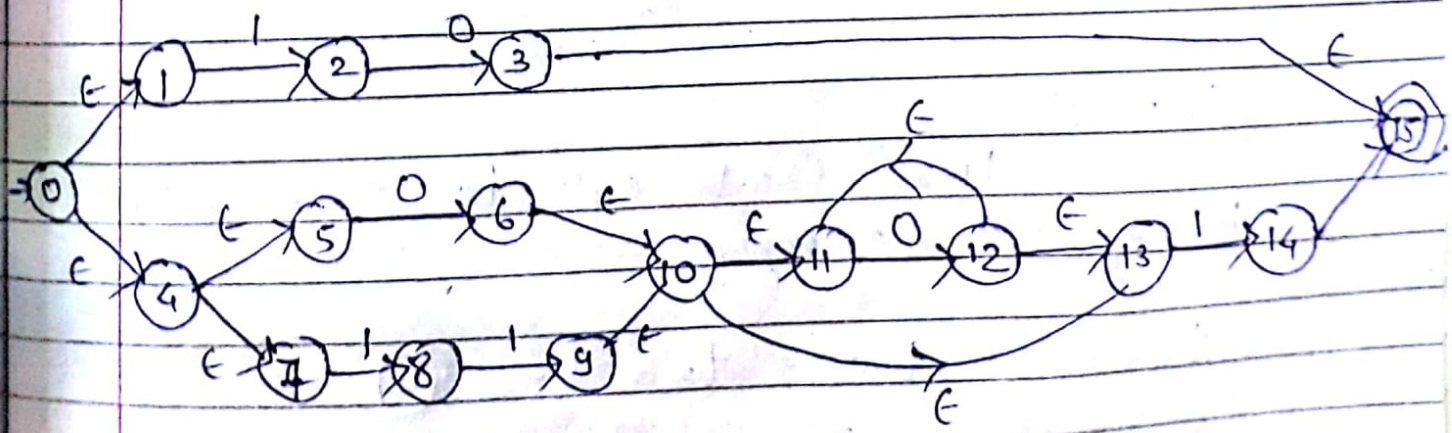
$R_1 \Rightarrow$



$R_2 \Rightarrow$



$R = R_1 + R_2 \Rightarrow$



Scanned by CamScanner

Page No.
Date :

* Regular Expression and FA Inter Conversion

In this conversion we can convert any DFA or NFA into regular expn.

* Arden's Theorem -

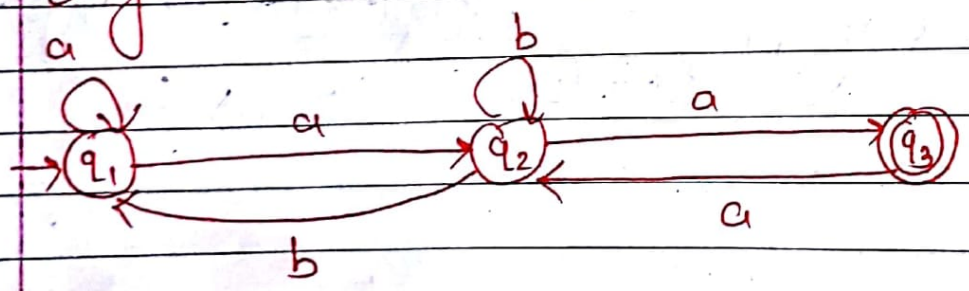
Let,

$$R = Q + RP$$

then,

$$R = QP^*$$

Q.1) Consider the following transition system. Find regular expn equivalent to the diagram below.



⇒

Step-1)

First create eqn for each state having an incoming arrows as.

$q_1 = \epsilon + q_1 a + q_2 b$ —— (a)

$q_2 = q_1 a + q_2 b + q_3 a$ —— (b)

$q_3 = q_2 a$ —— (c)

Scanned by CamScanner

Note → Regular expn will be find in final state only

Page No.
Date :

Step - 11)

Now solve these eqn using substitution method & by using Arden's theorem & find the value of final state, that will be required regular expression.

By substituting (c) in (b)

$q_2 = q_1 a + q_2 b + q_2 aa$

$q_2 = q_1 a + q_2 (b + aa)$

$\therefore q_2 = q_1 a (b + aa)^*$ --- by using Arden's theorem — (d)

By substituting (d) in (a)

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

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

$q_1 = \epsilon (a + a(b + aa)^* b)^*$ $\quad \because \epsilon R = R$

$q_1 = (a + a(b + aa)^* b)^*$ ——— (e)

Substitute the value of (d) in (c)

$q_3 = q_1 a (b + aa)^* a$ ——— (F)

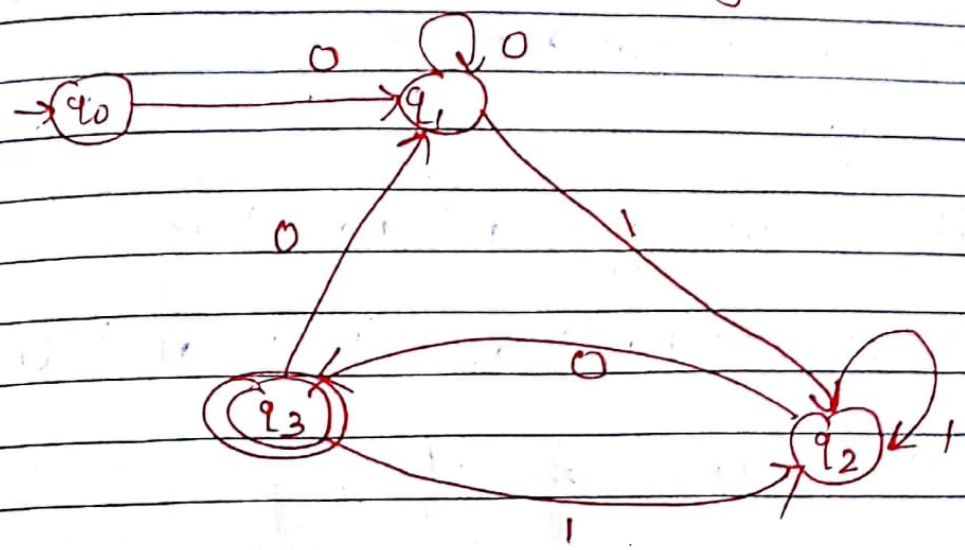
Substitute (e) in (F)

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

Scanned by CamScanner

Date:

Q.3) Find the R.E for the given diagram



$\Rightarrow$ $q_0 = \epsilon$ —— (1)

$q_1 = q_0 0 + q_1 0 + q_3 0$ —— (2)

$q_2 = q_1 1 + q_2 1 + q_3 1$ —— (3)

$q_3 = q_2 0$ —— (4)

From eqn (3), put the value of eqn (4) to (3)

$q_2 = q_1 1 + q_2 1 + q_2 01$

$q_2 = q_1 1 + q_2 (1 + 01)$ — by using Arden's thm

$q_2 = q_1 1 (1 + 01)^*$ —— (5)

From eqn (4), put value of eqn (5) to (4)

$q_3 = q_1 1 (1 + 01)^* 0$ —— (6) put this value to eqn (2)

$q_1 = q_0 0 + q_1 0 + q_1 1 (1 + 01)^* 00$

$q_1 = q_0 0 + q_1 (0 + 1(1 + 01)^* 00)$ — by using A.T

$q_1 = q_0 0 (0 + 1(1 + 01)^* 00)^*$ $\therefore$ put the value of eqn (1) to (7)

$\hookrightarrow$ (7)

Scanned by CamScanner

Page No.
Date :

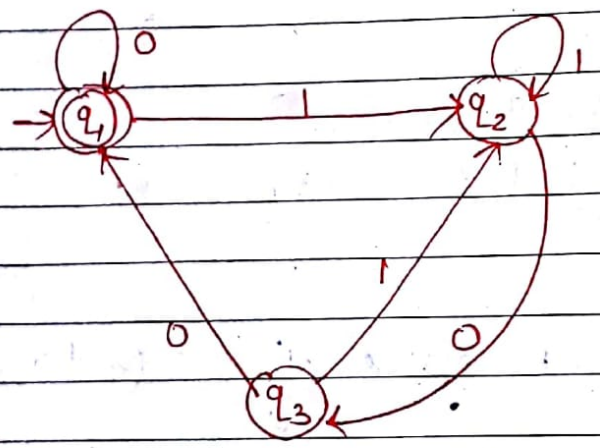
$q_1 = \epsilon 0 (0 + 1(1+01)^* 00)^*$ $\quad \because \epsilon R = R$

$q_1 = 0(0+1(1+01)^* 00)^*$ — (8)

put the value of eqn no. (8) in (6)

$$\boxed{q_3 = 0(0+1(1+01)^* 00)^* 1(1+01)^* 0}$$

Q.4) Find R.E from the following



⇒ Step 1)

$q_1 = \epsilon + q_1 0 + q_3 0$ — (1)

$q_2 = q_1 1 + q_2 1 + q_3 1$ — (2)

$q_3 = q_2 0$ — (3)

Step (2)

$q_2 = q_1 1 + q_2 1 + q_2 01$

$q_2 = q_1 1 + q_2(1+01)$

by using Arden's thm

$$\boxed{q_2 = q_1 1(1+01)^*} \quad \text{— (4)}$$

Scanned by CamScanner

Date :

put the value of eqn (4) to eqn (3)

$q_3 = q_1 1(1+01)^* 0$ —— (5)

put eqn (5) to eqn (1)

$q_1 = \epsilon + q_1 0 + q_1 1(1+01)^* 00$

$q_1 = \epsilon + q_1(0 + 1(1+01)^* 00)$ ... by Arden's thm

$q_1 = \epsilon(0 + 1(1+01)^* 00)^*$ $\because \epsilon R = R$

$$\boxed{q_1 = (0 + 1(1+01)^* 00)^*}$$

* PUMPING LEMMA

The pumping lemma, which is a powerful tool, for proving certain languages, i.e. non regular. It is also useful in the development of algorithms to answer certain questions concerning finite automata, such as whether the language accepted by a given FA is finite or infinite.

Given a language, with the help of pumping Lemma, we can determine whether it is a regular language or non-regular language.

Scanned by CamScanner

Page No.
Date :

* Pumping Lemma is used to prove that a Language is NOT REGULAR.

* It cannot be used to prove that a language is Regular.

If A is a Regular lang, then A has a pumping length 'P' such that any string 's' where $|s| \geq P$ may be divided into 3 parts $s = xyz$, such that the following conditions must be true:

1) $xy^iz \in A$ for every $i \geq 0$
2) $|y| > 0$
3) $|xy| \leq P$

To prove that a language is not Regular using PUMPING LEMMA, follow the below steps:

→ Assume that A is Regular
→ It has to have a Pumping Length (say P)
→ All strings longer than P can be pumped $|s| \geq P$
→ Now find a string 's' in A such that $|s| \geq P$
→ Divide s into xyz
→ Show that $xy^iz \notin A$ for some i
→ Then consider all ways that s can be divided into xyz.

Scanned by CamScanner

Page No.
Date :

→ Show that ~~pro~~ none of these can satisfy all the 3 pumping conditions at the same time
→ S cannot be pumped == CONTRADICTION

Q.1) Using Pumping Lemma prove that the language $A = \{a^n b^n \mid n \geq 0\}$ is not Regular.

⇒ Let, Assume that A is regular

∴ Pumping length = P

$S = a^P b^P$ ⇒ S = aaaaaaabbbbbbb

$\underbrace{\quad}_{x}\underbrace{\quad}_{y}\underbrace{\quad}_{z}$ ∴ P = 7

Case - 1) The y is in the 'a' part

$\underbrace{aa}_{x}\underbrace{aaa}_{y}\underbrace{aa\,bbbbbbb}_{z}$

Case - 2) The y is in the 'b' part

$\underbrace{aaaaaaa\,bb}_{x}\underbrace{bbbb}_{y}\underbrace{b}_{z}$

Case - 3) The y is in the 'a' as well as 'b' part

$\underbrace{aaaaaa}_{x}\underbrace{a\,bbb}_{y}\underbrace{bbbb}_{z}$

Scanned by CamScanner

put $i = 2$, for $a^n b^n$ is same no. of a & b

$xy^iz \Rightarrow xy^2z$

For case 1> aa aaaa aaaa a bbbb bbbb

11 'a's 7 'b'

no of a ≠ no of b

For case 2>

aaaaaaa bb bbbb bbbb b

7 'a's 11 'b

No of a ≠ no. of b

For case 3>

aaaaa aabbaabb bbbb

9 'a' 8 'b'

No. of a ≠ No. of b

∴ Therefore, given language 'L' is NON-REGULAR

Q. 2> $L = \{a^{i^2} \mid i \geq 1\}$ is regular or not.

> Suppose, given language is regular

$L^N = \{a, aaaa, aaaaaaaaa, \ldots\}$

Let, consider,

$S = aaaa$

Scanned by CamScanner

Date :

A/c to pumping lemma we can split this string as

$S = xy^iz$ where, $|xy| \leq P$, $|y| \geq 1$ &
$i = 1, 2, 3 \ldots$

Case-1) $S = aaaa$

$S = \underset{x}{a}\underset{y}{a}\underset{z}{aa}$ ⇒ put $i = 1$

$xy^iz = xy^1z$

$= aaaa.$ → Accepted ~~not accepted~~

Case-2) $S = aaaa$ put $i = 2$

$S = \underset{x}{a}\underset{y}{aa}\underset{z}{a}$ ⇒ $xy^iz = xy^2z$

$= aaaaaa$ → not accepted

Case-3) $S = aaaa$

$S = \underset{x}{a}\underset{y}{aa}\underset{z}{a}$ ⇒ put $i = 3$

$xy^iz \Rightarrow xy^3z$

$= aaaaaa$ → not accepted

If ~~both the~~ all the case is accepted then the grammar is regular but if one case not accepted then the grammar is not regular.

Hence, given language 'L' is Non-regular.

Scanned by CamScanner

Chomsky Hierarchy of Languages:

1) The class of phrase structure grammars is very large but we can put or impose certain constraints on the production rules.

2) This leads to different classes of phrase structure grammar depending on different restrictions.

3) Chomsky has suggested four different classes of phrase structure grammars as follows.
   i) Type 0 (unrestricted grammar)
   ii) Type 1 (context - sensitive grammar)
   iii) Type 2 (context - free grammar)
   iv) Type 3 (regular grammar)

i) Type 0 (unrestricted grammar)

- There are no restrictions on the productions of grammar of this type.
- This type of grammar permits productions of the form

$$\alpha \rightarrow \beta \text{ with } \alpha \neq \epsilon$$

Where, 'α' & 'β' are sentential forms i.e. any combinat^n of any number of terminals & non-terminals i.e. $\alpha, \beta \subset (V \cup T)^*$ but $\alpha \neq \epsilon$. Such a grammar is called as unrestricted grammar.

Scanned by CamScanner

Page No.
Date :

- e.g. let $V = \{A, B, C\}$
  $T = \{a, b, c\}$
  $G = \{V, T, P, A\}$

where, P consists of

$$A \rightarrow AB$$
$$AB \rightarrow BC$$
$$B \rightarrow a$$
$$ac \rightarrow abc$$

ii) Type 1 (context - Sensitive Grammar)

- The restrictions on this type of grammar are:

i) for each production of the form $\alpha \rightarrow \beta$, length of 'β' is atleast as much as the 'α' except for $S \rightarrow \epsilon$.

ii) Also, the start symbol 'S' doesn't appear on the right-hand side of any production.

- The term Context Sensitive is because of the productions of the form,

$$\alpha_1 A \alpha_2 \longrightarrow \alpha_1 \beta \alpha_2 \quad (\beta \neq \epsilon)$$

- Where, the replacement of a non-terminal 'A' is allowed by 'β' only in the context α_1 preceeds 'A' & 'α_2' succeeds 'A'.

Scanned by CamScanner

Page No.
Date :

- e.g Let, the grammar G given as,

$$G = \{\{A, B, C\}, \{a, b\}, P, A\}$$

where, 'P' consists of the productions,

$$A \rightarrow AB$$
$$AB \rightarrow AC$$
$$AC \rightarrow ab$$

iii) Type 2 (Context - Free Grammar)

- In this class of grammar the only allowed type of productions is, $A \rightarrow \alpha$
- Where, 'A' is a non-terminal & 'α' is a sentential form i.e. $\alpha \subset (V \cup T)^*$ $(\alpha = \epsilon)$
- The left hand side of the production should contain only one non-terminal, start symbol can appear on right hand side ~~only~~ also.
- e.g. Let the grammar G as,

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

- Where 'P' consists of the following productions

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

Scanned by CamScanner

Date :

iv) Type 3 (Regular Grammar)

- In this type of productions there are following restrictions on the type of productions:

1) Left hand side of each production should contain only one nonterminal.

2) Right hand side can contain at the most one non-terminal symbol which is allowed to appear as the right most symbol or leftmost symbol.

1) Left-Linear Grammar:

- Regular grammar can contain at the most one non-terminal on the right hand side of its production.

- If this variable appears as the leftmost symbol on the right hand side, the regular grammar is said to be left-linear.

- The allowed types of productions in left linear grammar are-

$A \rightarrow Bw$, $A \rightarrow \epsilon$ or $A \rightarrow w$

where, 'A' & 'B' are variables & 'w' is a string of terminals.

Scanned by CamScanner

'A' 'B' → non-terminal
'a' 'b' → terminal

Page No.
Date :

- e.g. Consider the grammar

$G = (\{S, B, C\}, \{a, b\}, P, S)$

where, P ⇒

$S \rightarrow Ca \mid Bb$
$C \rightarrow Bb$
$B \rightarrow Ba \mid b$

2) Right-linear Grammar :

- A regular grammar consisting of productions with at the most one non-terminal on the right hand side & as the right most symbol on the right hand side of the production then the grammar is said to be right-linear grammar.

- The allowed types of productions are,
  $A \rightarrow wB$, $A \rightarrow w$ or $A \rightarrow \epsilon$
  where, 'A' & 'B' are non-terminals
  'w' is terminals.

e.g. $G = (\{S, A\}, \{0, 1, \epsilon\}, P, S)$

where, P ⇒ $S \rightarrow 0A$
$A \rightarrow 1 \mid 0A \mid \epsilon$

Scanned by CamScanner

* Equivalence of Right-linear & Left-linear grammars.

* Conversion of Right-linear grammar to Equivalent Left-linear grammar:

Q.1) Convert the following right-linear grammar to equivalent left-linear grammar.

$S \rightarrow bB$

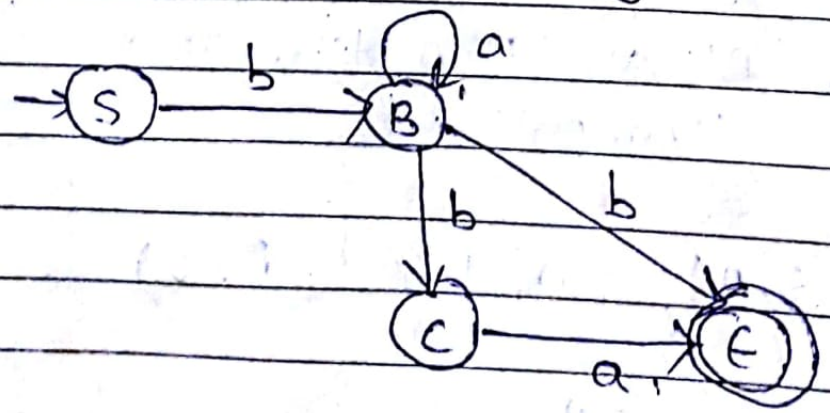
$B \rightarrow bC$

$B \rightarrow aB$

$C \rightarrow a$

$B \rightarrow b$

⇒ Step 1) Initial state will be labelled by start symbol 'S' & the final state will be labelled by 'ε'.

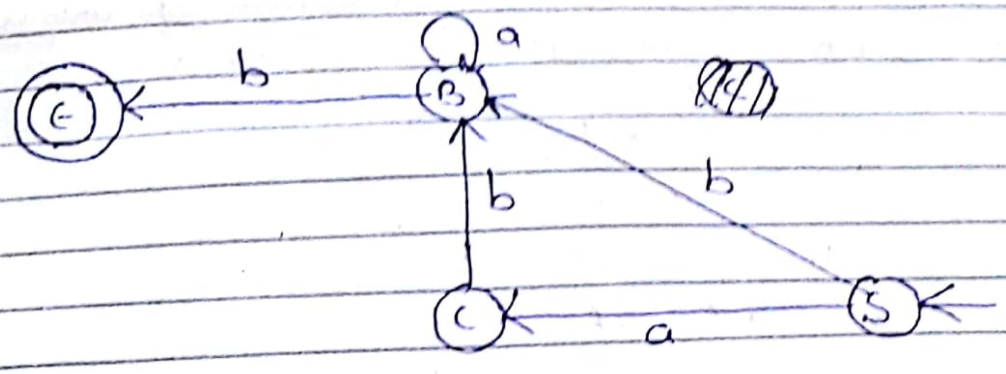


Representation of right linear grammar

Step-II)

Interchanging Initial & grammar using final states & reversing all the transitions.

Scanned by CamScanner



Step -III) We can rewrite the left-linear form of the grammar as follows,

$S \rightarrow Ca$
$S \rightarrow \cancel{bB}\ Bb$
$C \rightarrow Bb$
$B \rightarrow Ba$
$B \rightarrow b$

Let check it for a string "baab".
GR,

$S \Rightarrow bB \Rightarrow baB \Rightarrow baaB \Rightarrow baab$

Using above obtained GL, left-linear grammar we can derive the same string as,

$S \rightarrow Bb \Rightarrow Bab \Rightarrow Baab \Rightarrow baab$

↳ it generate the same string.

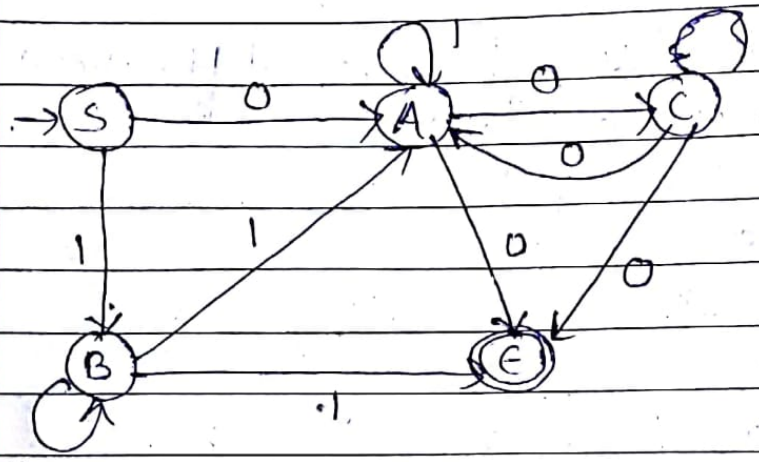
Scanned by CamScanner

Page No.
Date :

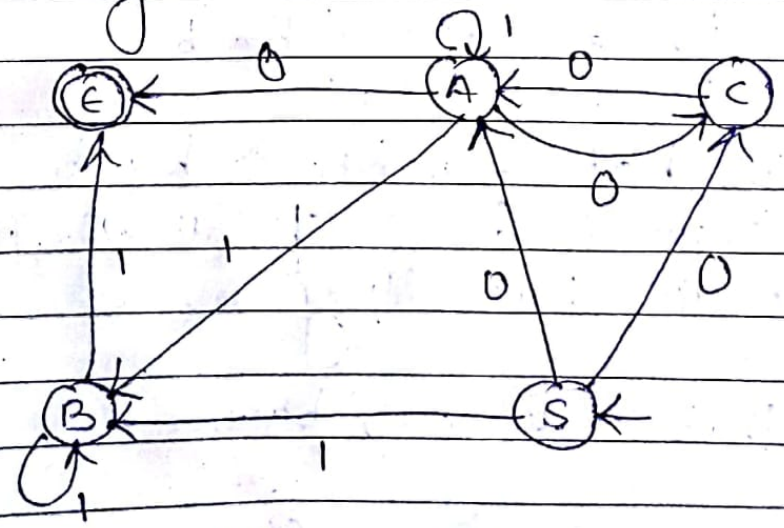
Q.2) Write an equivalent left-linear grammar for the right-linear grammar given as,

$S \rightarrow 0A \mid 1B$
$A \rightarrow 0C \mid 1A \mid 0$
$B \rightarrow 1B \mid 1A \mid 1$
$C \rightarrow 0 \mid 0A$

→



Change the State



$S \rightarrow B1 \mid A0 \mid C0$
$B \rightarrow B1 \mid 1$
$C \rightarrow A0$
$A \rightarrow A1 \mid B1 \mid C0 \mid 0$

Scanned by CamScanner

* Conversion of Left-Linear grammar to Equivalent Right-Linear Grammar

1.) Convert the following left-linear grammar to its equivalent right-linear grammar.

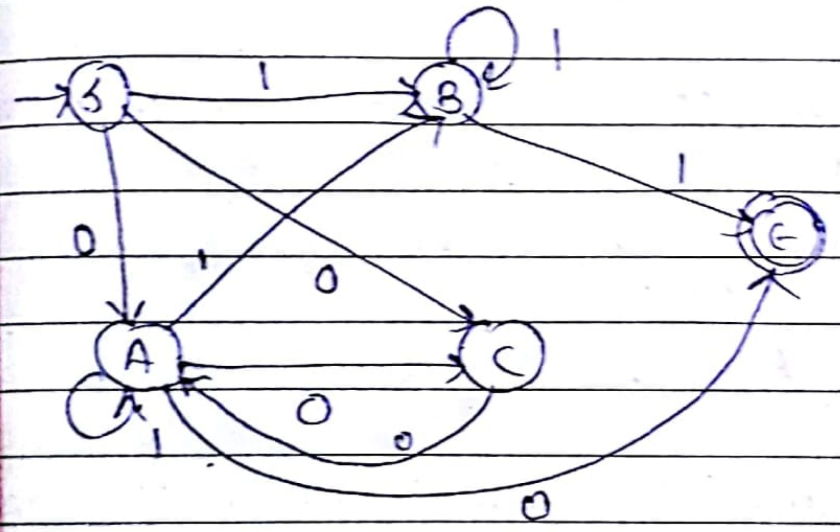
$S \rightarrow B1 \mid A0 \mid C0$

$A \rightarrow C0 \mid A1 \mid B1 \mid 0$

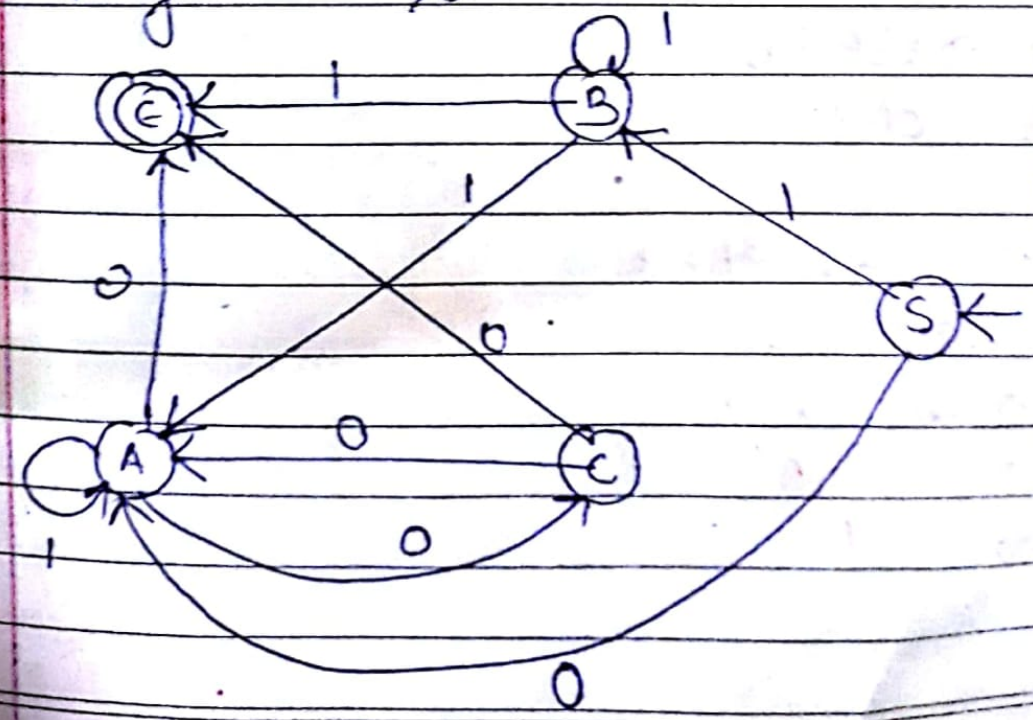
$B \rightarrow B1 \mid 1$

$C \rightarrow A0$

⇒



Change the State



$S \rightarrow 1B \mid 0A$

$B \rightarrow 1B \mid 1A \mid 1$

$A \rightarrow 1A \mid 0C \mid 0$

$C \rightarrow 0A \mid 0$

Scanned by CamScanner

L.L.G ⇒ move from Right to left

R.L.G ⇒ move from left to right

Q.2) Ans :- * Interconversion betn Regular expn & Regular Grammar

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

⇒

$0^*(10+11)^*$

i) We can produce the R.L.G as follows -

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

where,

$V = \{S, A\}$

$T = \{0, 1\}$

P = Production

S → Start State

P ⇒

$S \rightarrow 0S / A$

$A \rightarrow 10A / 11A / \epsilon$

ii) We can produce L.L.G as follows -

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

where,

$V = \{S, A\}$

$T = \{0, 1\}$

P ⇒

$S \rightarrow S11 / S10 / A$

$A \rightarrow A0 / \epsilon$

Scanned by CamScanner

ii) $L = \{a^n b^m \mid n \geq 2, m \geq 3\}$

$\Rightarrow$

$L = \{aa(a^*)bbb(b)^*\}$

i) We can produce the R.L.G as follows -

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

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

$P \Rightarrow$ $S \rightarrow aaA$

$A \rightarrow aA \mid bbbB$

$B \rightarrow bB \mid \epsilon$

ii) We can produce the L.L.G as follows -

$P \Rightarrow$ $S \rightarrow Sb \mid Abbb$

$A \rightarrow Aa \mid aa$

Q) Find the Regular grammar (RG) that generates the language, $L = (aa^*(ab+a)^*)$

$\Rightarrow$

i) We can construct the grammar by R.L.G

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

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

$P \Rightarrow$ $S \rightarrow aA$

$A \rightarrow aA \mid B$

$B \rightarrow abB \mid aB \mid \epsilon$

ii) For L.L.G

$P \Rightarrow$

$S \rightarrow Sab \mid Sa \mid A$

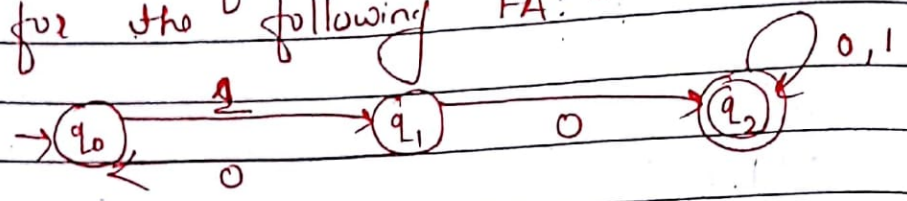
$A \rightarrow Aa \mid \epsilon$

Scanned by CamScanner

Page No.
Date :

* Interconversion betn Finite Automata (FA) & Regular Grammar

Q. → Find left-linear & right-linear grammar for the following FA:



⇒ $G = (V, T, P, S)$

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

$T = \{1, 0\}$

P →

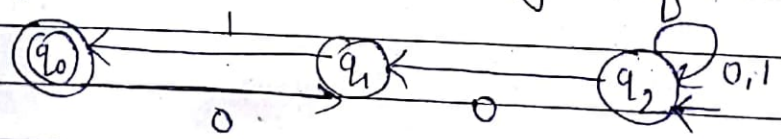
$q_0 \rightarrow 1q_1$

$q_1 \rightarrow 0q_0 \mid 0q_2 \mid 0$

$q_2 \rightarrow 0q_2 \mid 1q_2 \mid 0 \mid 1$

& $S = q_0$ (start symbol)

To find LLG first we reverse the transition arrows of given DFA & interchange start state & final state. We will get following DFA.



We can write productions for the DFA

$q_2 \rightarrow 0q_2 \mid 1q_2 \mid 0q_1$

$q_1 \rightarrow 1q_0 \mid 1$

$q_0 \rightarrow 0q_1$

Scanned by CamScanner

After reversing the productions we will get final LLG as follows,

$G = \{V, T, P, S\}$

where,

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

$T = \{0, 1\}$

$P \Rightarrow$

$q_2 \rightarrow q_2 0 \mid q_2 1 \mid q_1 0$

$q_1 \rightarrow q_0 1 \mid 1$

$q_0 \rightarrow q_1 0$

& $S = q_2$ (start symbol)

Scanned by CamScanner