Regular Languages

Can be represented by
1) Regular expression
2) Finite Automata
3) Regular grammar

Regular expression:- Given Σ be an alphabetical.

1. $a \in \Sigma$ is a regular expression.
2. ϵ and ϕ are regular expressions
3. If r_1 and r_2 are regular expression, then
   a) $r_1 + r_2$ is a regular expression representing $L_1 \cup L_2$.
   b) $r_1 r_2$ is a " " " $(L_1)(L_2)$
   c) r_1^* is " " " " L_1^*

e.g. $\Sigma = \{0, 1\}$

1. 0 is a regular expression $L_1 = \{0\}$
   1 is a " " $L_2 = \{1\}$
2. ϵ is a " "
   (null string)
   ϕ is a " "
   (null language)
3. $0+1$ is a regular expression $L_3 = \{0, 1\}$ $\quad L_3 = L_1 \cup L_2$
   concatenation 0.1 " " " $L_4 = \{01\}$

(KLEENE CLOSURE) $\leftarrow$ 0^* → zero or more repeat

$\{\epsilon, 0, 00, 000, \ldots\} \leftarrow L_1^*$

$L_1 = \{0, 00, 01, 10\}$

$L_2 = \{2, 3, 23\}$

$L_1 \cup L_2 = \{0, 00, 01, 10, 2, 3, 23\}$

$L_1 L_2 = \{02, 03, 023, 002, 003, 0023, 012, 013, 0123, 102, 103, 10232\}$

$L_1^* = \{0, 1\}$

$\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$

$L_1^* = \{\epsilon, 0, 00, 01, 10, 000, 001, 010, 0001, 0010, 0110\}$

$L_1 = \{0^x, 1^y\}$

$L_1^* = \{\epsilon, x, y, xy, yx, xx, yy, \ldots\}$

$L_1^* = \{\epsilon, 0, 11, 011, 110, 00, 1111, \ldots\}$

ans Design a regular expression that accept all strings over $\Sigma = \{a, b\}$.

$L = \{\epsilon, a, b, aa, ab, ba, bb, \ldots\} = (a+b)^*$

Ques Design a regular expression over $\Sigma = \{a, b\}$ that accept all strings that starts with a.

Soln → $L = \{a, ab, aa, aab, aaa, aba, abb, \ldots\}$

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

Ques Design a regular expression over $\Sigma = \{a, b\}$ that starts with a and ends with b.

Soln → $L = \{ab, aab, abb, aaab, aabb, abab, abbb, \ldots\}$

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

Ques Design a regular expression over $\Sigma = \{a, b\}$ that all strings having 1st & last letter diff.

Soln → $L = \{ab, ba, aab \ldots\}$

$L = L_1 + L_2 = a(a+b)^*b + b(a+b)^*a$

Ques Design a regular expression that accept all strings having atleast 1 'a'.

Soln → $L = \{a, ab, ba, aab, abb, aa, aaa, \ldots\}$

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

Ques Exactly one 'a'.

$L = \{a, ab, ba, \ldots\}$

$L = b^*ab^*$

Ques Atleast two 'a'.

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

Ques Exactly two a

$b^*ab^*ab^*$

Ques Atmost one a.

$b^* + b^*ab^*$

Ques Atmost 2 a.

$b^* + b^*ab^* + b^*ab^*ab^*$

DFA (DETERMINISTIC FINITE AUTOMATA)

For a Regular Language, we can define DFA, Regular Expression, Regular Grammar

DFA is a quadruple $(Q, \Sigma, q_0, \delta, F)$ where

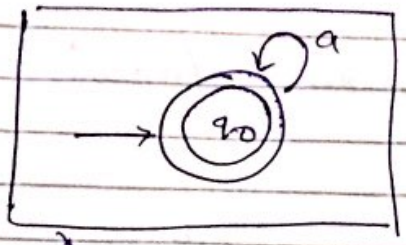
Q is a finite non-empty set of states.
Σ is a finite non-empty set of alphabets.
q_0 is an initial state s.t. $q_0 \in Q$.
δ is a transition function which is represented by transition diagram or table.
F is a finite non-empty set of states s.t. $F \subseteq Q$.

Ques Design DFA that accept all string over $\Sigma = \{a, b\}$

$L = \{\epsilon, a, b, \ldots\}$

$\Sigma = \{a\}$

$L = \{\epsilon, a, aa, \ldots\}$



Double circle
⇓
Final State

ϵ, a, aa → accepted

b → rejected.

$Q = \{q_0\}$

$\Sigma = \{a\}$

$F = \{q_0\}$

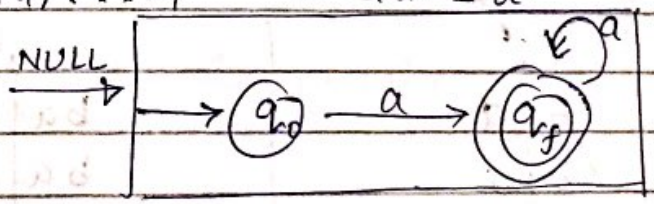
δ

Q) Design DFA that accept all strings of $l \geq 1$ over $\Sigma = \{a\}$

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

a	ϵ
a	a
a	a a
a	a a a
a	a a a a

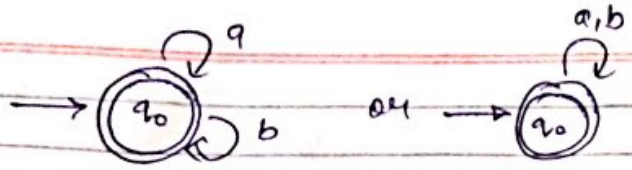
a^*



if NULL is input,
it can't be transited to q_f.
∴ it is rejected.

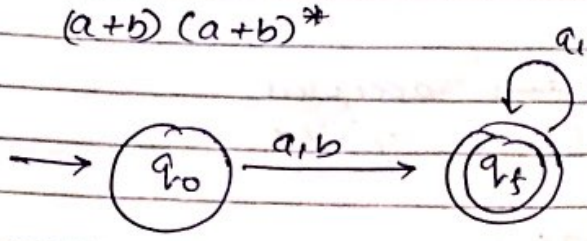
There can be more than 1 final states.

Ques



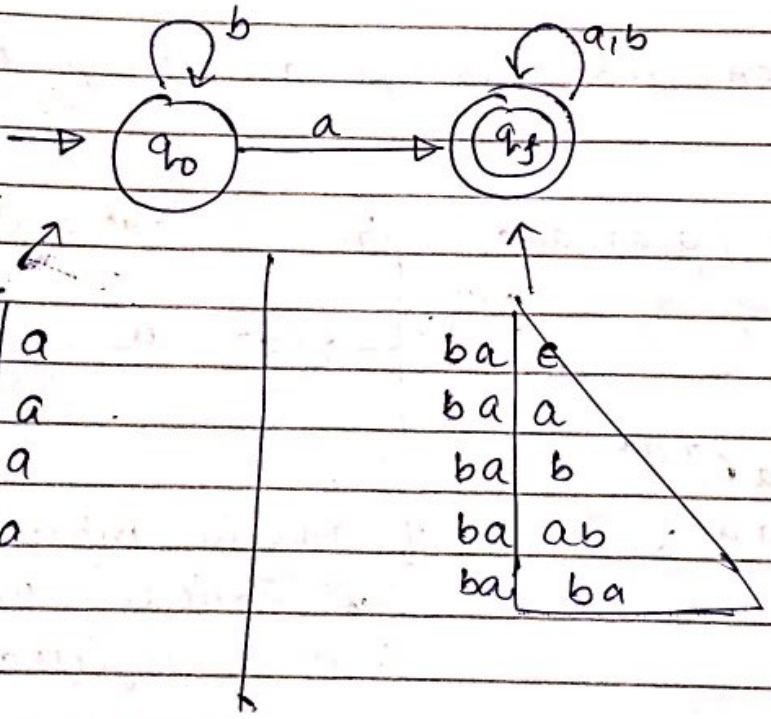
Regular expression = $(a+b)^*$

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

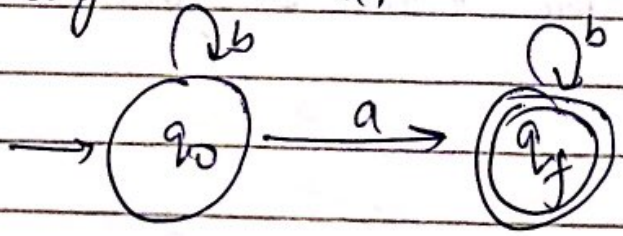


Ques Design DFA that accept all string having at least one a over $\Sigma = \{a, b\}$

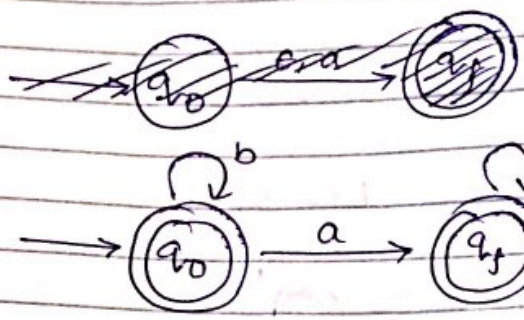
Regular expression → $(a+b)^* a (a+b)^*$



Ques Exactly one a. b^*ab^*

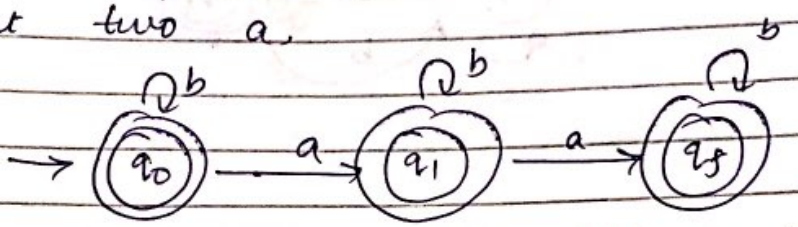


Ques almost one a.



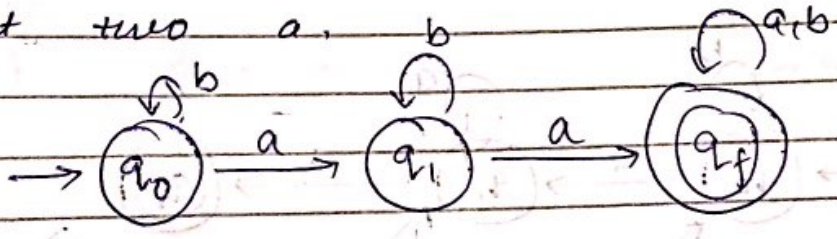
b^* b^*ab^*

Ques almost two a.

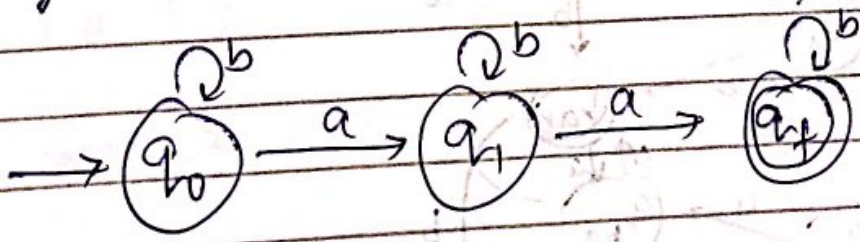


b^* b^*ab^* $b^*ab^*ab^*$

Ques atleast two a.



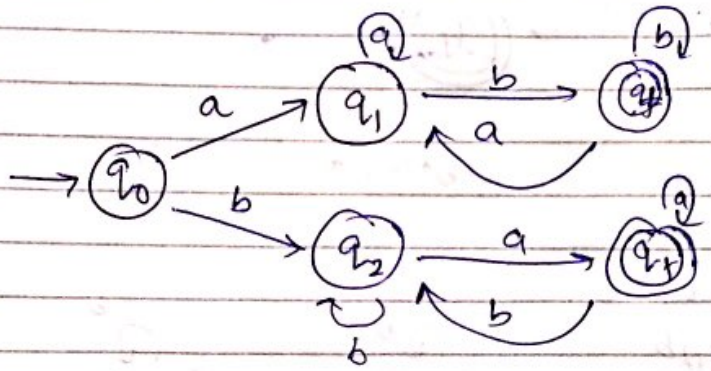
Ques exactly two a



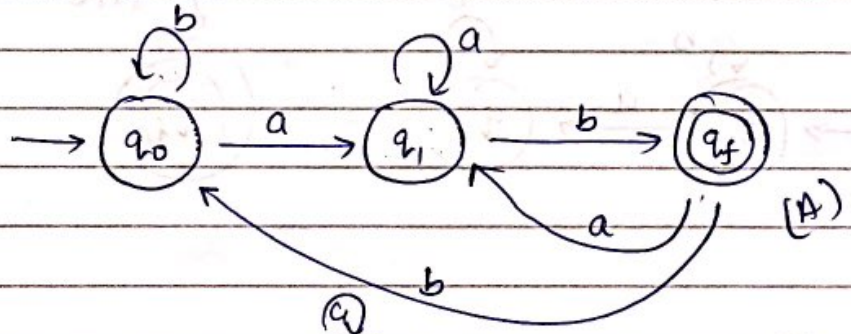
Ques all strings having diff. starting and last symbol over $\Sigma = \{a, b\}$.

$L = \{ab, ba, \ldots\}$

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



Ques Design a DFA that accept all strings ending with ab.

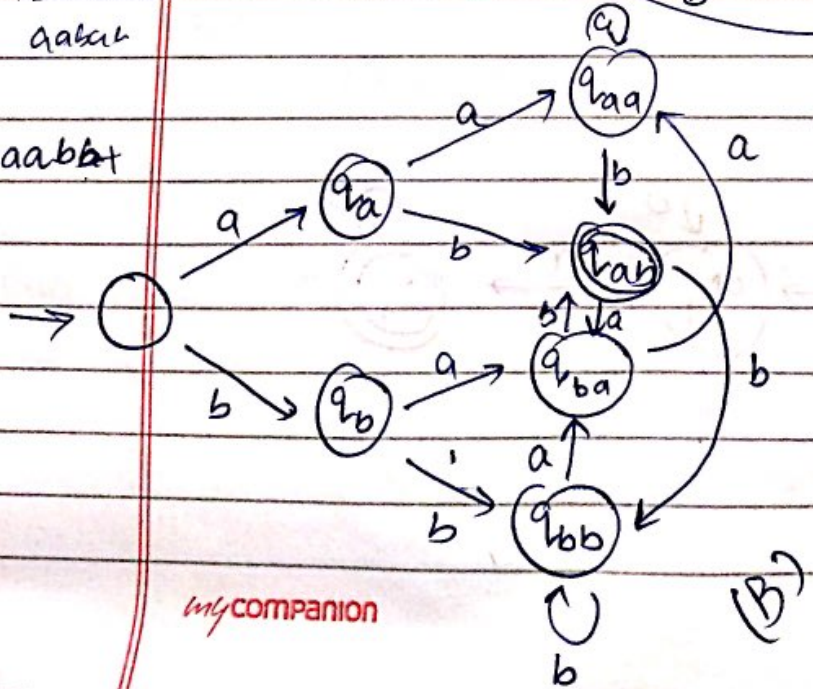


xxf ab

aaba

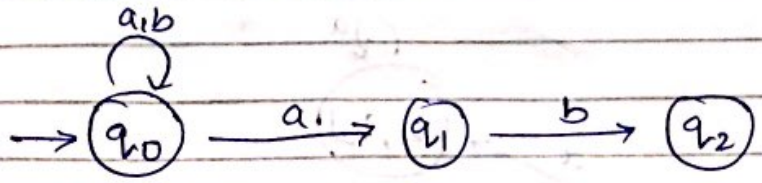
baabb

OR



B → A
(Minimization Process)

NDFA → DFA

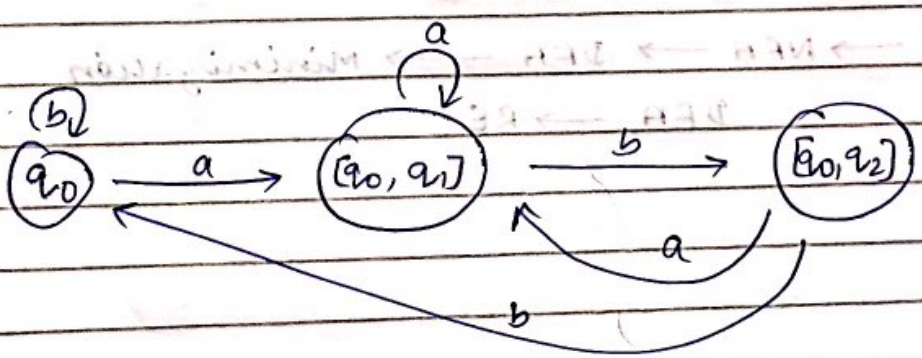


	a	b
q_0	q_0, q_1	q_0
q_1	—	q_2
q_2	—	—

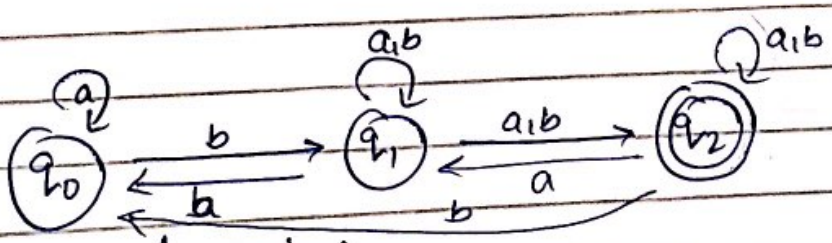
Table (1)
(NFA)

	a	b
→ q_0	$[q_0, q_1]$	q_0
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_2]$
$[q_0, q_2]$	$[q_0, q_1]$	q_0

{union of q_0, q_1 entries in Table (1)}

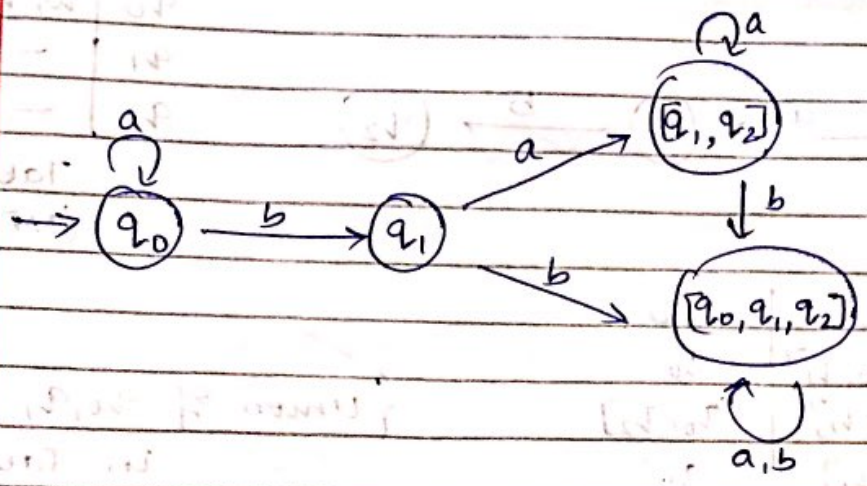


Ques



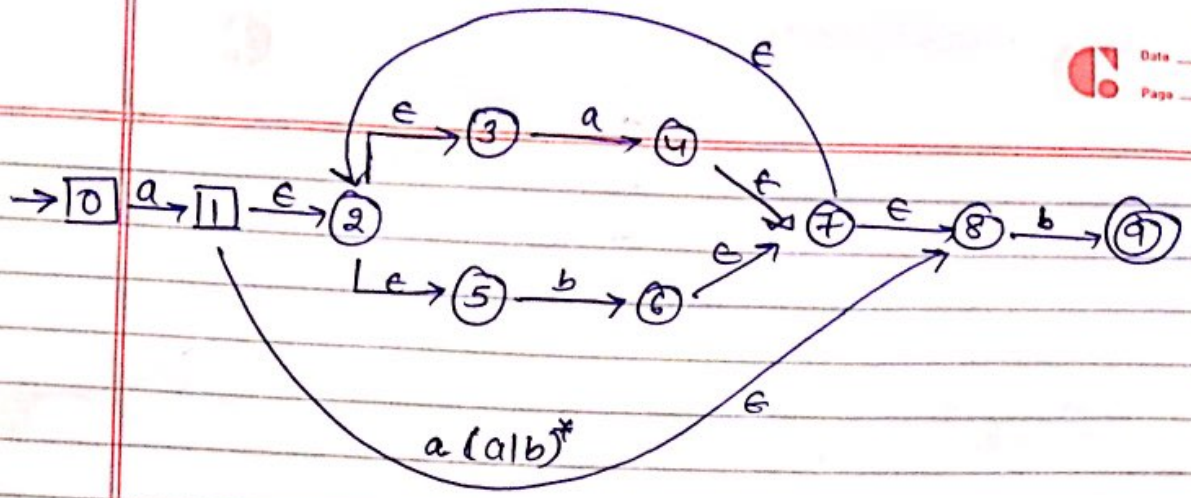
	a	b
q_0	q_0	q_1
q_1	~~q_0~~, q_1, q_2	q_1, q_2, q_0
q_2	q_1, q_2	q_0, q_2

	a	b
q_0	q_0	q_1
q_1	$[q_1, q_2]$	$[q_0, q_1, q_2]$
$[q_1, q_2]$	$[q_1, q_2]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$



RE → NFA → DFA → Minimization

DFA → RE



ε-closure(0) = {0} = (A)

a on A A = {0}

0 —a→ 1

ε-closure(1) = {1,2,3,5,8}

b on A = {0}

B = {1,2,3,5,8}

a on B

3 —a→ 4

ε-closure(4) = {4,7,8,2,3,5} = C

b on B

5 —b→ 6 8 —b→ 9

ε-closure(6,9) ≡ ε-closure(6) ∪ ε-closure(9)

= {7,8,2,3,5,6,9} = D

a on C = {2,3,4,5,7,8}

3 —a→ 4

ε-closure(4) = C

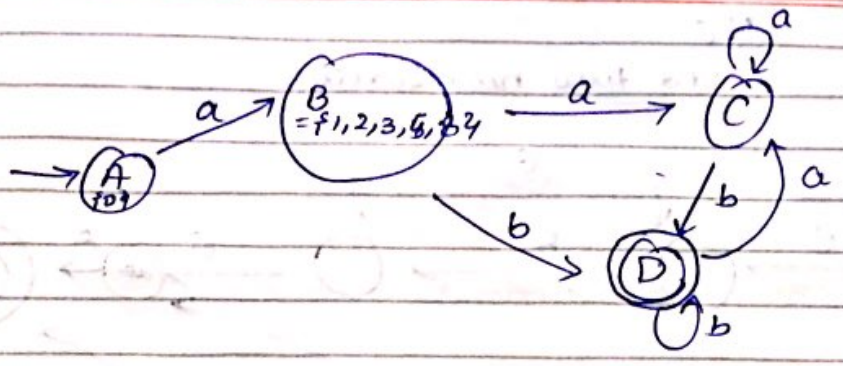
b on C = {2,3,4,5,7,8}

5 —b→ 6, 8 —b→ 9

ε-closure(6,9) = D

fun(void) {cout << "hello"}

main
{ a = fun(5,7) }



a on D = {2, 3, 5, 6, 7, 8, 9}

$3 \xrightarrow{a} 4$

ε-closure(4) = C.

b on D = {2, 3, 5, 6, 7, 8, 9}

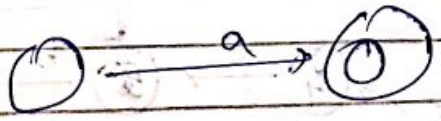
$5 \xrightarrow{b} 6, 8 \xrightarrow{b} 9$

Set containing final state "a" will be final i.e., 'D'.

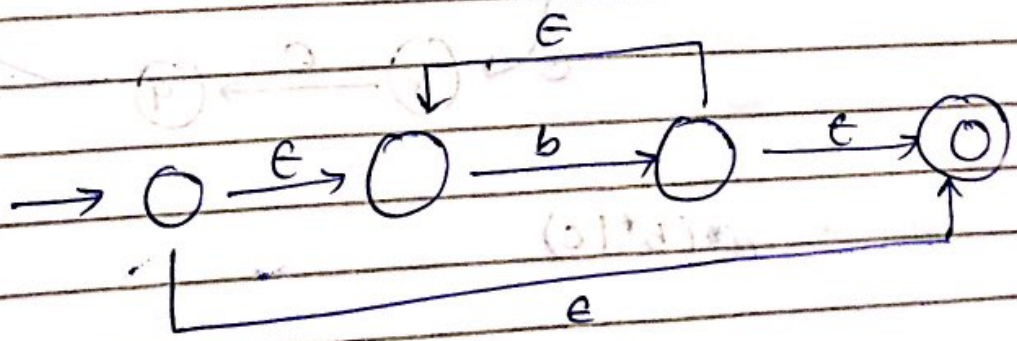
Ques Construct NFA for the following

a* (b* | c)

union



Kleen closure add two new state.



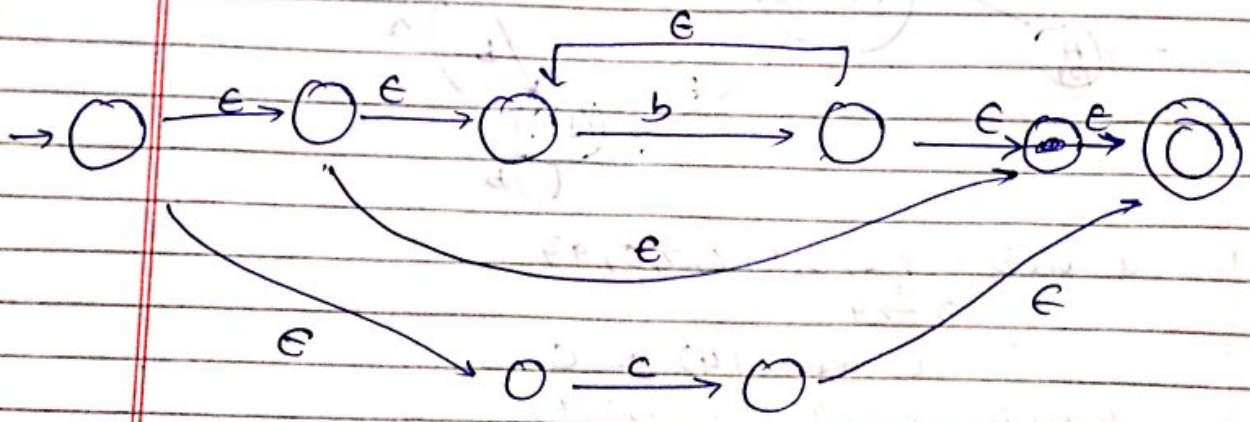
b*

(11)

Ques

$b^*|c$

∴ add two new states



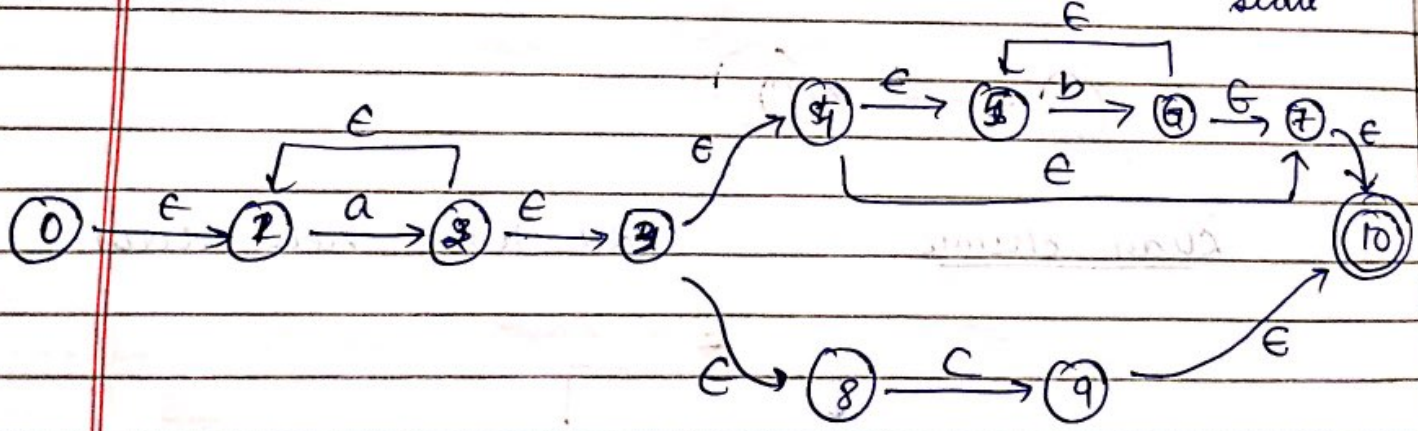
$b^*|c$

$a^*(b^*|c)$

↓

~~Contact~~ concatenation

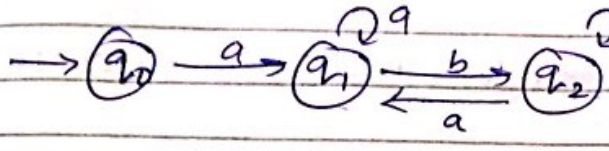
Merging of 1's final & $(b^*|c)$'s initial state



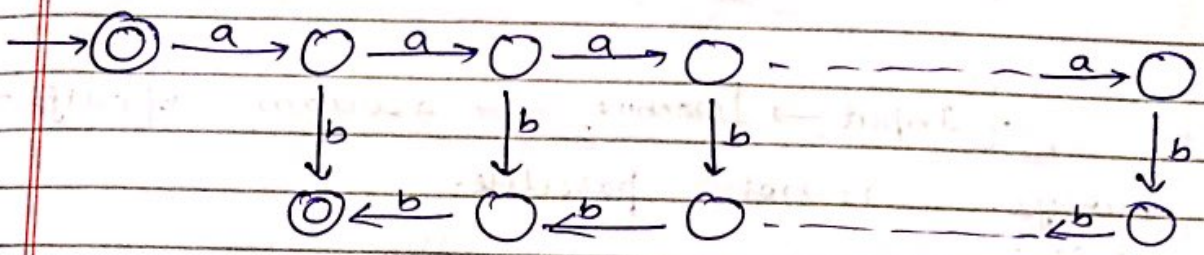
$a^*(b^*|c)$

Ques Design a DFA for

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



This cannot determine the count of 'a', & 'b'.



But DFA should have finite no. of states.
Thus, this is also not a DFA.

Conclusion :- DFA for this language is not possible.

If a language is regular, then we can write regular expression, DFA, NFA.
In this case, L is not regular i.e., we can't construct a DFA for it.

Ques Difference b/w ϕ and ϵ.

Merly and Moore machine

Limitations of DFA :-

1. It can't memorize. ∵ → PushDown Automata.
2. $w \rightarrow$ [DFA] → accepted / → rejected

Binary output for an input string (w)

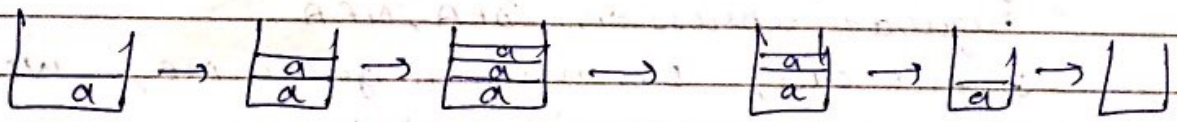
Input → [Machine] ⟶ a stream of output

Merly2 Moore machine. it's not possible.

3. We are can't determine the middle point of the string.

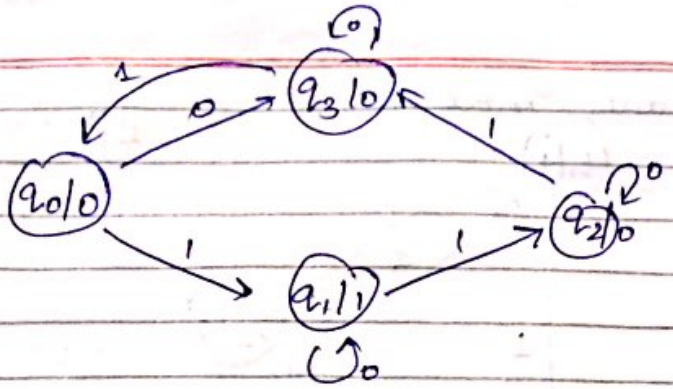
Pushdown Automata — uses stack

aaabbb



	a = 0	a = 1	output
q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

Moore Machine → Output depends only on state.



Input string w = 0110 $\quad |w| = 4$

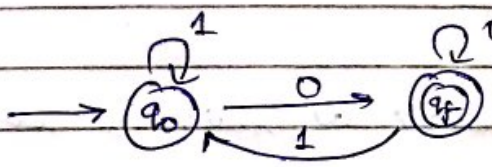
$q_0\ q_3\ q_0\ q_1\ q_1$

(0)(0)(0)(1)(1) $\quad |\text{output}| = 5$

length of output stream = length of input stream + 1.

Ques Design a DFA over $\Sigma = \{0, 1\}$ s.t. decimal equivalent is divisible by '2'.

Soln →

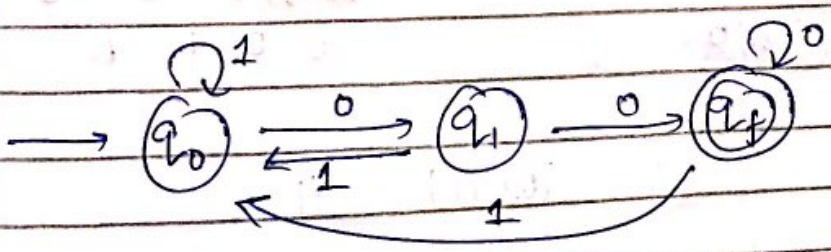


~~2, 4, 8, 12, 16~~

16	8	4	2	1
		1	0	0
	1	0	0	0
	1	1	0	0
1	0	0	0	0
1	0	1	0	0

ending with '00'.

Divisible by '4', 8, 16

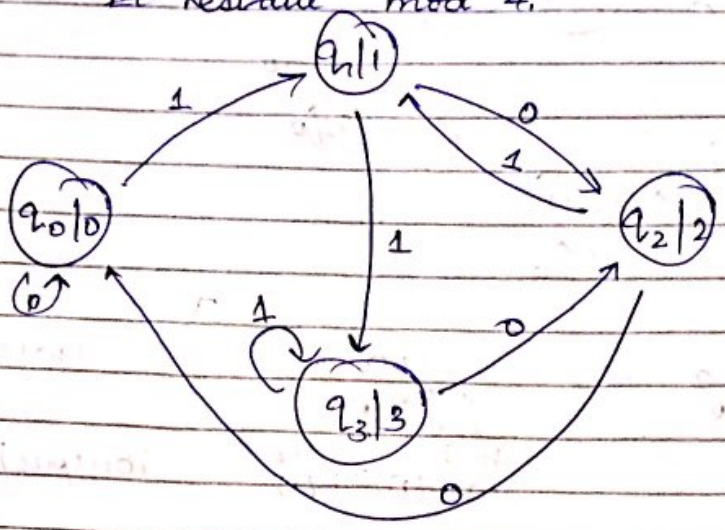


Moore machine:- By determining the output, we can tell whether it is divisible by 'n' or not.

(Moore machine)

Binary

Residue mod 4.



110 (6)

111 (7)

10

11

100

10 → 2

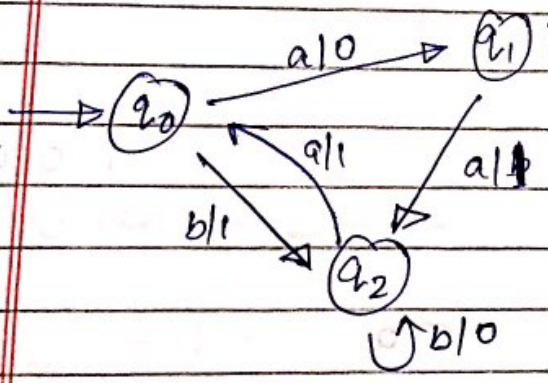
|01| 01 → 1

10 → 2 mod 4 = 2

Mealy Machine

No final state.

Output depends on state as well as input.



w = abaa

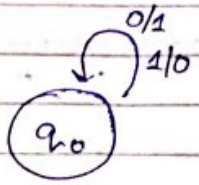
q_0 q_1 q_1 q_2 q_0

0 0 1 1

|w| = 4 |0011| = 4

If we have an input string of length 'n', then output string will have length 'n'.

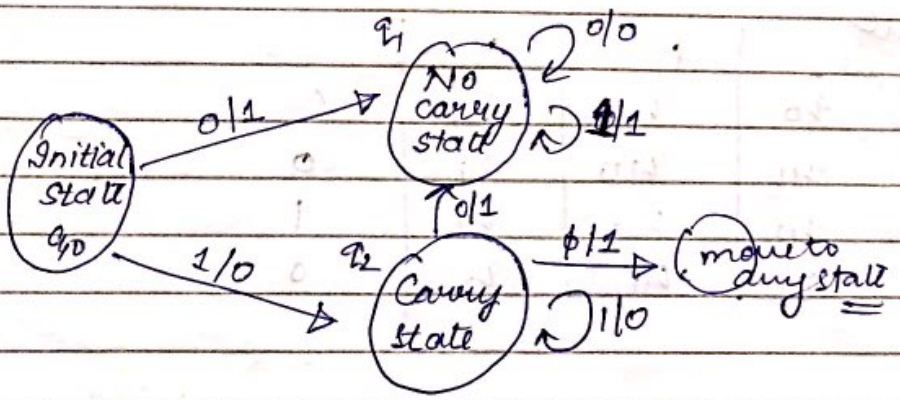
Ques Design Mealy machine which print 1's compliment of input bit string over alphabet $\Sigma = \{0, 1\}$



Input :- 0 1 1 1
Output 1 0 0 0

Ques Design a Mealy Machine which works an increment machine that assumes its input as a binary no. and prints out binary no. that is one greater than input.

Soln →



ϕ 1 1 1
+ 1
1 0 0 0

ϕ | ϕ | 1 | 0 | 0 | 0 | Δ | Δ | .. tape

Initial State	0		1	
q_0	q_1	1	q_2	0
q_1	q_1	0	q_1	1
q_2	q_1	1	q_2	0

(Transition Table)

$q_1 \rightarrow 1, 0$

$q_2 \rightarrow 0$

If $n+1$ outputs are possible,
q_i can be associated with
$q_i \rightarrow 0, 1, 2, \ldots, n$

So, q_i will be defined by $n+1$ states.

q_{i0}	0 (output)
q_{i1}	1
q_{i2}	2
q_{i3}	3
⋮	⋮
q_{in}	n

$\Sigma = \{1, 2, \ldots, n\}$ $\quad |\Sigma| = n$

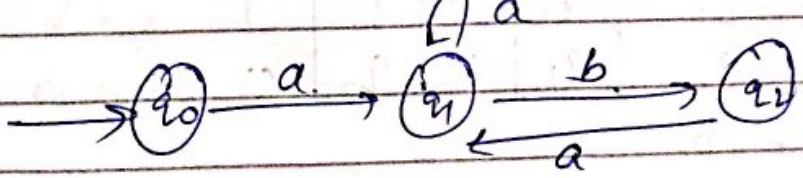
$Q' = \{q_1, q_2, \ldots, q_m\}$ $\quad |Q| = m$

Moore to Mealy machine

	$a=0$	$a=1$	
q_0	q_{11}	q_2	^
q_{10}	q_{10}	q_{11}	0
q_{11}	q_{10}	q_{11}	1
q_2	q_{11}	q_2	0

Pumping Lemma (Based on Pigeon Hole Principle)

∑ and no. of states = 3.
$|w| \geq 3$ and $|w| = 3$ e.g. aab
Then atleast one state will be visited again.

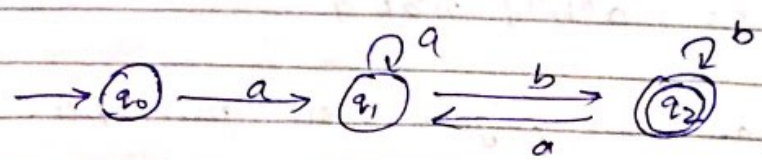


Let L be a regular language, then we can design a DFA with n states.
Consider, $w \in L$, w can be divided into x, y, z such that $w = xyz$.
Following condns are satisfied :-

i) $|y| > 0$

ii) $|xy| \leq n$

iii) $xy^i z \in L$ for $i \geq 0$



States = 3 $\quad a(a+b)^* b$

w = aab $\quad$ w = abb $\quad$ w = abab

w = xyz $\quad$ w = xyz $\quad$ w = xyz

y → on repeating we reach the same state.

'b' → we loop at q_2.

$w = \overbrace{ab}^{x}\overbrace{b}^{y}$

$z = \epsilon$

$i = 1 \quad xy^1z = abb\epsilon$

$i = 0 \quad xy^0z = ab\epsilon\epsilon$

$i = 2 \quad xy^2z = abbb\epsilon$

$w = \overbrace{aa}^{x}\overbrace{a}^{y}\overbrace{b}^{z}$

↳ y ;

$i = 0 \quad ab$ ✓

$i = 1 \quad aab$ ✓

$i = 2 \quad aaab$ ✓

$w = abab$ (x, y, z)

Pumping lemma :-

1. It is not used to prove that a language is regular.
2. It is used to prove that a language is non-regular.

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

3. It is based on pigeon hole principle.

Ques P.T., $L = \{ a^n b^m \mid n \geq 0 \}$ is not a regular langu

Soln :-

STEP 1 :- Let L be a regular language, we can be able to design a DFA with n states.

STEP 2 :- choose $w = a^k b^k$

Case 1 :- $y = a^l \mid l \geq 1$ & $l \leq n$

Case 2 :- $y = ab$

Case 3 :- $y = b^l \mid l \geq 1$ & $|xy| \leq n$

$$\underbrace{a \ldots a}_{x} \underbrace{ab \ldots b}_{y} \underbrace{\ldots b}_{z}$$

$$w = \underbrace{a^{k-l}}_{x} \underbrace{a^{l}}_{y} \underbrace{b^{k}}_{z} \quad | \quad l \geq 1$$

$xy^i z \notin L$

If $i = 0$

$xz \notin L \rightarrow a^{k-l} b^k \notin L$

$\therefore$

case 2 :- $y = ab$

$\underbrace{a^{k-1}}_{x}\ \underbrace{ab}_{y}\ \underbrace{b^{k-1}}_{z}$

$xyyz = a^{k-1}ababb^{k-1} \notin L$

case 3 :- $\underbrace{a^{k}}_{x}\ \underbrace{b^{l}}_{y}\ \underbrace{b^{k-l}}_{z}$

$xz = a^{k}b^{k-l}$ l will be at least 1

$xz \notin L$.

Qu P.T: $L = \{a^{i^2} \mid i \geq 1\}$ is not a regular lang.

$L = \{a, a^4, a^9, a^{16}, \ldots\}$

Soln → STEP 1 :- same.

STEP 2 :- $w = a^{n^2}$

$|w| \geq n^2 > n$

$y = a^l \mid l > 1$ and $l \leq n$

STEP 3 :-

$|xy^2z| = |xyyz| = |xyz| + |y|$

$|xy^2z| = |xyyz| = \underbrace{|xyz| + |y|}_{n^2 \quad 1.\ (\min)} \rightarrow \geq n^2$

$= \underbrace{|xyz| + |y|}_{n^2 \quad n} > n^2.$

$n^2 < |xy^2z| < (n+1)^2$

$\therefore (xy^2z) \notin L$

Ques $L = \{a^p \mid p \text{ is a prime number}\}$ is not a regular language.

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

Step 1:- Let L be a regular language with n states DFA.

Step 2:- $w \in L$, $|w| \geq n$

$w = a^q$ with condition $q \geq n$

$w = xyz$ $|y| > 0$ & $|xy| \leq n$

$y = a^k$ with condn. $k \geq 1$ & $k \leq n$

Now $|xy^i z| = |xyz| + |y^{i-1}|$ $\quad i \geq 0$

$= |xyz| + (i-1)|y|$

$= q + (i-1)(k)$

$= q + qk$

$= q + (q+1-1)k$

Suppose $i = 1 = q$

$|xy^i z| = q(1+k)$

If $k = 2$ then it will be divisible by 3. But should be a prime number.

$\Rightarrow xy^i z \notin L$

Hence Proved

DFA to RE ~~regu~~ conversion

1. Using Arden's Theorem.
2. Using state elimination method.
3. Transitive closure.

ARDEN THEOREM

Let P and Q be regular expression over Σ. If P doesn't contain ε, and following eqn hold on R,

$$R = Q + RP,$$

then it has a unique soln,

$$R = QP^*$$

Proof:-

$$R = Q + RP$$

$$R = Q + [Q + RP]P$$

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

$$R = Q + QP + [Q + RP]P^2$$

$$R = Q + QP + QP^2 + RP^3$$

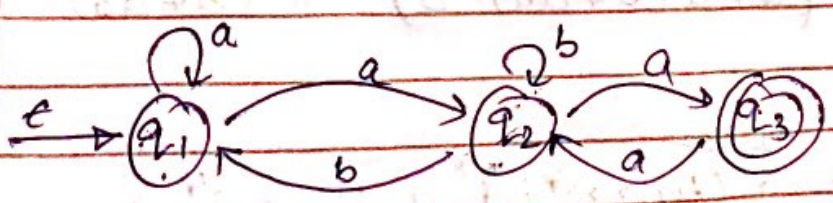
$$R = Q + QP + QP^2 + QP^3 + \cdots + QP^{n-1} + RP^n$$

$$R = Q(\varepsilon + P + P^2 + P^3 + \cdots + P^{n-1})$$

RP^n ⇓ neglect

$$R = QP^*$$

Ques



Eqns:- $q_1 = q_1 a + q_2 b + \varepsilon$ — (1)

If u Don't write

$q_1 = aq_1 + bq_2 + \varepsilon$ →

$q_2 = q_1 a + q_2 b + q_3 a$ — (2)

$q_3 = q_2 a$ — (3)

Substitute value of q_3 in (2).

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

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

$R = Q + RP$

$R = QP^*$

$R = q_1 a (b+aa)^*$

~~$q_1 = \epsilon + q_1 a + q_1 a + q_1 a +$~~

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

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

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

$R = Q + RP$

$R = QP^*$

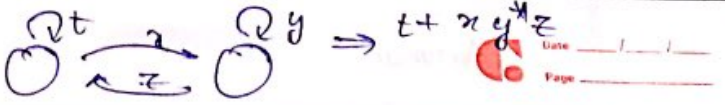
$q_1 = R = \epsilon (a + a(b+aa)^* b)^*$

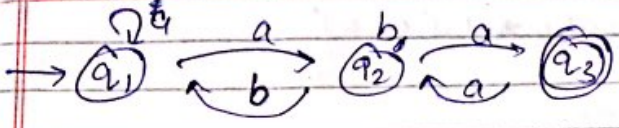
$q_2 = q_1 a (b+aa)^*$

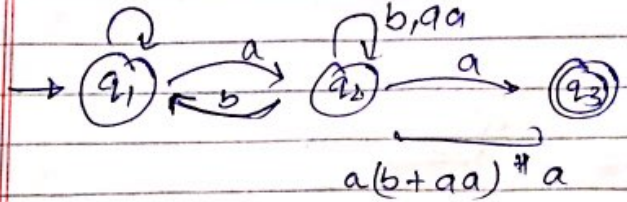
$q_2 = [a + a(b+aa)^* b]^* a (b+aa)^*$

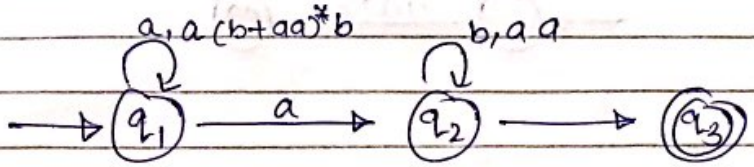
$q_3 = q_2 a$

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

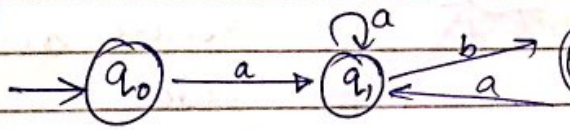








e.g.



$q_0 = \epsilon$

$q_1 = q_0 a + q_1 a + q_2 a$

$q_2 = q_1 b + q_2 b$

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

$q_1 = a + q_1 a + q_2 a$

$q_1 = \underbrace{(a + q_2 a)}_{Q} + \underbrace{q_1 a}_{R\ P}$ (q_1 = R)

$q_1 = R = QP^* = (a + q_2 a)\, a^*$

$q_2 = ((a + q_2 a) a^*)\, b + q_2 b$

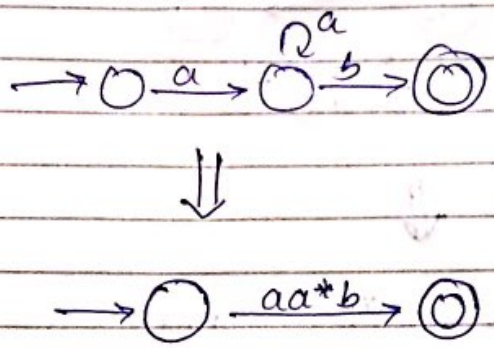
$q_2 = \underbrace{aa^* b}_{Q} + \underbrace{q_2}_{R}\underbrace{(aa^* b + b)}_{P}$ (q_2 = R)

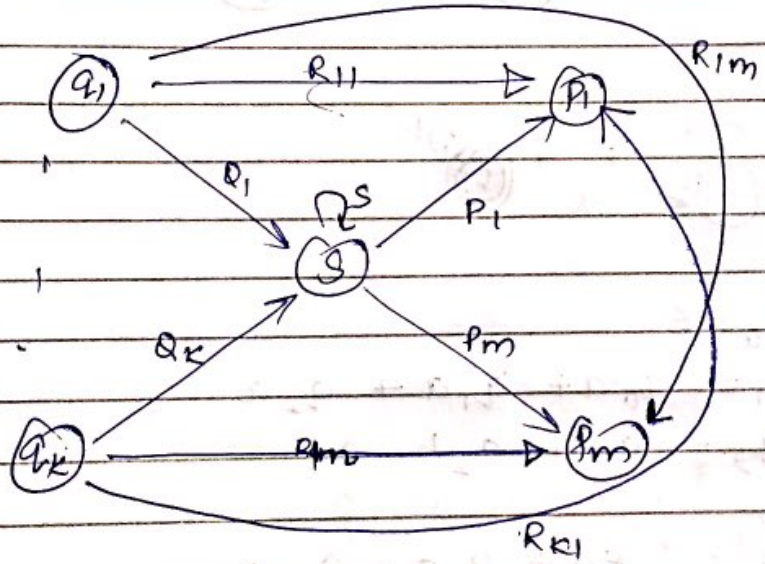
Vitlman

$r_2 = (aa^*b)(aa^*b+b)^*$

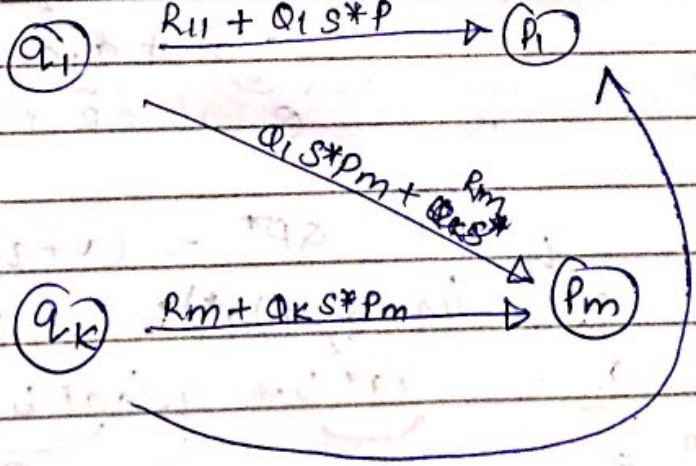
STATE ELIMINATION

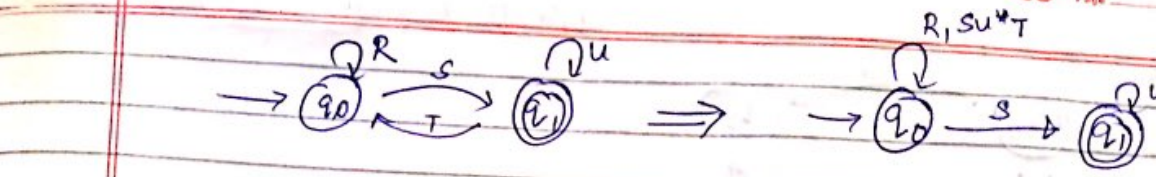
1. First try to eliminate non-final, non-starting state by one.



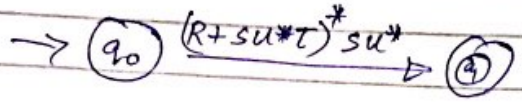


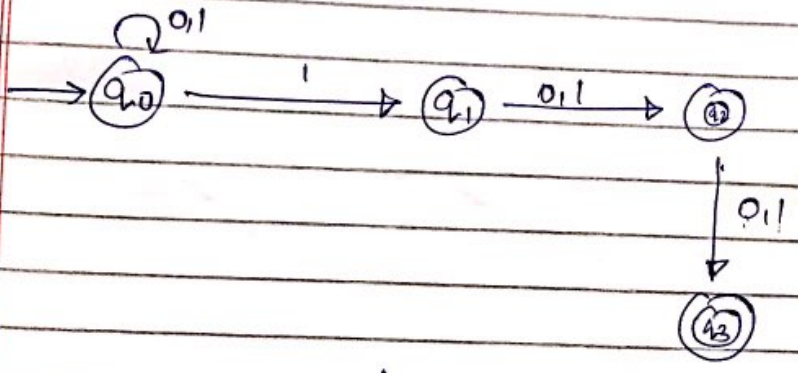
there are more states. Eliminate s.

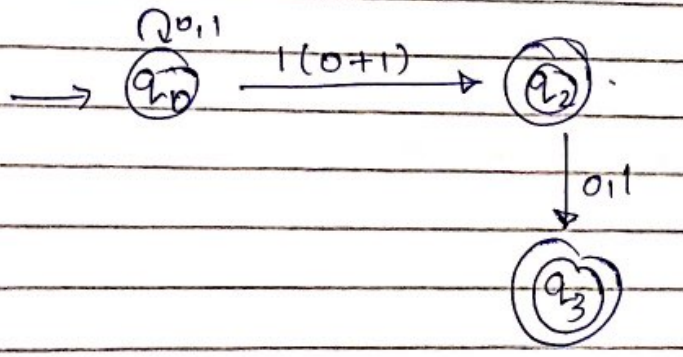


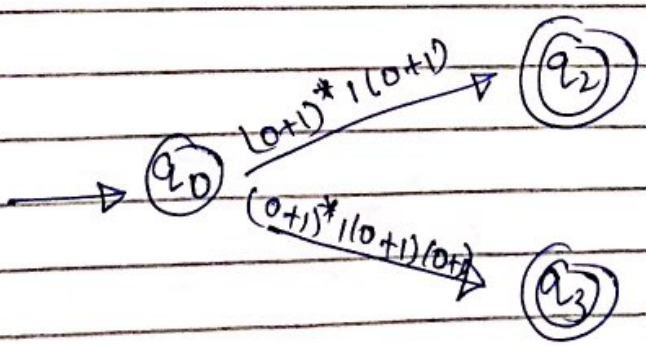


$$((R+SU^*T)^*)(SU^*)$$









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