Module 2

Regular Expression.

Kavya.S
Asst. prof
CDesign dept.
PESITM

⇒ Regular Expression:

* "Regular expression is a simple expression used to describe the language accepted by the finite Automata."
* It is an effective way to represent any language
* They are used to represent certain set of strings in an algebraic fashion
* Symbols we use 0, 1, a, b, ∈, ϕ etc are Regular expression
* union of 2 Regular expression is also Regular Expression.
* concatination of 2 RE's is also RE
* Iteration (closure) of a Regular expression is also a R.E i.e if 'b' is Regular expression then b^* is also Regular expression.

Application of Regular Expression

(1) Lexical Analysis:

Task of lexical analyser is to scan the input code and separate out the tokens.

OR: Identifier is a category of token in the source code and it can be identified by Regular Expression

(letter) (letter+digit)*

② Finding patterns in the text

Regular expression is used for finding patterns from the given text. (pdf, doc, file or any document)

ex: strings ending with digits is a pattern in the given text which can be recognised by the following R.E as [a-z A-Z]* [0-9]+

③ In unix

egrep is the tool (command) used for searching text pattern in the file and list of words containing that pattern.

ex: Assume, file xyz is having the following data

Ram, Dexter, Raman, Thon

% egrep 'n' xyz → This will search all words in file xyz having letter 'n'

o/p: raman
Thon

Problems on Regular expression

(1) write Regular expression denoting language with strings having any number of a's over $\Sigma = \{a\}$

$$RE = a^*$$

(2) write RE to accept all strings having either a or b over $\Sigma = \{a, b\}$ of length '1'

$$RE = a + b$$

(3) write RE to accept all strings that contains any combination of a and b of length 2 $\Sigma = \{a, b\}$

$$RE = (a+b)(a+b) \text{ or } (a+b)^2$$

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

(4) write Regular RE to accept all strings that contains any combination of a's and b's of length 3 $\Sigma = \{a, b\}$

$$L = \{aaa, aba, abb, bab, bba, abb---\}$$

$$RE = (a+b)(a+b)(a+b)$$

(5) String that contains any combination of 'a' and 'b' of any length

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

(6) String start with 'a' and ends with 'b'

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

(7) Strings begin with ab over $\Sigma = \{a, b\}$

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

(8) String end with ba

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

(9) String containing substring 'a' $\Sigma = \{a, b\}$ i.e a must be present in the string and before a & after a anything can be present.

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

(10) String containing one more a's over $\Sigma = \{a, b\}$

$$\boxed{RE = aa^*}$$

(11) String length of length atleast 3 over $\Sigma = \{a, b\}$ i.e $|w| \geq 3$

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

(12) Represent string of a's & b's having odd length

$$RE = \underbrace{((a+b)(a+b))^*}_{\text{this will give even}} (a+b) \quad + \text{one odd length}$$

(13) String of odd length over $\Sigma = \{1\}$

$L = \{1, 111, 11111, 1111111111 \cdots\}$

$$RE = 1(11)^* \text{ or } (11)^* 1$$

(14) String length atleast 3 over $\Sigma = \{a\}$ (3)

$$RE = aaaa^*$$

(15) string of even length over $\Sigma = \{a, b\}$

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

$$L = \{\in, aa, ab, aaba, abba, abbb\cdots\}$$

(16) string of length atmost 2 over $\Sigma = \{a, b\}$ i.e maximum length of string = 2. not more than 2

$$RE = \{\underset{\text{length 0}}{\in} + \underset{\text{length 1}}{a+b} + \underset{\text{length 2}}{(a+b)(a+b)}\}$$

(17) string start over $\Sigma = \{0, 1\}$ that contain 1

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

(18) String having exactly 2 a's over $\Sigma = \{a, b\}$

$$RE = b^* a b^* a b^*$$

(19) String of odd length

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

20) w corresponds to the binary encoding without leading 0's of natural number that are evenly divisible by 4.

i.e binary number evenly divisible by 4. & there should not be leading 0's.

$L = \{ \underset{4}{100}, \underset{8}{1000}, \underset{(12)}{1100}, \underset{16}{10000} \text{ ---} \}$

$RE = (1\ (0+1)^*\ 00)$

21) w correspond to the binary encoding without leading 0's of natural number that are power of 4.

$L = \{ \underset{4^0=1}{1}, \underset{4^1=4}{100}, \underset{4^2=16}{10000}, \underset{4^3=64}{1000000} \text{ --} \}$

$RE = 1\,(00)^*$

22) $w \in \{0-9\}^*$ w corresponding to the decimal encoding without leading 0's of an odd natural number. i.e every odd decimal no end with 1 or 3 or 5 or 7 or 9 & begin with (1-9) & in b/w we can have anything

$RE = (1-9)(0-9)^*(1+3+5+7+9)$

(23) string doesn't contain substring 01

$RE = 1^*0^*$

(24) Starts with a but not having consecutive 6's

$RE = (a+ab)^*$

(25) $L=\{a^n b^m : (m+n) \text{ is even}\}$

either m and n should be even/m & n odd

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

{even a's & even b's} or {odd a's & odd b's}

m	n	m+n
0	0	0
1	1	2
1	3	4
3	1	4
2	2	4
1	5	6

(26) $L=\{a^n b^m \mid n \geq 4 \text{ & } m \leq 3\}$

$RE = (aaaaa a^*)(\epsilon + b + bb + bbb)$

atleast 4 a's; almost 3 b's

(27) $L=\{a^n b^n \mid 0 \leq n \leq 4\}$

$RE = \{\epsilon + ab + aabb + aaabbb + aaaabbbb\}$

(28) $L=\{a^{2n} \cdot b^{2m} \mid n >= 0 \text{ & } m >= 0\}$

$RE = (aa)^*(bb)^*$

(29) a's appear triples there is no restriction on the no of b's

$RE = (b^*(aaa)^*b^*)^*$

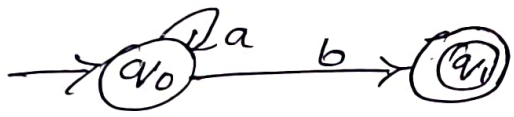
(30) $L=\{w \in \{a,b\}^* : |w| \bmod 3 = 0$

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

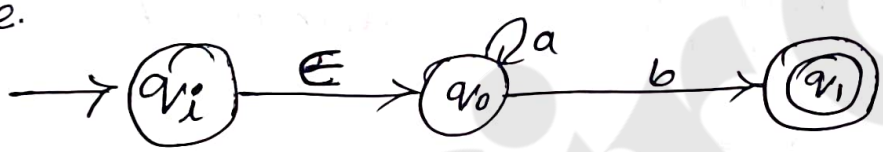
Converting FSM to Regular Expression

[State elimination method.]

Step1: Initial state should not have incoming edge. If exist create one new state and make it as the initial state

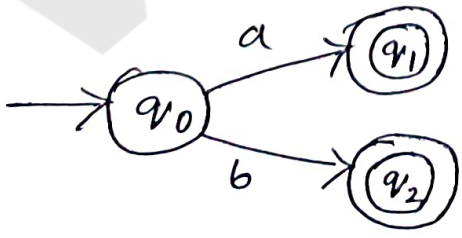
ex: 

q_0 is having incoming edge. so create new state and make it as a initial state, so that we can remove incoming edge.

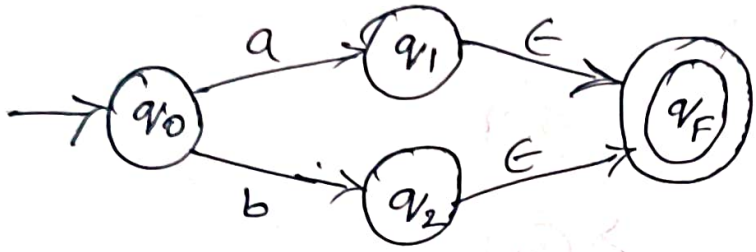


'q_i' is the new state make q_i as the initial state and show ϵ transition from q_i to q_0

Step2: Finite state machine should not have multiple final states. If exists than create one or more new state make this state as the final state and show ϵ transition from old final states to new final state



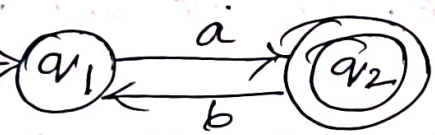
Here FSM is having 2 final states q_1 & q_2 we need to remove multiple final state



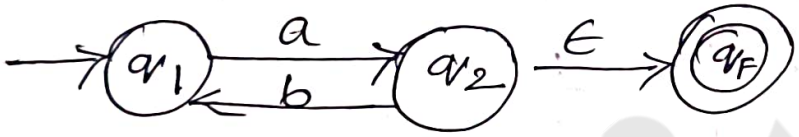
q_F is new final state.

Step3 : final state in FSM should not have outgoing edge. If exist then create new state and mark it as a final state

ex :



here q_2 is the final state having outgoing edge.

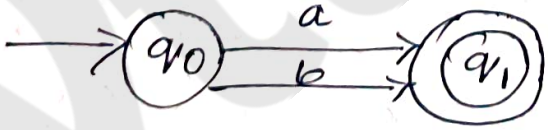


now q_F does not contains outgoing edge.

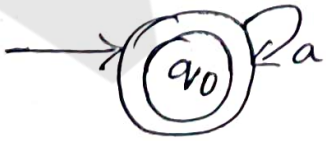
Step4 : Remove every state one after another in any order and at the same time write equivalent R.E
Finally FA should have one initial & one final state.

ex :

R.E



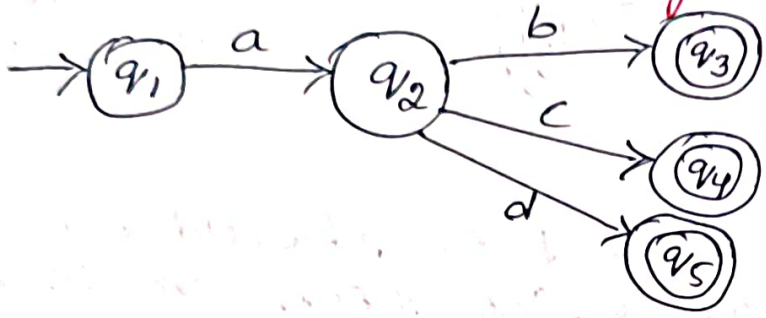
$(a+b)$



(a^*)

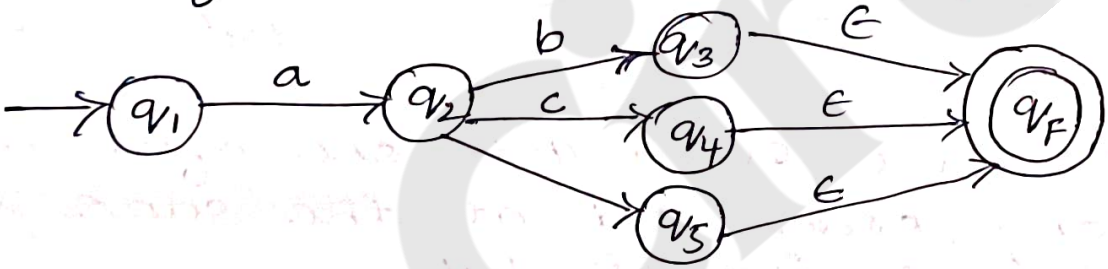
$\rightarrow (ab)$

(2) convert FSM to regular Expression.



Step1: Initial state q_1 doesnot have any Incoming edge. so no modification required.

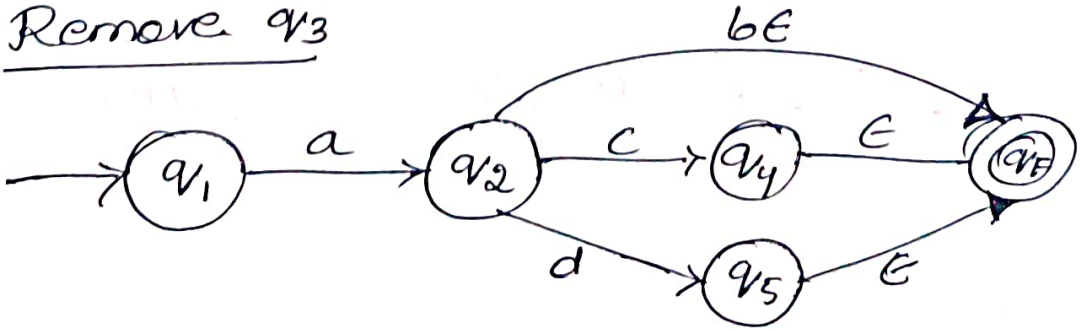
Step2: Here FSM is having 3 final state. This has to be removed. introduce new final state q_F



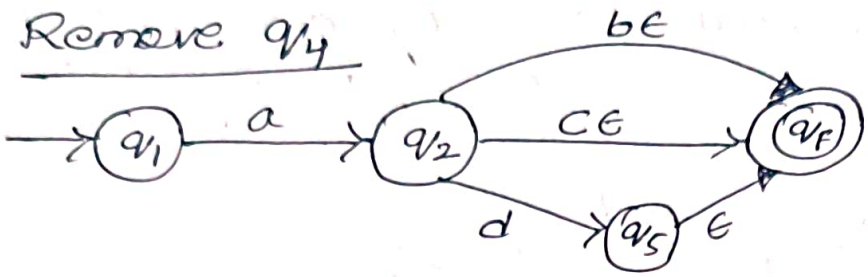
Step3: FSM should not have outgoing edge for final state. In this FSM q_F does not have any outgoing edge. so no modification is required.

Step4: Eliminate state one after another and write equivalent Regular Expression till we get one Initial and final state.

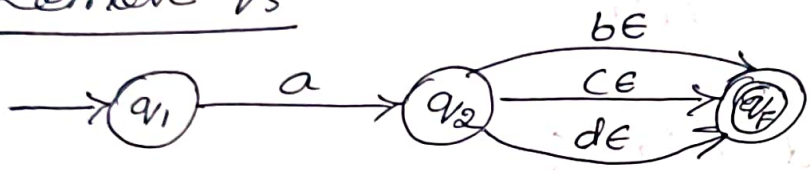
Remove q_3



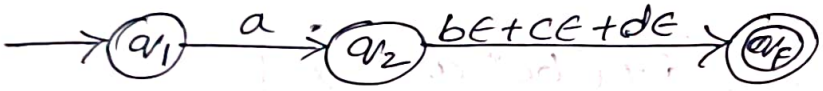
Remove q_4



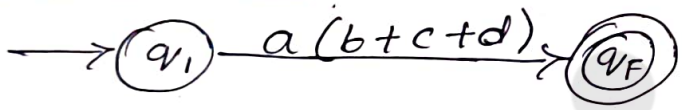
Remove q_5



or

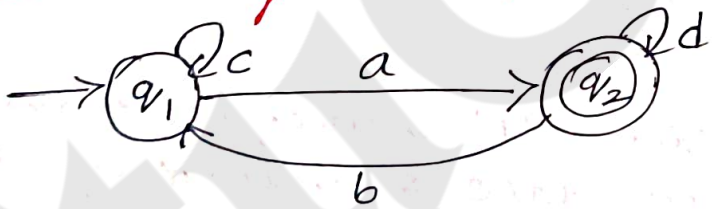


Remove q_2



$\therefore$ $\boxed{RE = a(b+c+d)}$ //

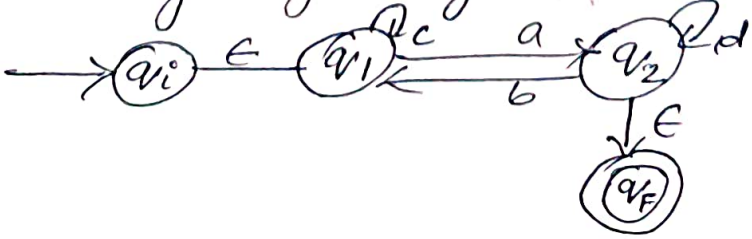
(2) obtain regular expression for the following FSM



step1 :

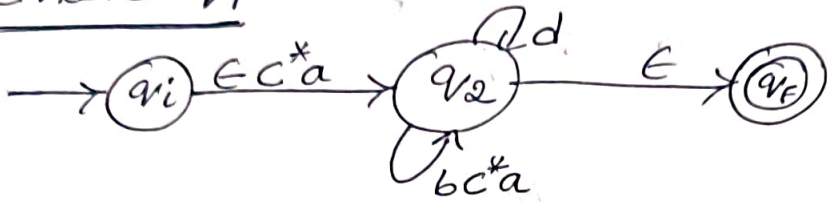
step2 : No multiple final state so no modification

step3 : outgoing edge exist for q_2

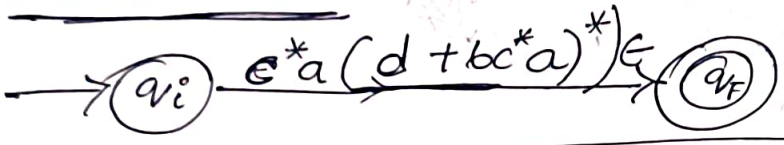


Step4 : Remove state one by one untill we get one initial & one final state

Remove q_1

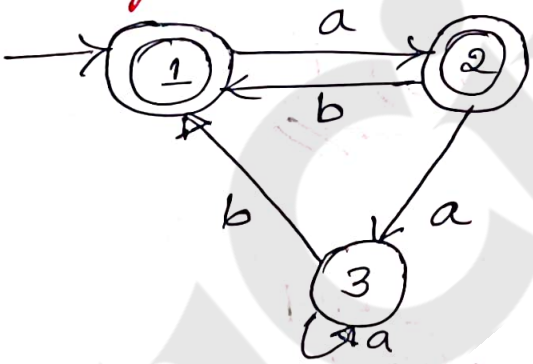


Remove q_2

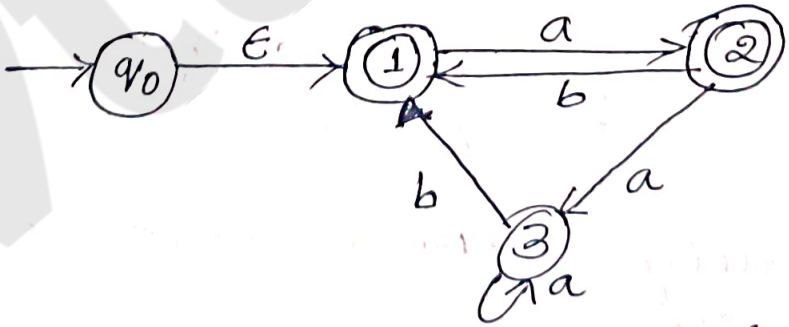


$\therefore$ $RE = c^*a(d+(bc^*a)^*)$

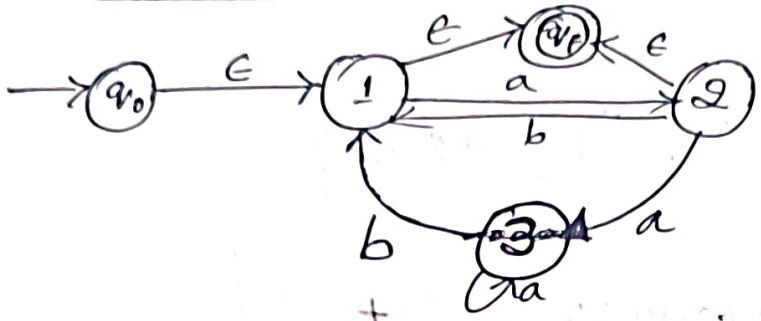
(3) Obtain regular expression for given FSM.



Step1 : Initial state (1) having incoming edge so create a new state & make it as initial state



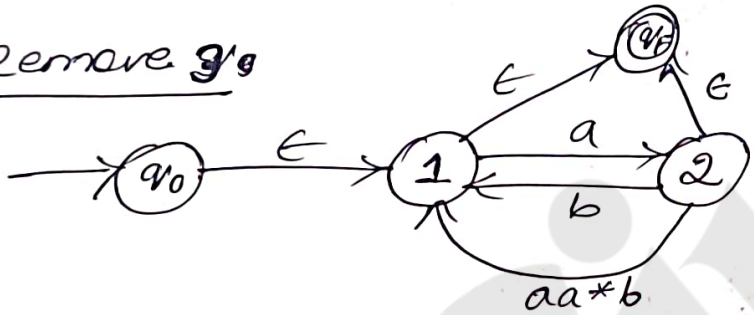
Step2 : FSM having multiple final state so create a new final state



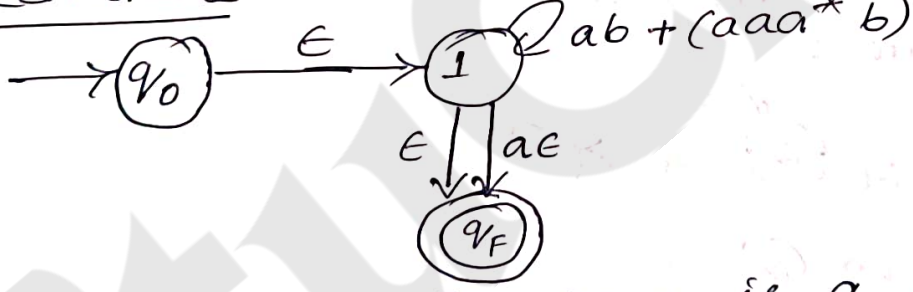
Step3 : Final state q_F is not having any outgoing edge. So no modification is required

Step4 : Remove state one by one till we get one initial & one final state.

Remove 3's



Remove 2

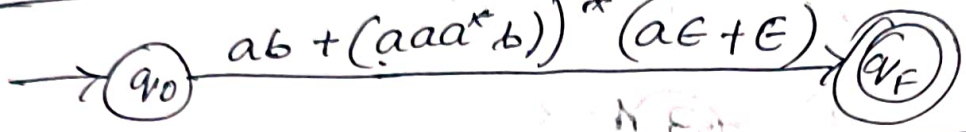


→ State (1) to (1) there is a loop 'ab' through state (2). since we are removing state (2) mention the loop 'ab' to state (1).

→ State (1) to (1) there is one more loop aaa*b through (2). so mention loop aaa*b to state (1)

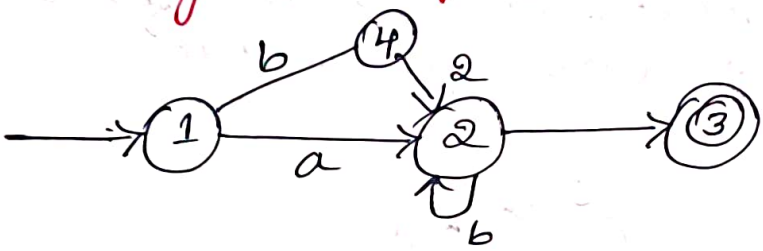
→ from (1) to (q_F) there is a transition 'aϵ' through (2). since we are removing (2) write transition 'aϵ' from (1) to (q_F).

Remove state (1)

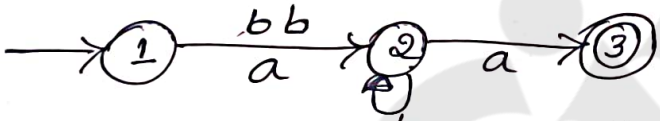


$\therefore$ **RE = $ab+(aaa^*b))^*(a+\epsilon)$**

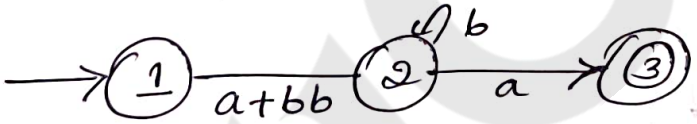
(4) Build Regular expression from FSM



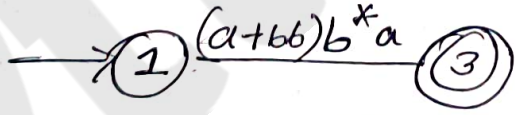
Step1:
Let replace state 4



step2: collapse multiple transition from state 1 to state 2

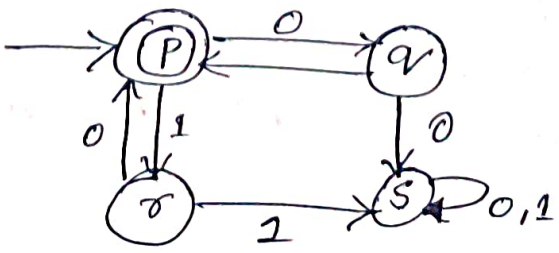


Step3 : remove (2)

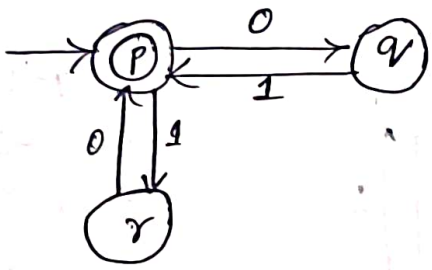


$\therefore$ **RE = $(a+bb)b^*a$**

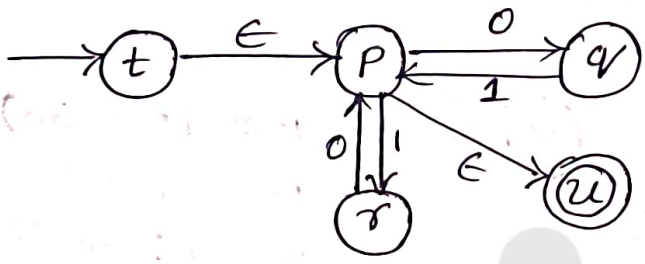
(5) Build RE from FSM



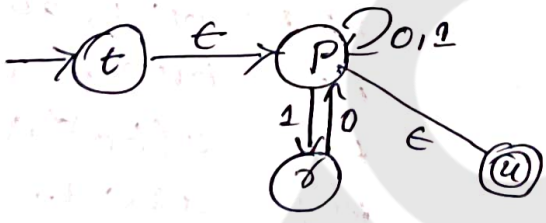
Step1: Remove state s as its dead state.



Step2: Add new state start state t & new accept state u. After adding t and u.



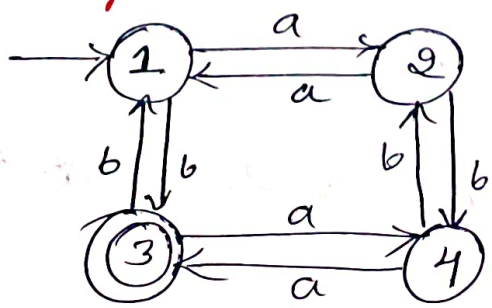
Step3: Let remove state q



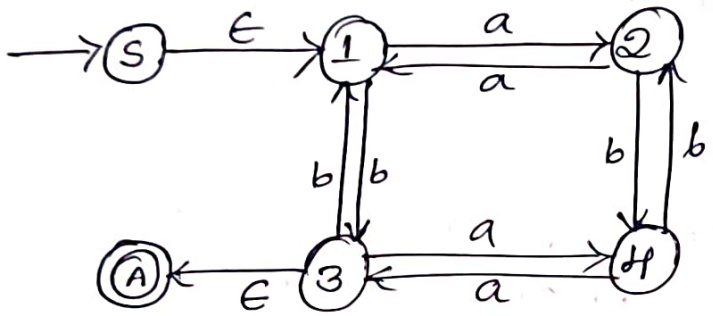
Step4: Let remove state r

$\therefore$ RE = $(01+10)^*$

6) Build Regular expression from FSM

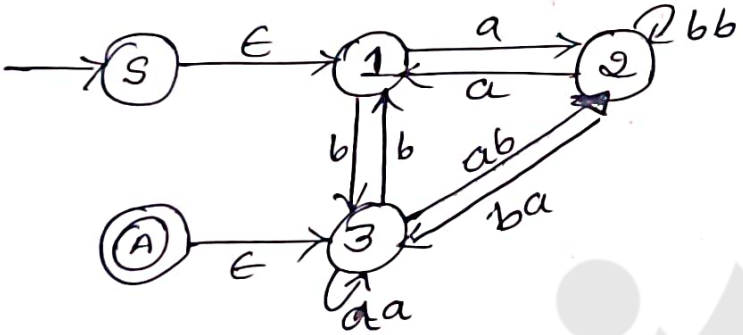


Step1: Add new start state S and new accepting state A.



2-4-2 = bb
3-4-3 = aa
2-4-3 = ba
3-4-2 = ab

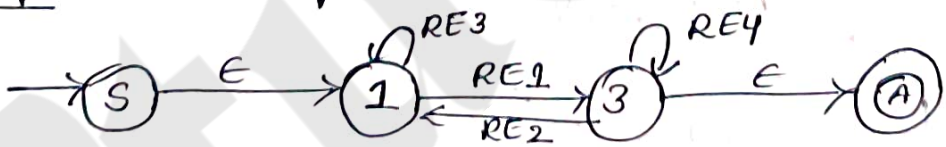
Step2: replace state 4



1-2-1 : $a(bb)^*a$
1-2-3 : $a(bb)^*ba$
3-2-1 : $ab(bb)^*a$
3-2-3 : $ab(bb)^*ba$

$RE1 = b \cup a(bb)^*a$
$RE2 = b \cup ab(bb)^*a$
$RE3 = a(bb)^*a$
$RE4 = aa \cup ab(bb)^*ba$

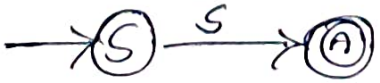
Step3: Let replace state 2



Step4: Let replace state 3

$RE5 = (RE1)(RE4)^*$
$REG = (RE3) \cup (RE1)(RE4)^* (RE2)$

Step5: ~~RE : (REG)*~~ Let replace state 1



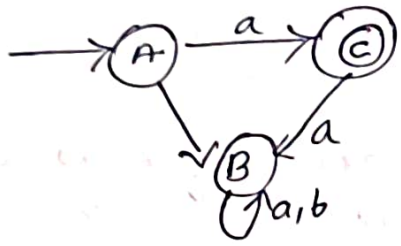
$RE = (REG)^*(RE5)$

~~$RE = a(bb)^*aa$~~

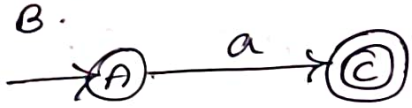
$RE = (RE3) \cup (RE1)(RE4)^* (RE2))^* ((RE1)(RE4)^*)$

substitute ↑ regular expression value.

(7) Find RE for following DFSM

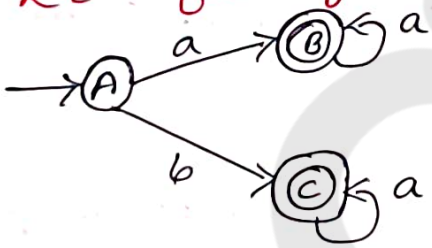


Step1 : since state B is dead state, we eliminate state B.

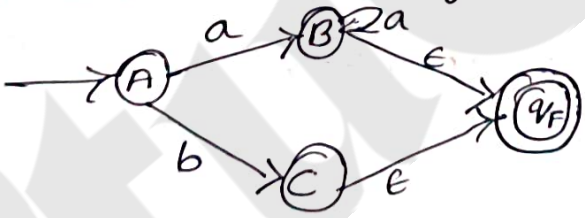


$RE = a$

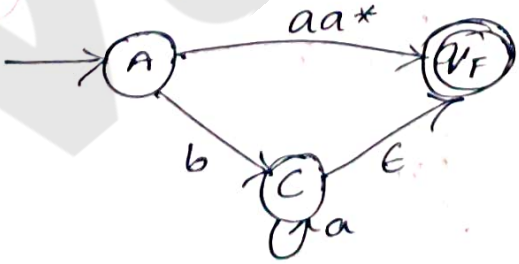
7) Find RE for given DFSM



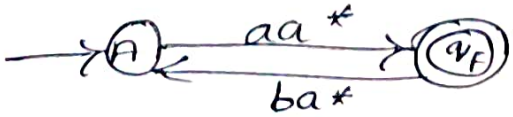
Step1 : create new final state



Step2 : Eliminate B



Step3 : Eliminate C



$RE = aa^* + ba^*$

Conversion from Regular expression to FSM
[using subset method]

OR

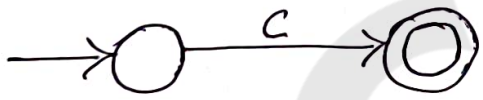
Building FSM from Regular Expression

→ Any language that can defined by regular expression can be accepted by some FSM.

→ According to Kleen's theorem 1, for every Regular expression, there is an Equivalent FSM.

(1) $\alpha = RE$ i.e $L(\alpha) = L(M)$

$\alpha = c \in \Sigma$, construct simple FSM



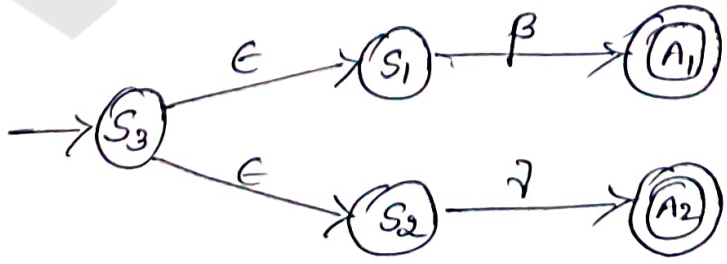
(2) If α in any ϕ, we construct simple FSM

(3) If α is ϵ, we construct simple FSM

(4) Let β and γ two Regular expression

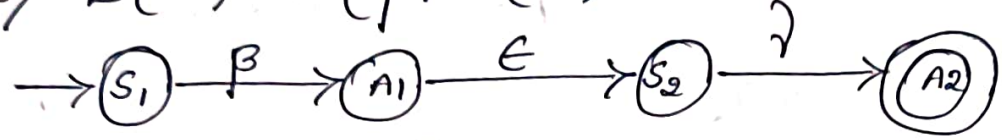
$L(M) = L(\alpha) = L(\beta) \cup L(\gamma)$

(5)



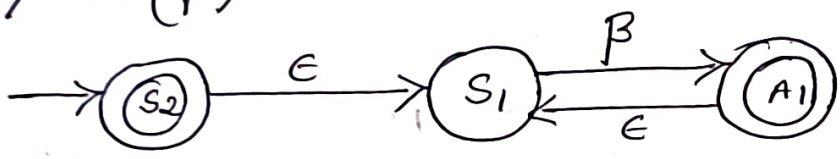
$$\alpha = \beta \cup \gamma$$

5) $L(\alpha) = L(\beta)\,L(\gamma)$



$\alpha = \beta\gamma$

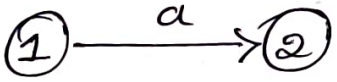
6) $\alpha(\beta)^*$



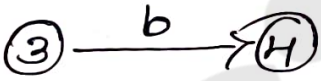
(1) obtain FSM for the given regular Expression

b(a∪b) or b(a+b)

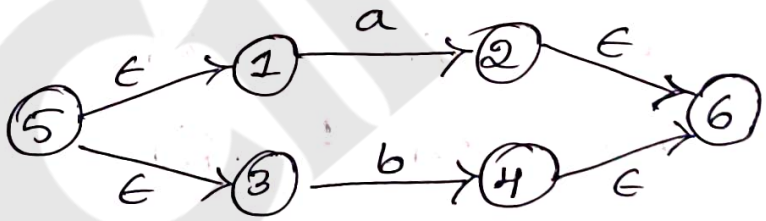
a =



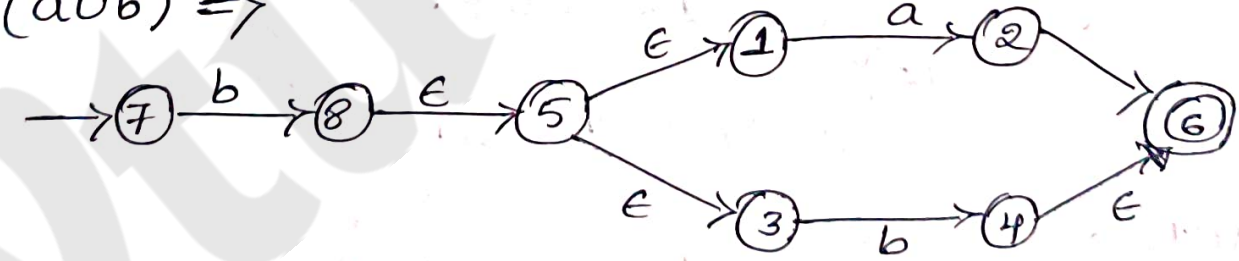
b =



a+b/a∪b =



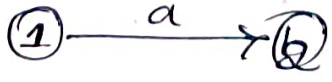
b(a∪b) ⇒



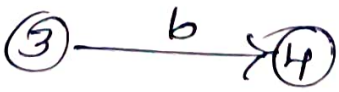
(2) obtain FSM for the given regular Expression

b(a∪b)*

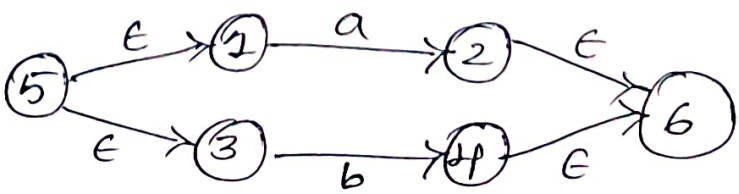
a =



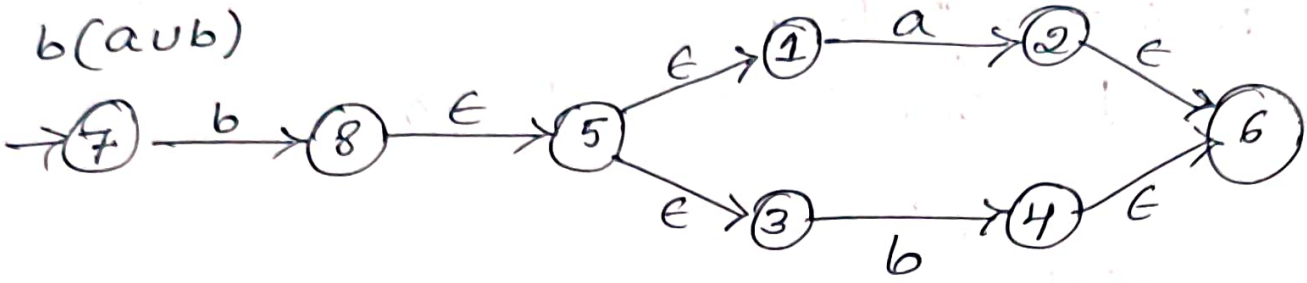
b =



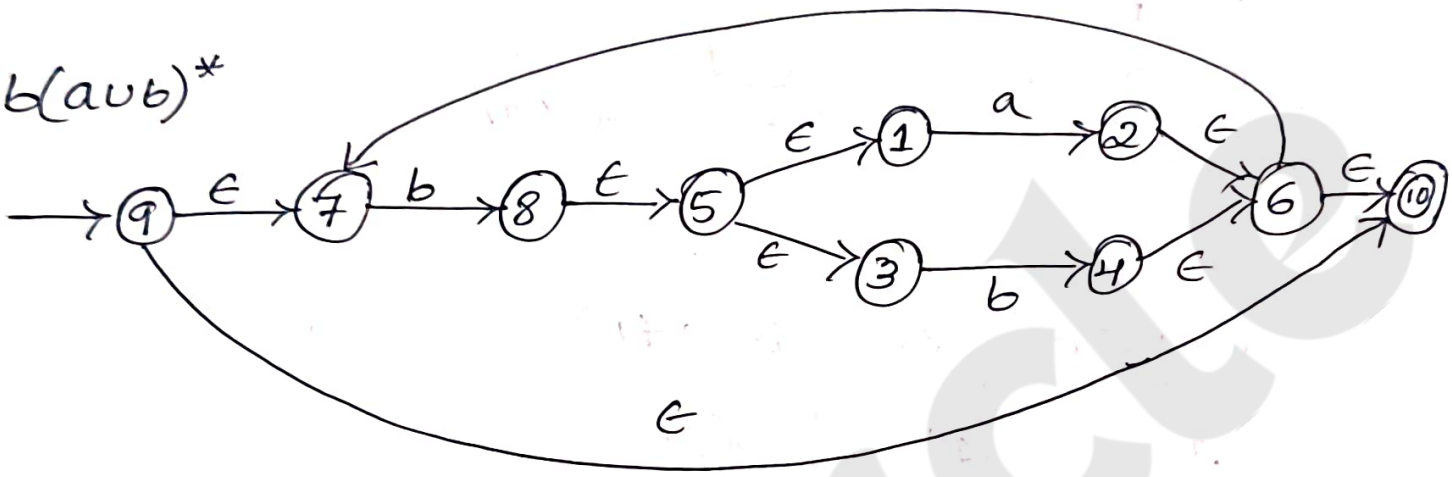
(a∪b)



b(aub)



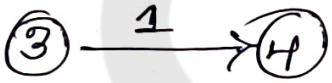
b(aub)*



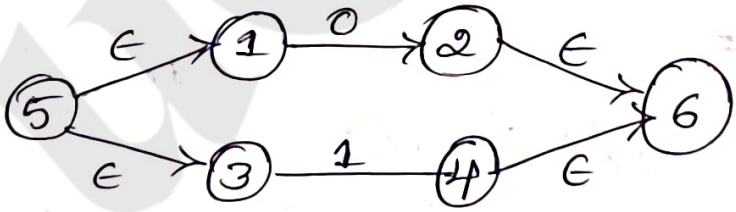
(3) obtain FSM from RE (0U1)*

0 =

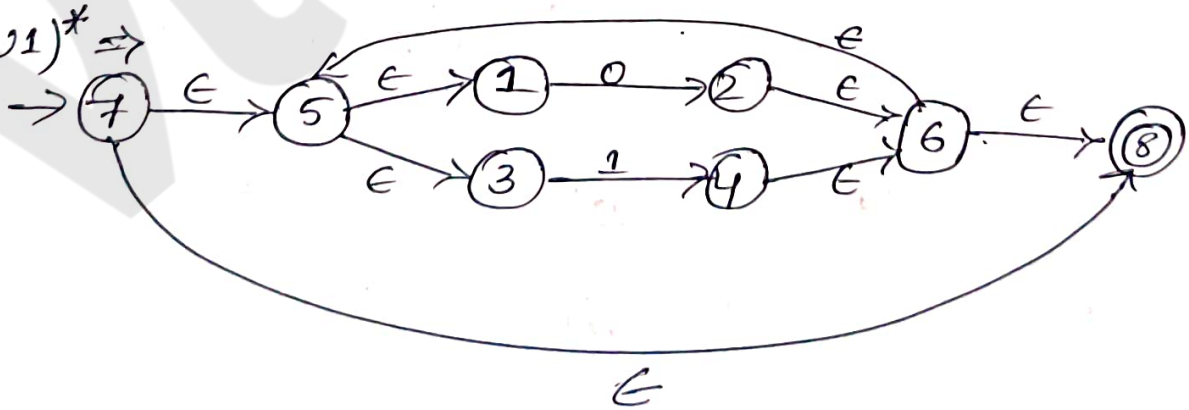
1 =



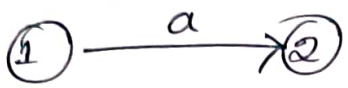
0U1 =

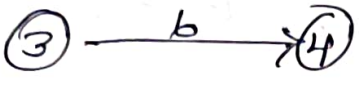


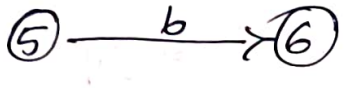
(0U1)* ⇒

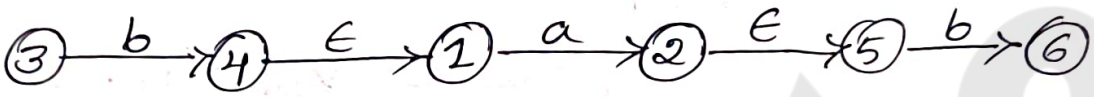


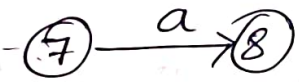
(4) Obtain FSM from R.E (babUa*)

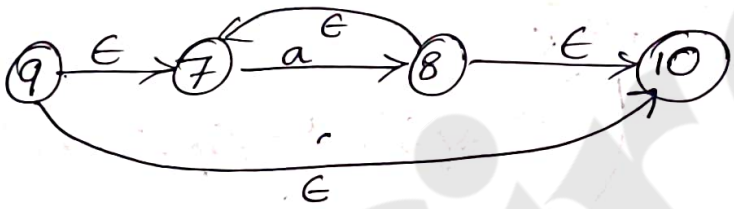
a = 

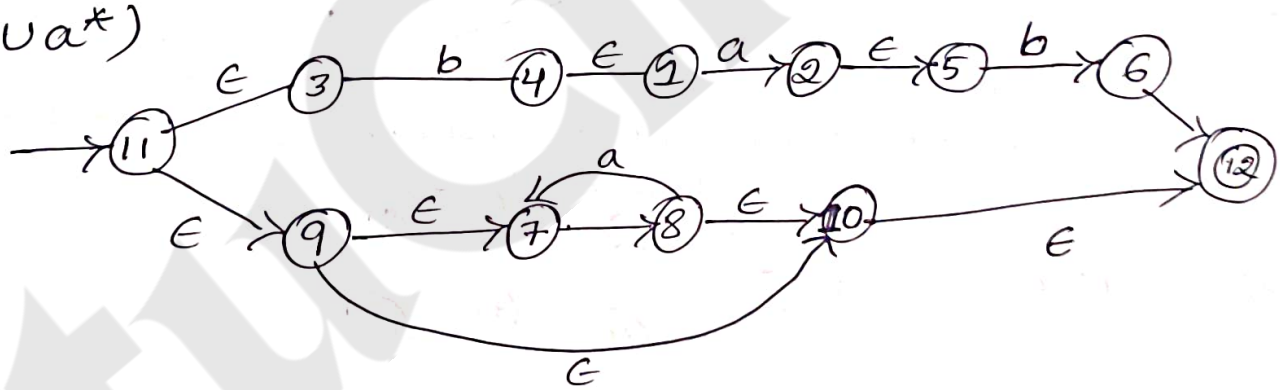
b = 

b 

bab 

a = 

a^* = 

(babUa*) 

QP question

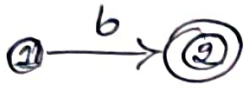
1) 10 + (0+11) 0* 1

2) (a+b)* abb

3) (a* + ab) (a+b)*

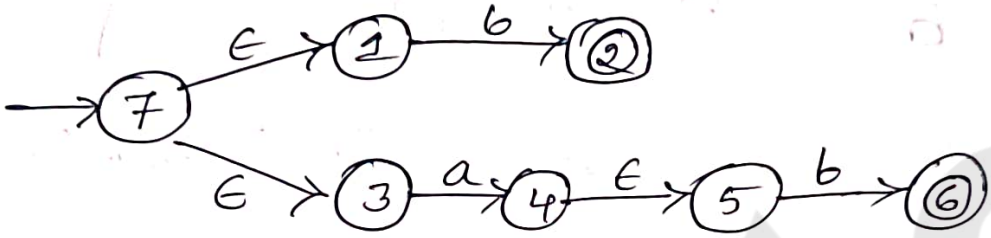
(5) construct FSM for given RE $(b \cup ab)^*$

b =

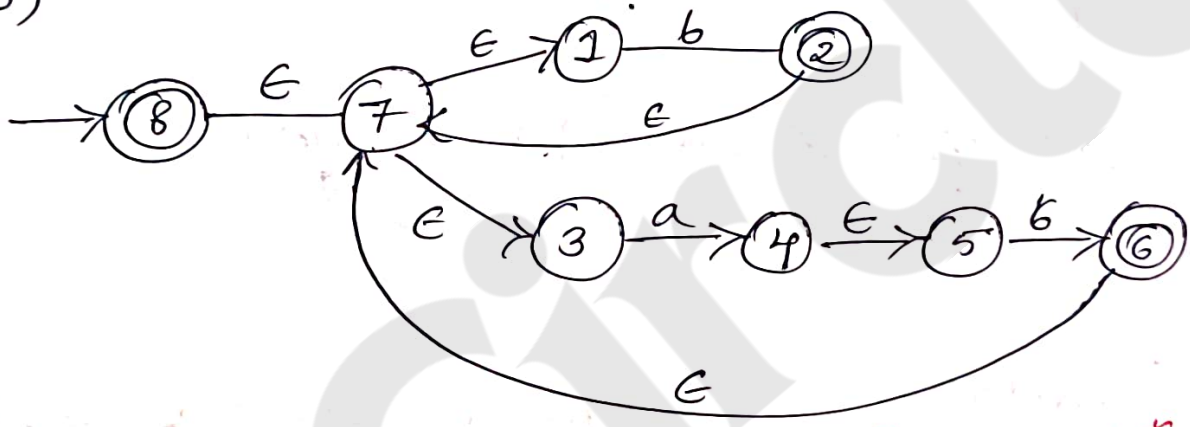


ab =

$b \cup ab$ =



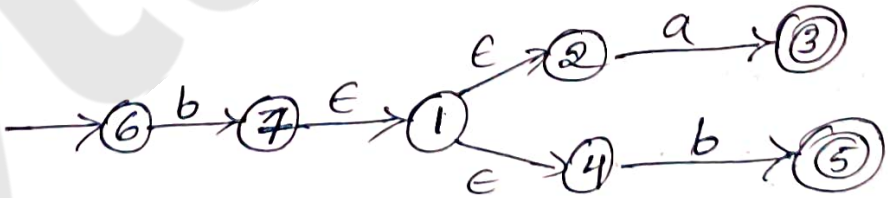
$(b \cup ab)^*$



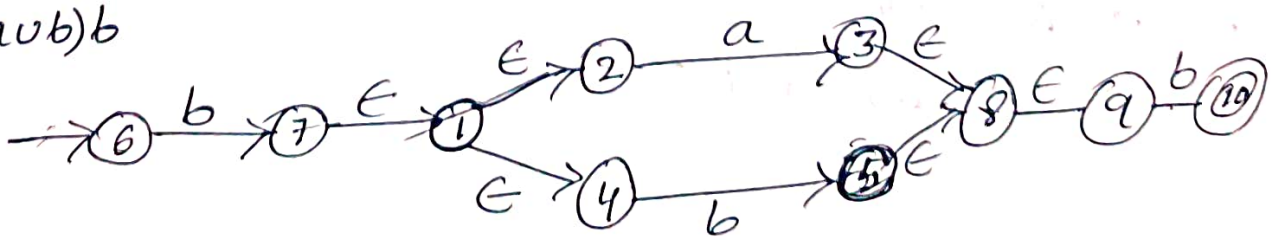
(6) construct FSM for given RE $(b(a \cup b)b)^*$

$a \cup b$

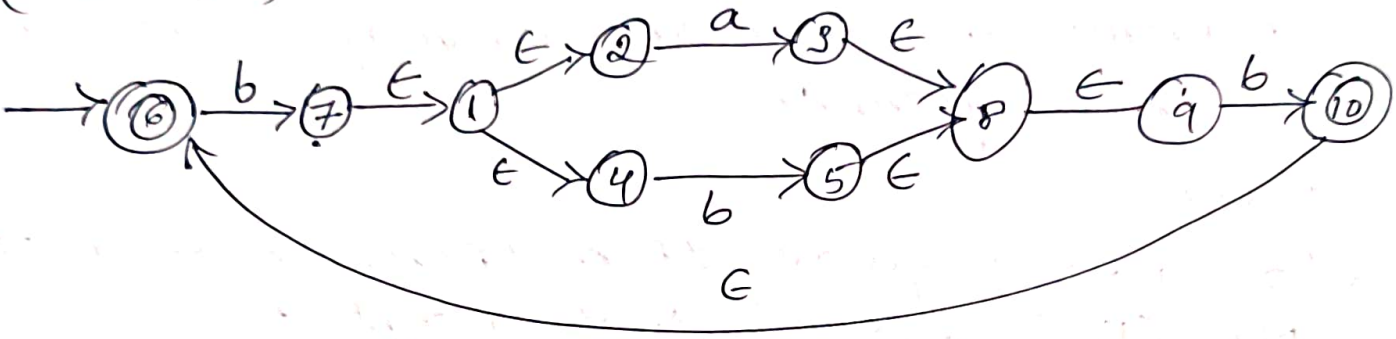
$b(a \cup b)$



$b(a \cup b)b$



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



closure property of Regular Expression

Theorem: The regular languages are closed under union, concatenation and Kleen star

Proof:
It is given that L_1 and L_2 are regular languages so there exists regular expression R_1 and R_2 such that.

$$L_1 = L(R_1) \; ; \; L_2 = L(R_2)$$

By the definition of regular expressions we have.

→ $R_1 + R_2$ is a regular expression denoting language $L_1 \cup L_2$

→ $R_1 R_2$ is a regular expression denoting the language $L_1 \cdot L_2$

→ R_1^* is a regular expression denoting the language $R_1^* \; L_1^*$

So the regular language are closed under union, concatenation and Kleen star operation.

Closure under complement

Theorem: The regular language are closed under complement

proof:

* If L_1 is regular, then there exist a DFSM $M_1 = (K, \Sigma, \delta, s, A)$ that accept it.
* The DFSM $M_2 = (K, \Sigma, \delta, s, A)$ namely M_1 with accepting and non accepting states swapped accepts $\sim(L(M_1))$ because it rejects all the strings that M_1 accepts & rejects all strings that M_1 accepts.

Closure under Intersection

Theorem: The regular language are closed under intersection

proof:

- $L(M_1) \cap L(M_2) = \sim(\sim L(M_1) \cup \sim L(M_2))$

we have already shown that all the regular language are closed under both complement and union.

- Thus they are closed under intersection

Closure under the difference

Theorem: The regular language are closed under set difference.

proof:

$$L(M_1) - L(M_2) = L(M_1) \cap \sim L(M_2)$$

- Regular language are closed under both complement and intersection
- Thus regular language are closed under set difference.

closure under Reverse

Theorem: The regular language are closed under reverse.

proof : $L^R = \{ w \in \Sigma^* \; w = x^R \text{ for some } x \in L \}$

ex Let $L = \{001, 10, 1111\}$ then $L^R = \{100, 01, 111\}$

closure under substitution

The regular languages are closed under letter substitution

- consider any two alphabet Σ_1 & Σ_2
- Let sub be a function from Σ_1 to Σ_2^*
- then let-sub is a letter substitution from L_1 to L_2 iff $let\text{-}sub(L_1) = \{ w \in \Sigma^* : \exists y \in L_1$ ($w = y$ except that every character of y has been replaced by $sub(c)$ }

** Pumping Lemma Theorem

→ Anything (Any language) that can be represented using FSM or R.E is called Regular Language

→ we use this theorem called pumping Lemma to prove the language is not regular.

Pigeonhole principle:

According to this principle long strings force Repeated States

State and prove pumping lemma theorem for regular language.

Theorem: If L is regular language then:

$\exists K >= 1$ ($\forall$ string $w \in L$, where $|w| >= K$ ($\exists x, y, z$ ($w = xyz$; $|xy| <= K$, $y \neq \epsilon$ and $\forall q >= 0$ $(xy^q z \in L)$)))

proof:

- If L is a regular then it is accepted by some DFSM $M = (K, \Sigma, \delta, s, A)$
  Let K be $|K|$
- Let w be any string in L of length K or greater
- By theorem to accept w, M must traverse some loop atleast once.
- we can carve w up and assign the name y to the first substring to drive M through a loop.
- Then x is the part of w that preceeds y & z is the part of w that follows y.
- we show that each of the last three condition must then hold:

1. $|xy| <= K$
   M must not traverse through a loop It can read K-1 character without revisiting any states. But K^{th} character will take M to a state visited before.
2. $y \neq \epsilon$
   since M is deterministic, there are no loops traversed by ϵ
3. $\forall q >= 0$ $(xy^q z \in L)$
   y can be pumped out once and the resulting string must be in L.

Steps to prove language is not regular by contradiction Method.

1. Assume L is regular
2. Apply pumping theorem for the given language.
3. choose a string w, where $w \in L$ and $|w| \geq K$
4. Split w into xyz such that $|xy| \leq K$ & $y \neq \epsilon$
5. choose a value for q such that xy^qz is not in L.
6. our assumption is wrong and hence the given language is not regular.

problems on pumping lemma theorem

[showing that the language is not regular]

1) $A^nB^n = \{ a^nb^n : n \geq 0 \}$ is not regular

Assume L is regular

$w = a^Kb^K$ $\quad |w| = K+K = 2K$ $\quad |w| \geq K$

w is in language

w should satisfy condition of pumping lemma there must exist some x,y,z

$w = x,y,z$ such that

$|xy| \leq K \quad y \neq \epsilon \quad \forall\ q \geq 0 \quad xy^qz \in L$

$w = xyz$

Since $|xy| \leq K$ y must occur within first K character

$w = \underbrace{a \text{----} a}_{xy}\ \underbrace{b \text{----} b}_{z}$

$= a \cdots a \mid a \; b \cdots b$

$x^{K-1} \quad y \quad z$

$= a^{K-1} \; a \; b^{K}$

$x \quad y \quad z$

According to pumping lemma $\forall q \geq 0$

$xy^q z \in L$

$= a^{K-1} (a)^q b^K$

Assume $q = 2$

$= a^{K-1} a^2 b^K$

$= a^{K-1+2} b^K$

$= a^{K+1} b^K \notin L$

Since w has more a's than b's

∴ Hence the language is not regular.

(2) Show that language $L = \{ww^r ; w \in \{a,b\}^*\}$ is not regular

If $w = abab$

$w^r = baba$

Let $v = ababbaba$ is a string $\in L$

$|v| = 8$ Let $K = 8$

$\therefore |v| \geq K$

divide v into 3 parts xyz such that $|xy| \leq K$ & $y \neq \epsilon$ and for $q \geq 0$ $xy^q z \in L$

$v = \underbrace{ab}_{xy} \underbrace{abbaba}_{z}$

$|xy| \leq 8$ i.e $|ab| \leq 8$ i.e $2 \leq 8$ — 1st condition holds

$y \neq \epsilon$ because $y = b$ [second condition holds]

for $q \geq 0$ $xy^q z \in L$

for $q = 0$ $xy^q z = xy^0 z$

$= ab^0 abbaba$

$= aabbaba \notin L$

$\therefore$ Hence given language is not regular

(3) The prime number of a is not regular

Let L = prime $a = \{a^n ; n$ is prime$\}$

Let $w = a^j$ assume j is smallest prime number of length $> k+1$ $|w| \geq k$

$|xy| \leq k$, $y \neq \epsilon$, $xy^q z \in L$ $\forall$ $q \geq 0$

$w = a^{|x|+|z|+q|y|}$ must $\in$ L

$|x|+|z|+q|y|$ must be prime

$q = |x| + |z|$ $q \geq 0$

$= |x| + |z| + |x| + |z| * |y|$

$= |x| + |z| (1 + |y|)$ — composite

which is a composite (non prime) when both factors are > 1

$|x| + |z| > 1$ $\because$ $|w| = k+1$ $|y| \leq k$

$1 + |y| > 1$

Hence the result string is not in L

$\therefore$ Hence given language is not regular.

4) The language with more a's & b's is not regular

$L = \{ a^n b^m : n > m \}$

$w = a^{K+1} b^K \qquad |w| = K+1+K = 2K+1$

$|w| \geq K$ ✓

$|xy| \leq K$, $y \neq \epsilon$ such that $xy^q z \in L \; q \geq 0$

$w = a^{K-1} a \; ab^K$

$\quad\; x \; y \; z$

$= a^{K-1} a^q ab^K$

Here if you pump up (in) number of a's will still be greater number of b's

Here we will pump out $\forall \; q \geq 0 \; xy^q z \in L$

$q = 0 \qquad = a^{K-1} a^0 ab^K$

$= a^{K-1} ab^K$

$= a^K b^K \notin L$

∴ Hence L is not regular.

5) ~~The~~ Even palindrome is not regular

Let $L_{\text{paleven}} = \{ ww^r : w \in \{0,1\}^* \}$

$w = 0^K 1^K | 1^K 0^K \qquad |w| = K+K+K+K = 4K$

$\therefore |w| \geq K$

$|xy| \leq K$, $y \neq \epsilon$, such that $xy^q z \in L, q \geq 0$

Since $|xy| \leq K$ y must appear within first K character.

Let $q=2$

$$w = \underbrace{0^{K-1}}_{x}\ \underbrace{0}_{y}\ 1^K 1^K 0^K$$

$$= 0^{K-1}\ 0^2\ 1^K 1^K 0^K$$

$$= 0^{K+1}\ 1^K 1^K 0^K$$

First half of the has more length than second half.

$\therefore$ Hence it is not palindrome $\notin L$

$\therefore$ Hence it is not regular.

⑥ prove that $L = \{a^{2n} \mid n \geq 1\}$ is not regular

$L = \{aa, aaaa, aaaaaa \cdots\}$

Assume $w = aaaa$

$|w| = 4$ let $K = 4$

$|w| \geq K$

$|aaaa| \geq 4$

$4 \geq 4$

we can divide w into 3 parts

$$w = \underbrace{a}_{x}\ \underbrace{a}_{y}\ \underbrace{aa}_{z}$$

I $|xy| \leq K$ i.e $2 \leq 4$ which is true

II $y \neq \epsilon$ i.e $y = a$ which is true

III $q \geq 0$ $xy^qz \in L$

let $q=0$ $xy^0z = aa^0aa$

$= aaa \notin L$

$\therefore$ Hence given language is not regular

(7) prove that $L = \{a^i b^j \mid i^0 \leq j\}$ is not regular? (12)

$L = \{aabb, abb, aabbb, abbbbb\}$

Assume L is Regular

Let $w = aabbb$

$|w| = 5$ Let $K = 5$

$|w| \leq K$

divide w into 3 parts xyz.

$w = \underset{x}{a}\ \underset{y}{a}\ \underset{z}{bbb}$

I $|xy| \leq K$ i.e $|aa| \leq 5$ i.e $2 \leq 5$ true

II $y \neq \in$ here $y = a$ which is true

III for $q \geq 0$ $xy^q z \in L$

take $q = 0$ $xy^0 z = aa^0 bbb \in L$

$q = 1$ $xy^1 z = aa^1 bbb \in L$

$q = 2$ $xy^2 z = aa^2 bbb \in L$

$q = 3$ $xy^3 z = aa^3 bbb$

$= aaaabbb \notin L$

∴ Hence given language is not regular