Madhusudan G
Dept of CSE

Regular Expressions

Regular expressions are used for representing certain set of strings in an algebraic function.

Rules of Regular Expression :-

i) Any terminal symbols i.e., Symbols ϕ, ϵ & Σ are regular expression.

ii) The union of two regular expressions is also a regular expression. i.e., R_1 & $R_2 \Rightarrow (R_1 + R_2)$

iii) The concatenation of two regular expressions is also a regular expression R_1 & $R_2 \Rightarrow R_1 \cdot R_2$

iv) The iteration (or a closure) of a regular expression is also a regular expression. $R \rightarrow R^*$

(v) The regular expression over Σ are precisely those obtained recursively by the application of the above rules once or several times.

Identities of Regular expression

i) $\phi + R = R$

ii) $\phi R + R\phi = \phi$

iii) $\epsilon R = R\epsilon = R$ iv) $\epsilon^* = \epsilon$ and $\phi^* = \lambda$

v) $R + R = R$ vi) $R^* R^* = R^*$

vii) $R \cdot R^* = R^* R$ viii) $(R^*)^* = R^*$

ix) $\epsilon + RR^* = \epsilon + R^* R = R^*$ x) $(pq)^* p = p(qp)^*$

xi) $(p+q)^* = (p^* \cdot q^*)^* = (p^* + q^*)^*$ xii) $(p+q)R = pR + qR$

$R(p+q) = Rp + Rq$

Examples of Regular expressions

i) $\{0, 1, 2\}$: 0 or 1 or 2

$R = 0 + 1 + 2$.

ii) $\{\wedge, ab\}$

$R = \wedge . ab$

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

$R = abb + a + b + bba$

iv) $\{\wedge, 0, 00, 000, \ldots\}$

$R = 0^*$

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

$R = 1^+$

A language is said to be a regular language if and only if some finite state machine recognizes it.

So what languages are not regular?

The language i) which are not recognized by any FSM. ii) which require memory (Memory of FSM is very limited, it cannot store or count strings)

eg:- i) ababb aba bb

ii) $a^n b^n$

aaa bbb

Examples of regular expression

i) $(a+b)^*$ — Set of strings of a's & b's of any length including the NULL string.

ii) $(a+b)^* abb$ — set of strings of a's & b's ending with the string abb.

iii) $ab(a+b)^*$ — set of strings of a's & b's starting with the string ab.

iv) $(a+b)^* aa (a+b)$ — set of strings of a b's having a substring aa.

v) $a^* b^* c^*$ — set of string consisting of any number of a's (may be empty string also) followed by any number of b's (may be empty string also) followed by any number of c's (may be empty string also)

1) Design a regular expression for the following cases over the alphabet $\Sigma = \{a, b\}$

i) the string length is exactly two

ii) The string length is atleast two

iii) The string length is atmost two.

i) $L_1 = \{aa, ab, ba, bb\}$

R.E: $aa + ab + ba + bb$

$a(a+b) + b(a+b)$

$(a+b)(a+b)$

$L_1 = \{ aa, ab, ba, bb, aaa \text{———} \}$

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

iii) $L_1 = \{ \epsilon, a, b, aa, ba, ab, bb \}$

$\{ \epsilon, a, b, aa, ab, ba, bb \}$

$RE = \epsilon + a + b + aa + ab + bb + bb$

$\epsilon + a + b + (a+b)(a+b)$

$(a+b+\epsilon)(a+b+\epsilon)$

2) Design a regular expression over the alphabet $\Sigma = \{a, b\}$

i) Even length string.

ii) odd length string

$L_1 = \{ \epsilon, aa, ab, ba, bb, \text{- - -} \}$

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

$L_2 = \{ \epsilon, a, b, aaa, bbb \text{— —} \}$

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

3) 18 Design a regular expression over $\Sigma = (a, b)$

i) divisible by 3

ii) 2 mod 3

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

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

4) Design a regular expression over $\Sigma = (a, b)$

i) a's exactly 2 $= b^*ab^*ab^*$

ii) a's atleast 2 $= b^*ab^*a(a+b)^*$

iii) a's atmost 2 $= b^*(\epsilon+a)b^*(\epsilon+a)b^*$

iv) a's are even $= (b^*ab^*ab^*)^* + b^*$

v) Starts with a $= a(a+b)^*$

vi) ends with a $= (a+b)^*a$

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

viii) starting & ending with different symbol

ix) Starting & ending with same symbol $= a(a+b)^*a + b(a+b)^*b + a + b$

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

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

x) no 2 a's should come together : $(b+ab)^*$

i) Write a regular expression for strings that either start with 01 or end with 01.

ii) Write a R.E that have at least two consecutive 0's or 1's.

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

ii) $R.E = (0+1)^* 00 (0+1)^* + (0+1)^* 11 (0+1)^*$

6) Write a regular expression for the set of strings whose third symbol from the right end is 0.

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

7) Write the R.E over $\Sigma = (a, b)$ for the set of strings with even number of a's followed by odd nos of b's

$$L = \{ a^{2n} b^{2n+1} \mid n \geq 0, m \geq 0 \}$$

$$R.E = (aa)^* (bb)^* b$$

8) Write the R.E for the language

$L = \{ w \in \{0,1\}^* : w$ has no pair of consecutive zeros$\}$

$$R.E = (1 + 01)^* (0 + \epsilon)$$

20

Write the R.E for the language $L = \{ a^n b^{n+1} \mid n \leq 3 \}$

$$R.E = a^4 a^* (\epsilon + b + b^2 + b^3)$$

Write the R.E for the language $L = \{ w \mid |w| \bmod 3 = 0 \}$, $w \in \{a, b\}$

$$R.E = ((a+b)^3)^*$$

Write the R.E for the language $L = \{ w \in (a,b)^*, n_a(w) \bmod 3 = 0 \}$

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

Write the R.E for the language of all strings in which every 0 is followed by immediately by 11.

$$R.E = (1 + 011)^*$$

Prove that $(1+00^*1) + (1+00^*1)(0+10^*1)(0+10^*1)$ is equal to $0^*1(0+10^*1)^*$

$$= (1+00^*1) + (1+00^*1)(0+10^*1)^*(0+10^*1)$$

$$= (1+00^*1)\left[\epsilon + (0+10^*1)^*(0+10^*1)\right]$$

$$= (1+00^*1)(0+10^*1)^* \quad \left| \; \epsilon + R^*R = R^* \right.$$

$$= (\epsilon \cdot 1 + 00^*1)(0+10^*1)^* \quad \left| \; \epsilon \cdot R = R \right.$$

$= (\epsilon + 00^*)1\,(0 + 10^*1)^*$

$= 0^*1\,(0 + 10^*1)^*$

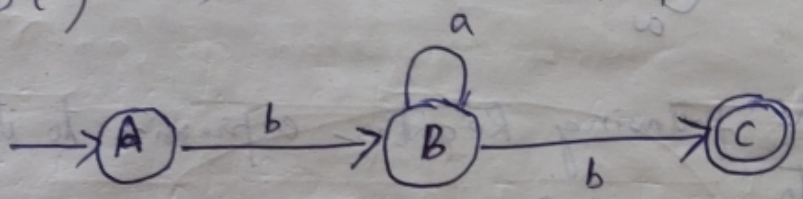
Conversion of Regular Expression to Finite Automata

i) ba^*b

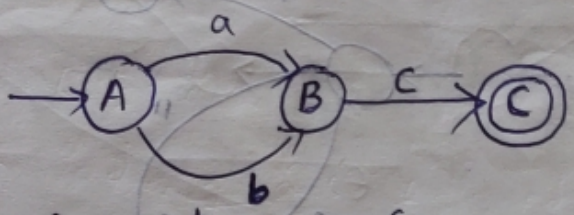
ii) $(a+b)c$

iii) $a(bc)^*$

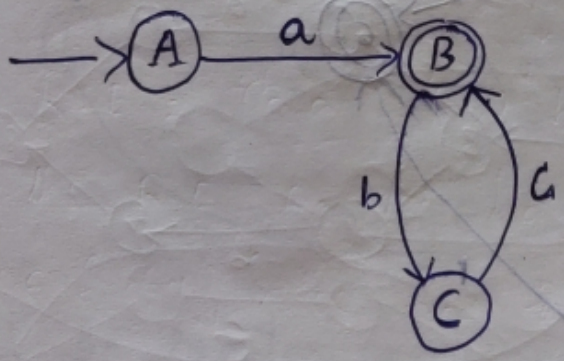
i) $R = ba^*b$: $L_1 = \{bab, baaab \text{---} \}$



ii) $(a+b)c$: $L_1 = \{ac, bc\}$

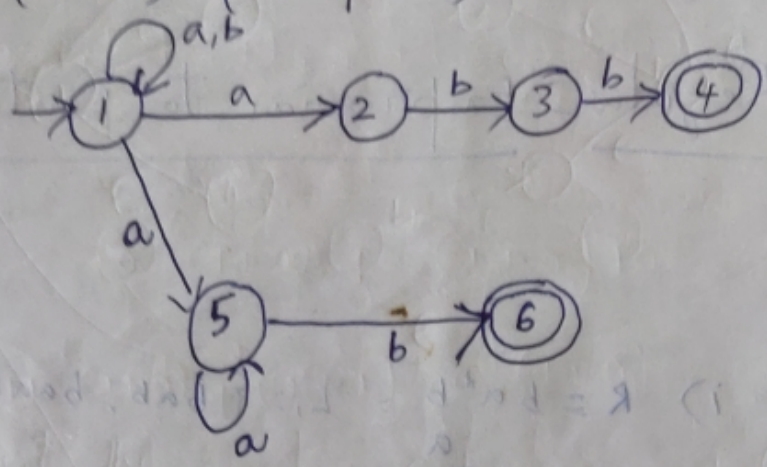


iii) $a(bc)^*$: $L_1 = \{a, abc, abcbc, abcbcbc \}$



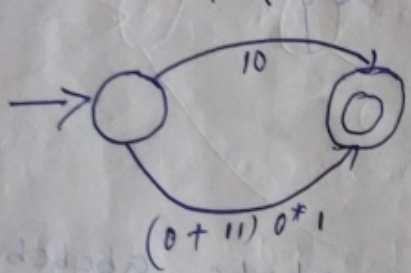
2) Convert the following expression to its Equivalent finite

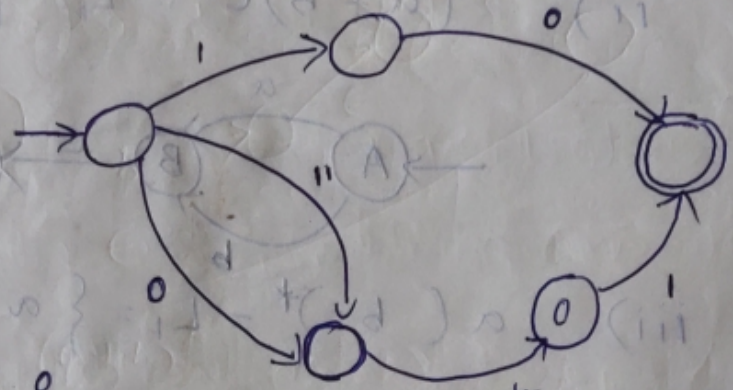
i) $(a|b)^*(abb|a+b)$

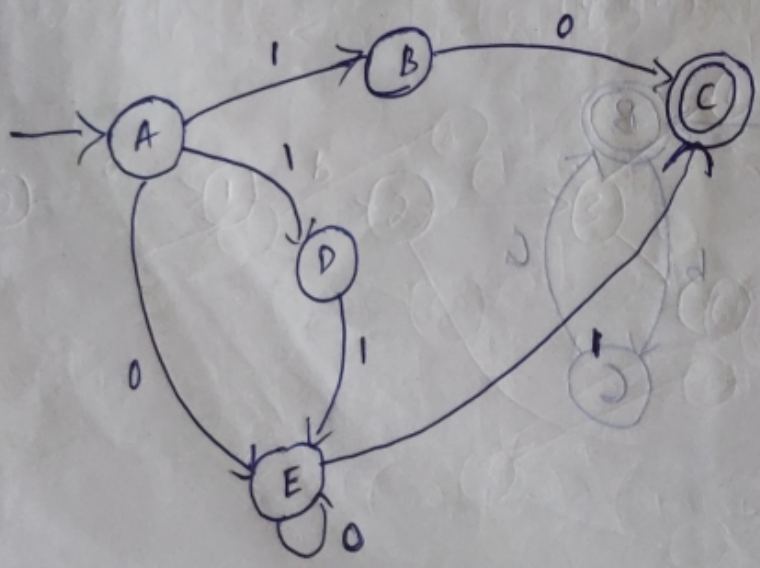


3) Convert the following Regular expression to its Equivalent finite Automata

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

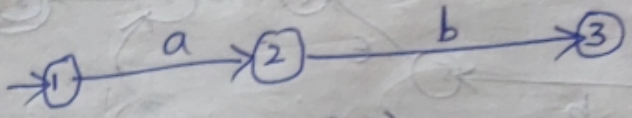






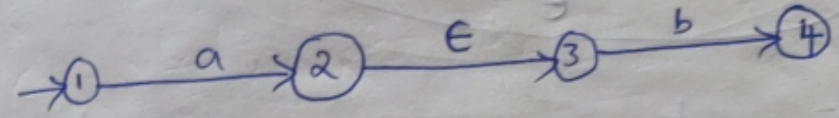
...ersion of Regular expression to ϵ-NFA using Thompson Construction method

ab

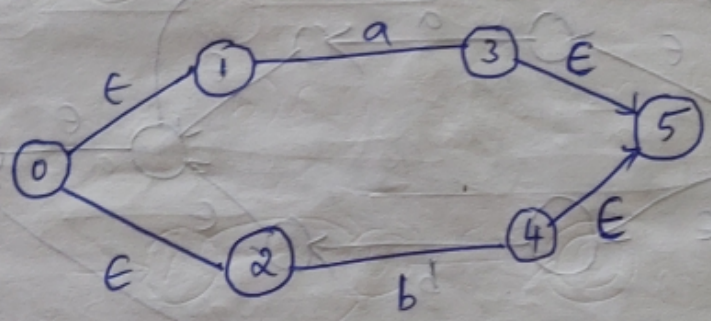


(OR)

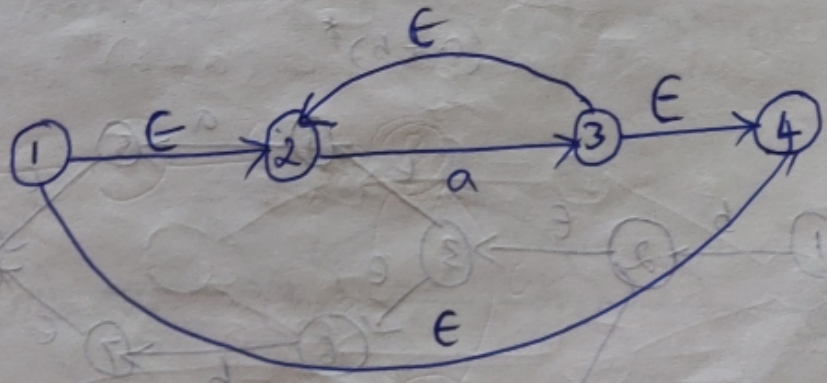
ab



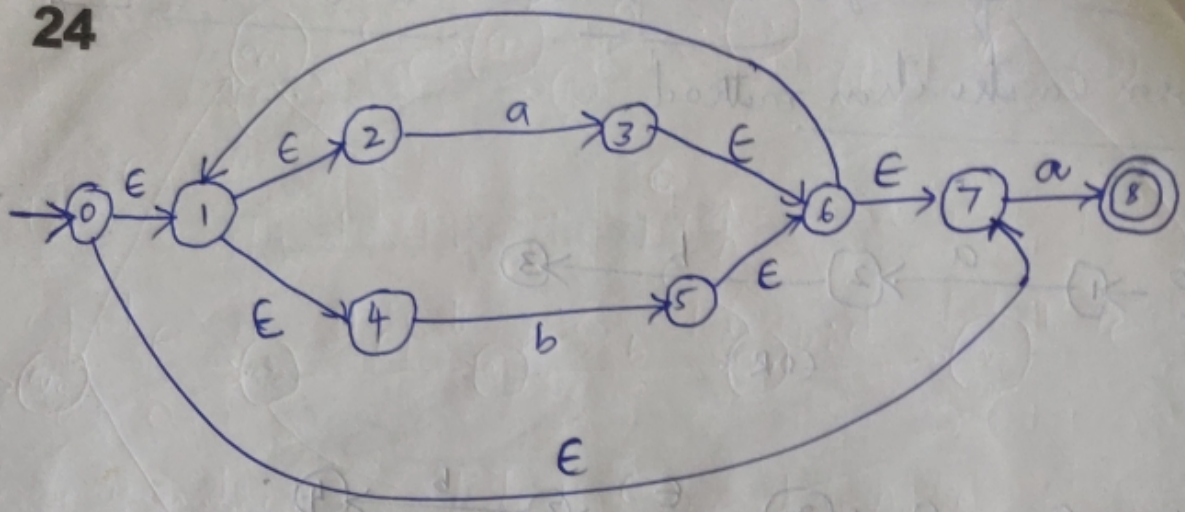
a+b



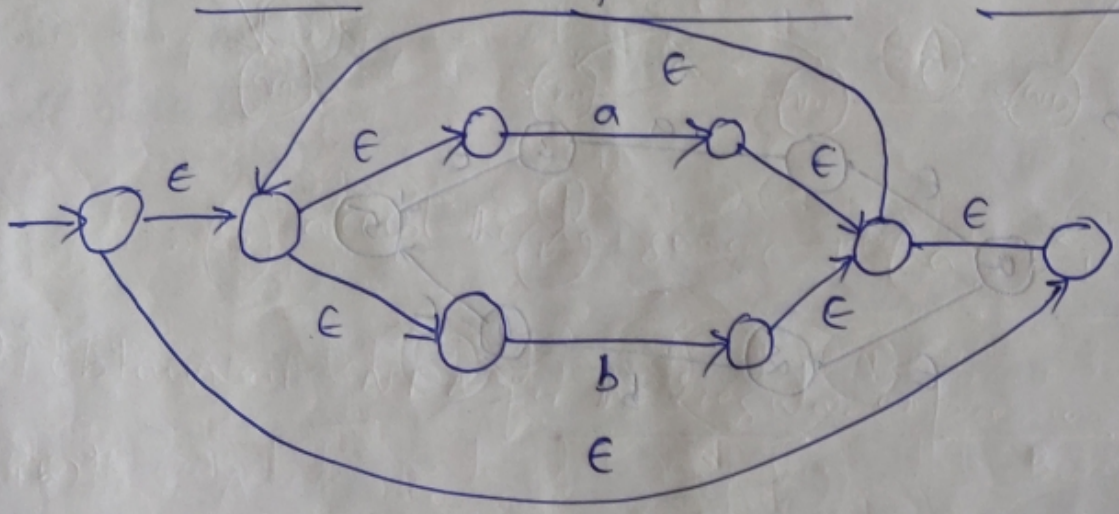
a*



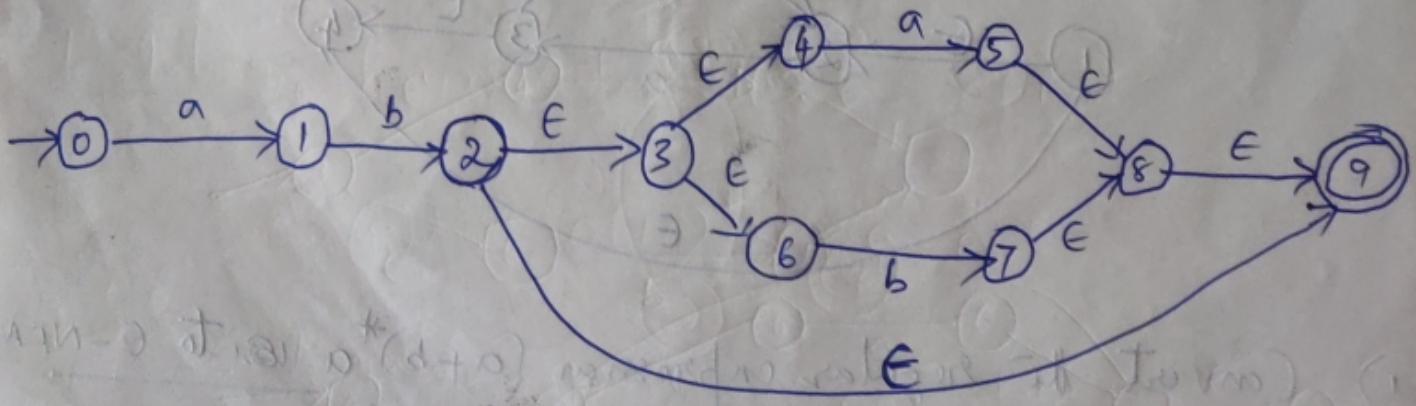
1) Convert the regular expression $(a+b)^*a$ to ϵ-NFA



2) Convert the regular expression ab (a+b)* to ε-NFA

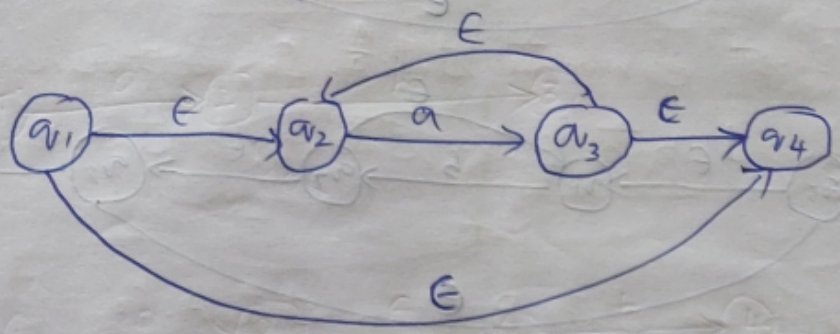


(a+b)*

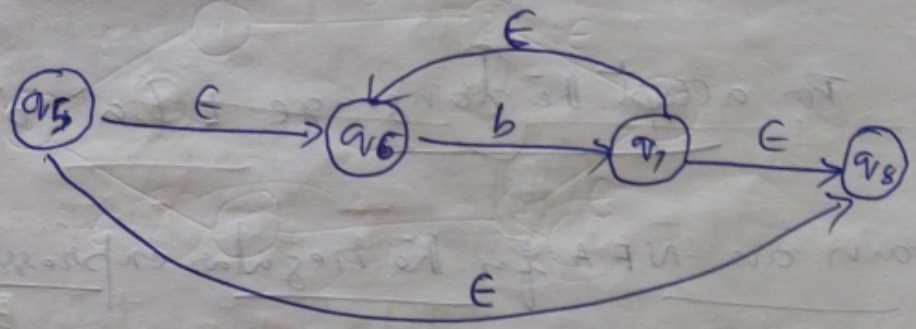


Obtain an NFA for the regular expression $a^*+b^*+c^*$

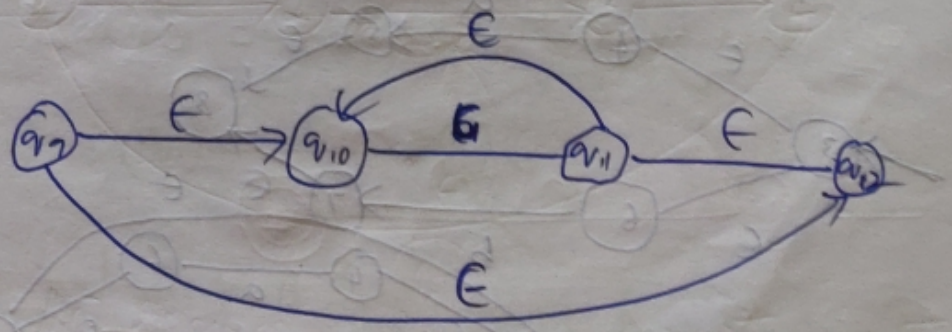
The m/c corresponding to the regular expression a^* can be

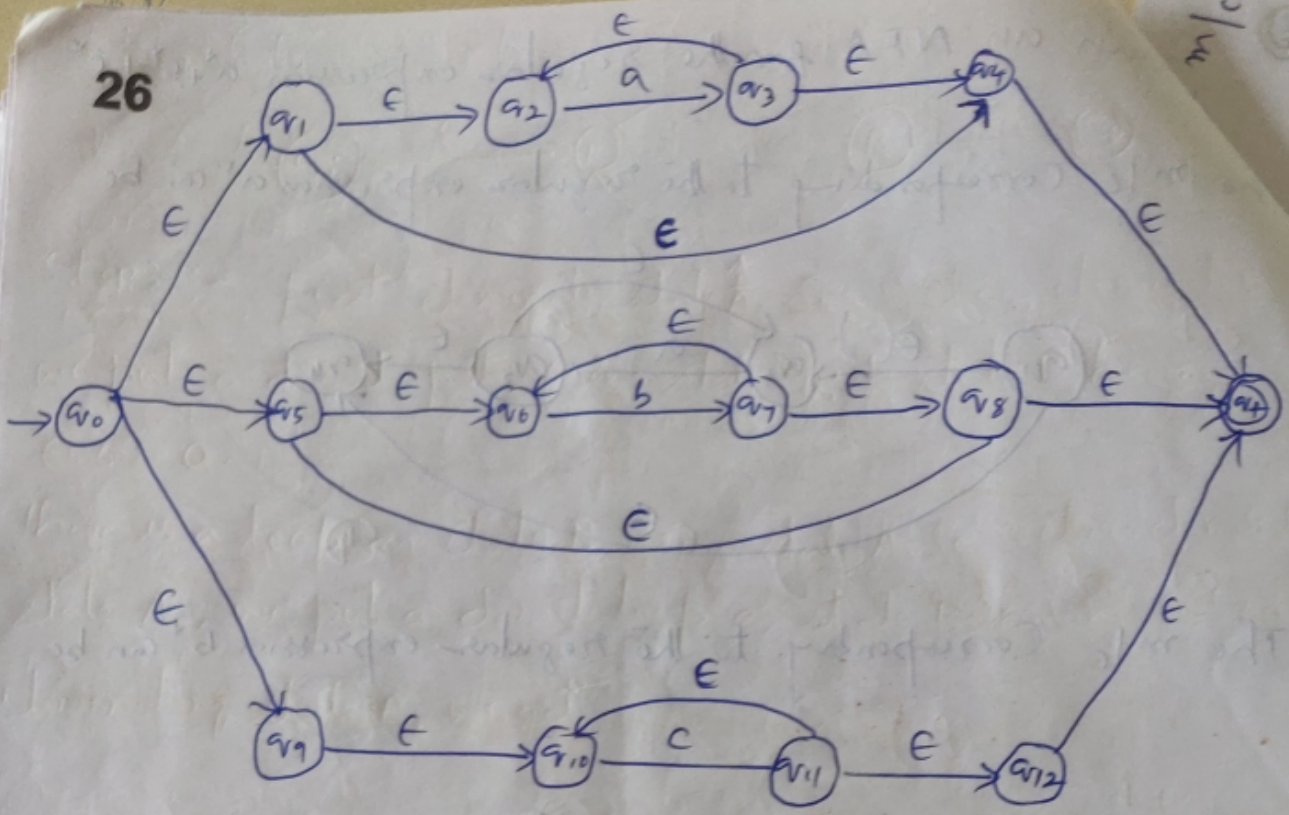


The m/c corresponding to the regular expression b^* can be



The m/c corresponding to the regular expression c^* can be



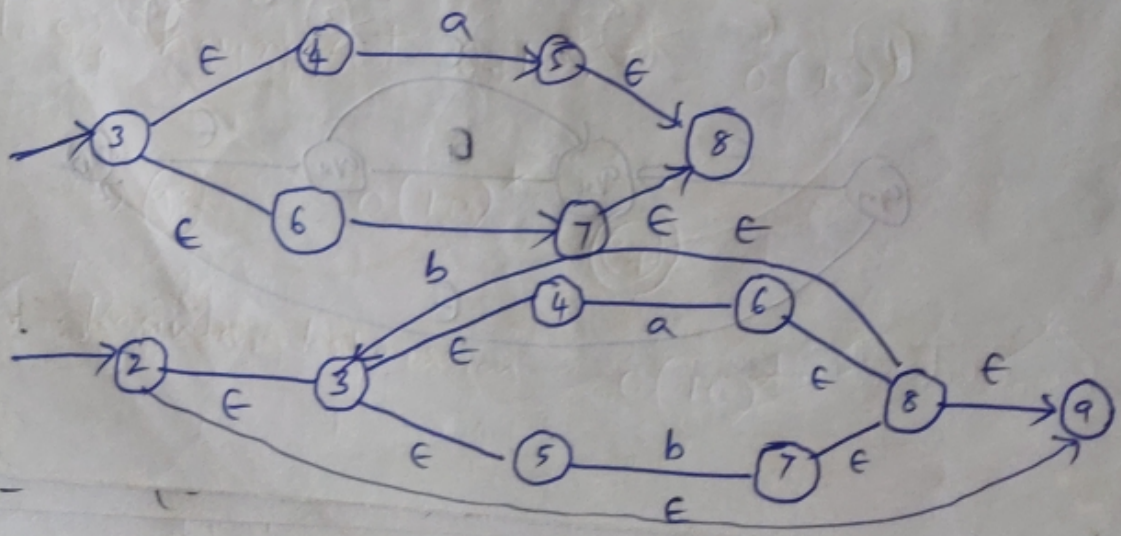


To accept the language $L(a^* + b^* + c^*)$

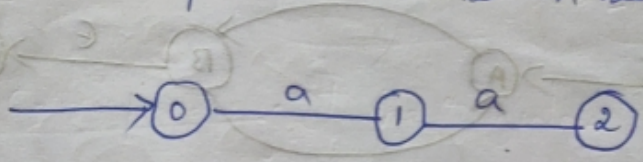
4) obtain an NFA for the regular expression $(a+b)^*aa$

$(a+b)^*$

The m/c to accept $(a+b)$ and $(a+b)^*$ is shown below:



e m/c to accept aa is shown below:

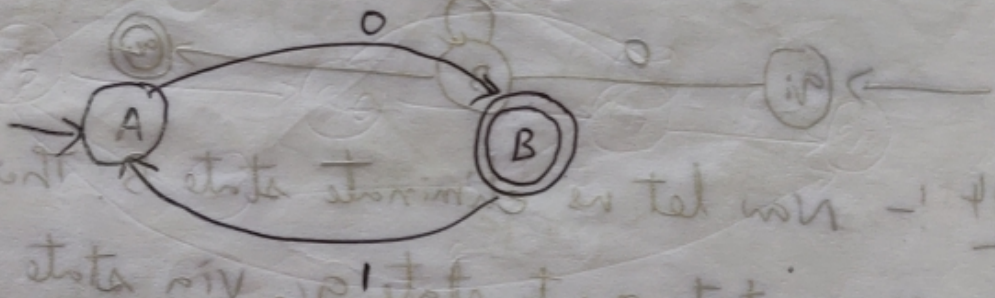


Conversion :- Converting DFA to regular Expression

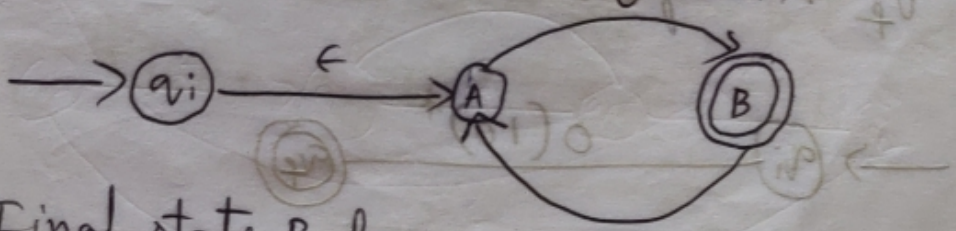
method

- State Elimination Method
- Arden's method.

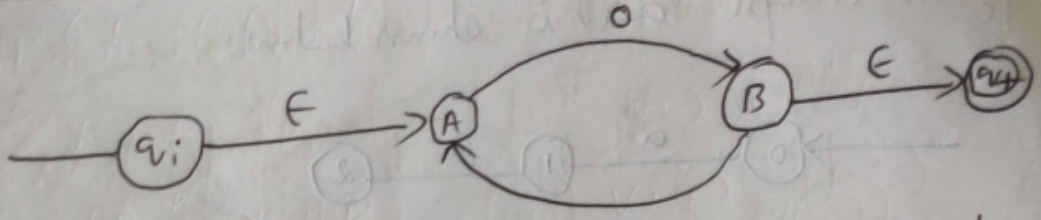
1) Find regular expression for the following DFA using State Elimination method



Step1 :- Initial state A has an incoming edge, So Create a new initial state q_i



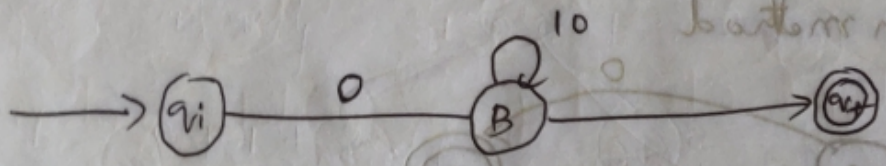
Step2 :- Final state B has an outgoing edge, So Create a new final state q_f



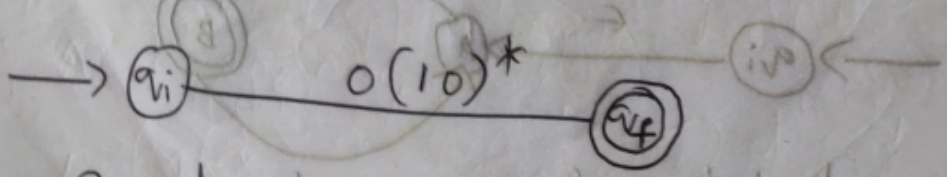
Step3 :- First eliminate state A. So, after eliminating A, we put a direct path from state q_i to state B having cost $\epsilon.0 = 0$.

There is a loop on state B using state A. So, after eliminating state A, we put a direct loop on state B having cost $1.0 = 10$

Eliminating state A, we get

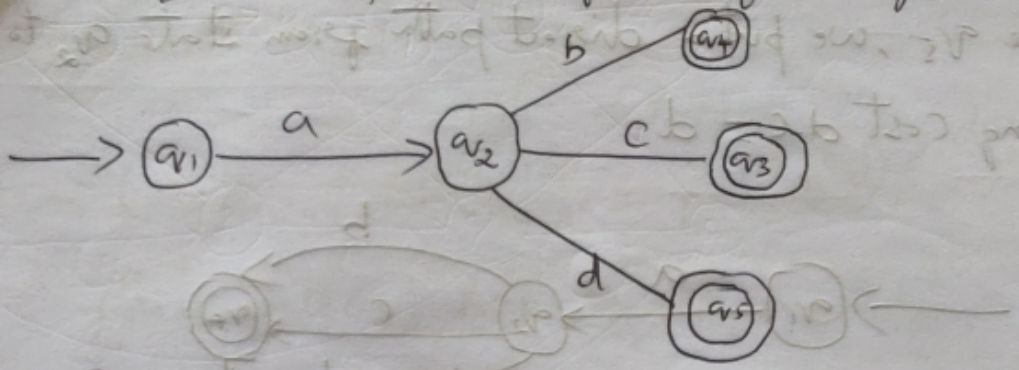


Step4 :- Now let us eliminate state B. There is a path going from state q_i to state q_f via state B. So, after eliminating state B, we put a direct path from state q_i to state q_f having cost $0.10^*.\epsilon = 0(10)^*$

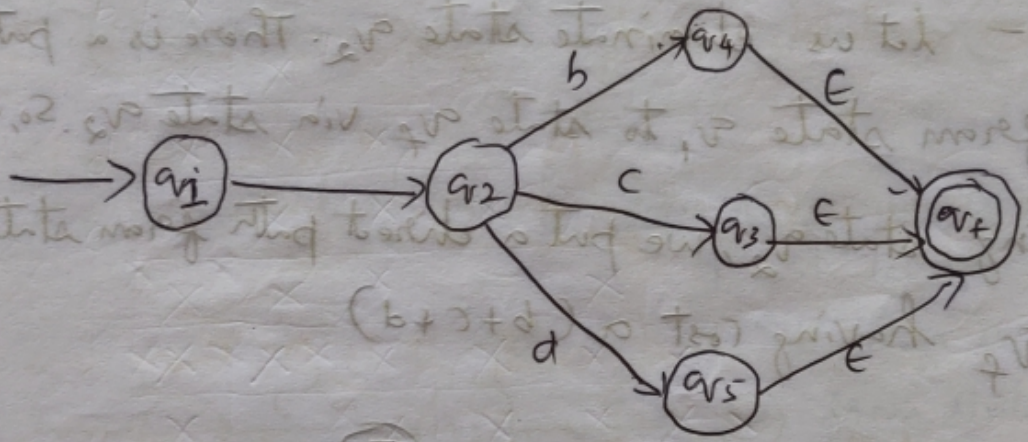


Regular Expression = $0(10)^*$.

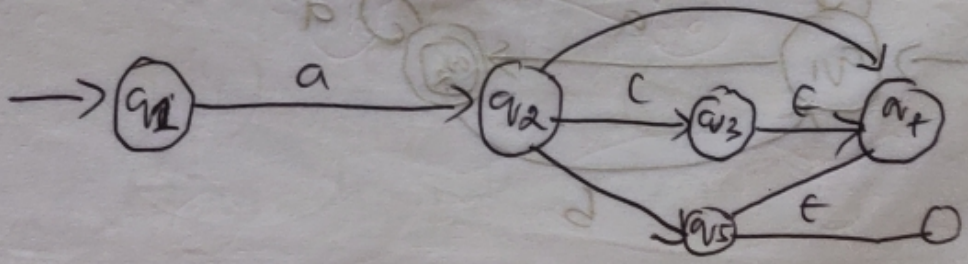
Find regular expression for the following DFA



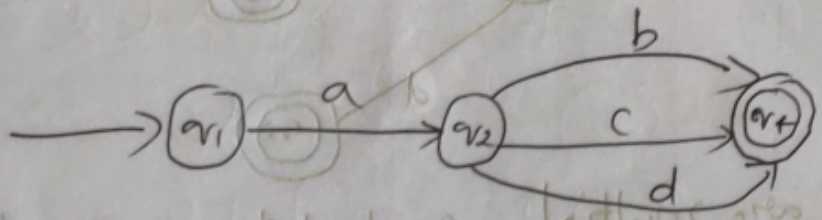
Step1 :- There are multiple final states, so Converting them into a single final state. The resulting DFA is

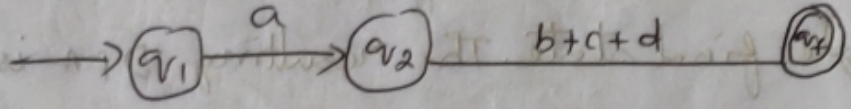


Step2 :- we start eliminating the intermediate states. There is a path going from state q_2 to state q_f via state q_4. So, after eliminating state q_4, we put a direct path from state q_2 to state q_f having cost $b\epsilon = b$

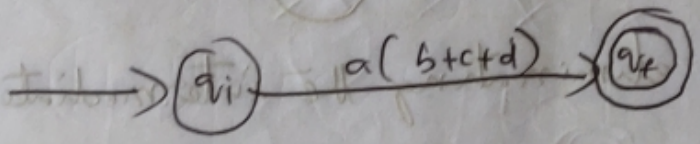


step4 :- let us eliminate state q_5. So after eliminating state q_5, we put a direct path from state q_2 to state q_5 having cost $d.\epsilon = d$



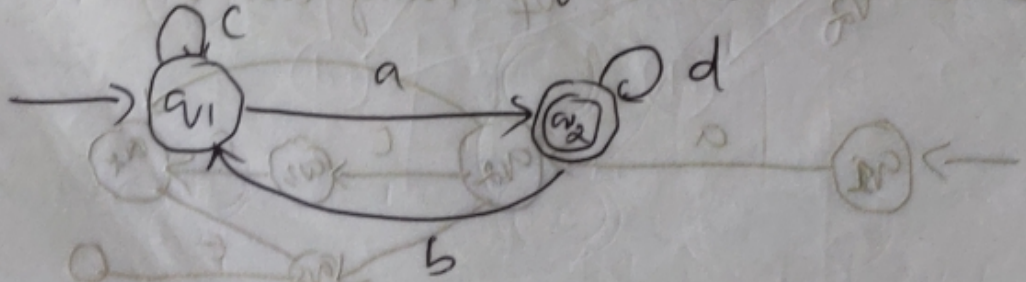


Step5 :- let us eliminate state q_2. There is a path going from state q_1 to state q_f via state q_2. So, after eliminating state q_2, we put a direct path from state q_1 to state q_f having cost $a(b+c+d)$

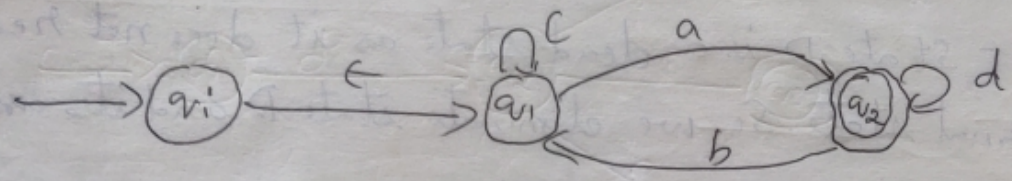


R.E = $a(b+c+d)$

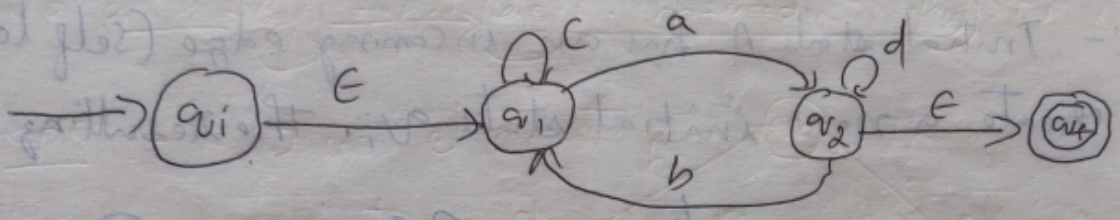
3) Find regular expression for the following DFA



Step 1 :- Initial state q_i has an incoming edge. So we create new initial state q_i

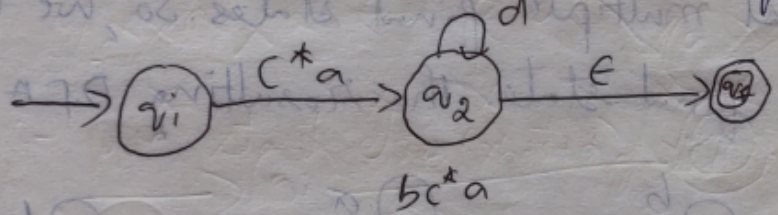


Step 2 :- Find state q_2 has an outgoing edge, so we create a new final state.

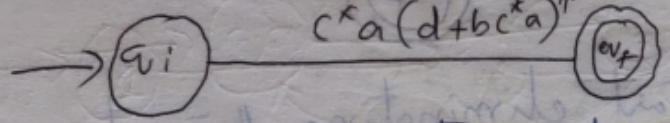


Step 3 :- We start eliminating the intermediate states.

First, let us eliminate state q_1. There is a path going from state q_i to state q_2 via state q_1. So,

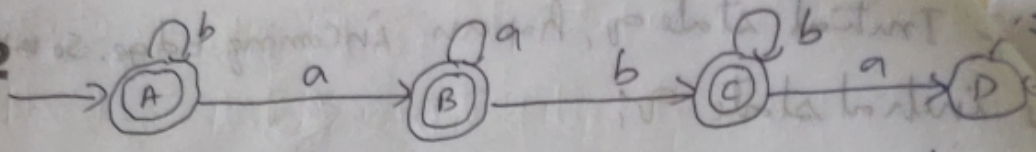


after eliminating state q_1, we put a direct path from state q_i to state q_2 having cost $\epsilon . c^*a = c^*a$. There is a loop on state q_2 using state q_1. So, after eliminating state q_1, we put a direct loop on state q_2 having cost $bc^*a = bc^*a$

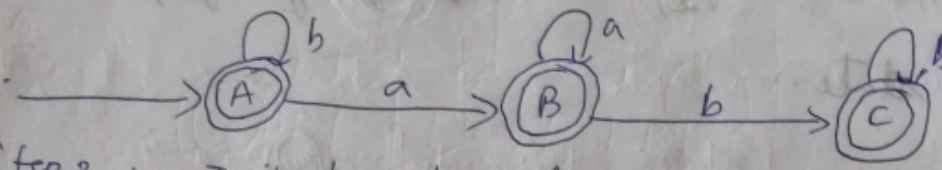


Step 4 :- Let us eliminate state q_2.

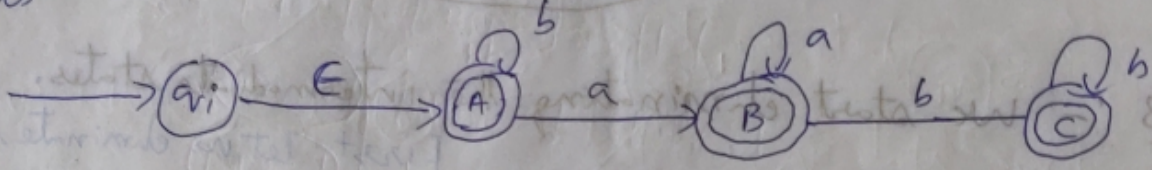
32



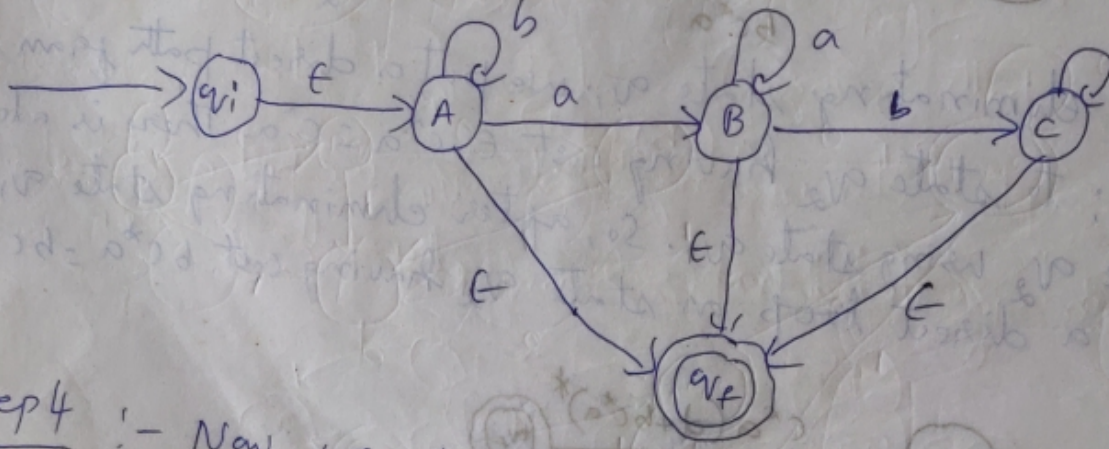
Step1 :- State D is a dead state as it does not reach any final state. So, we eliminate state D and its associated edges. The resulting DFA is



Step2 :- Initial state A has an incoming edge (self loop). So, we create a new initial state q_i. The resulting DFA is



Step3 :- There exist multiple final states. So, we convert them into a single final state. The resulting DFA is

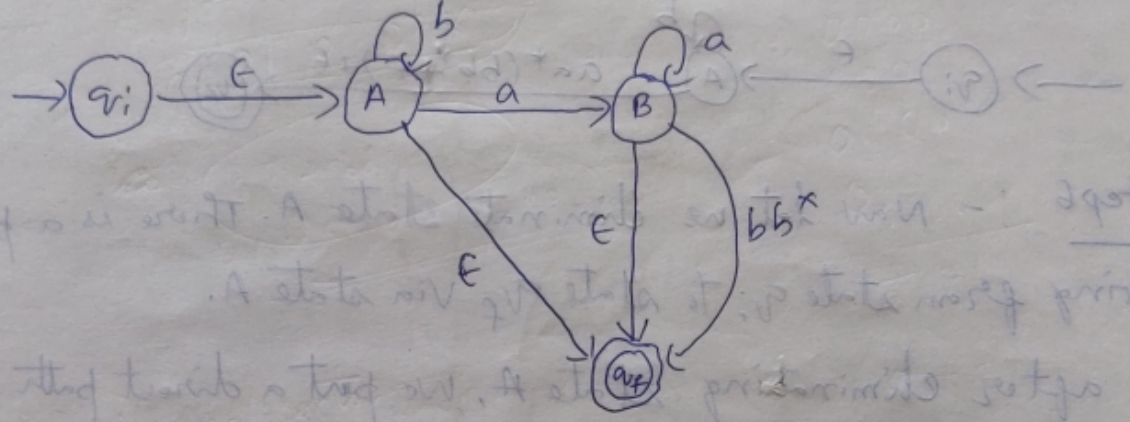


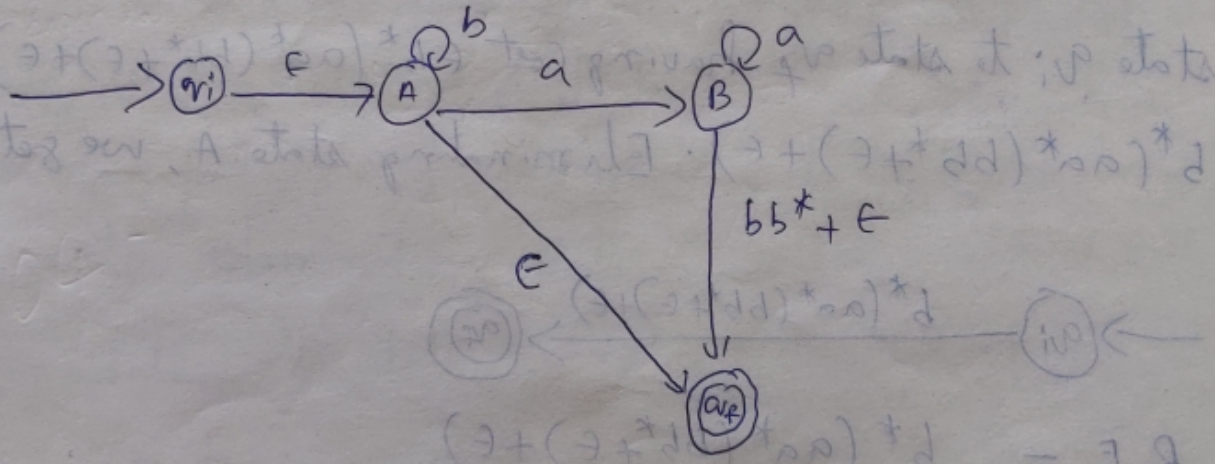
Step4 :- Now, we start eliminating the intermediate states. First, let us eliminate state C.

is not reach to state C and its associated.

There is a path going from state B to state q_f via state C. So, after eliminating state C, we put a direct path from state B to state q_f having cost $b.b^*\epsilon = bb^*$

Eliminating state C, we get

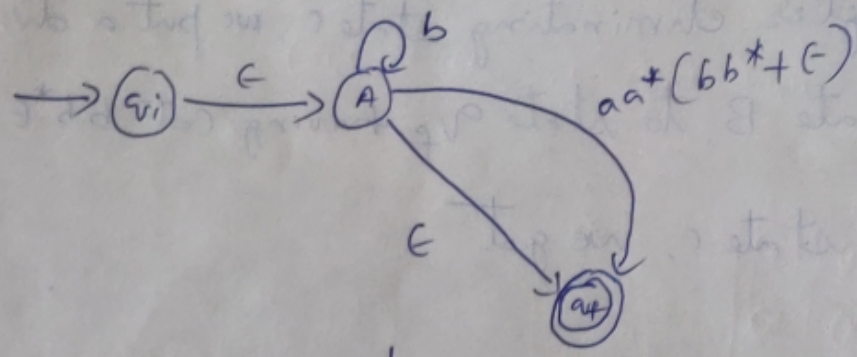


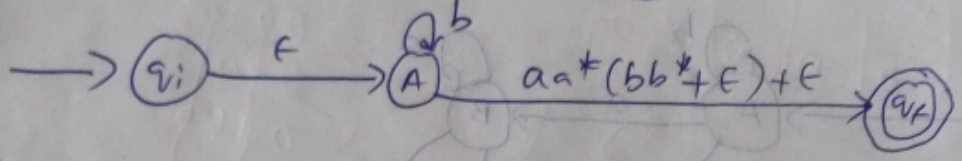


Step 5 :- Now let us eliminate state B. There is a path going from state A to state q_f via state B.

So, after eliminating state B, we put a direct path from state A to state q_f having cost $a.a^*(bb^* + \epsilon) = aa^*(bb^* + \epsilon)$

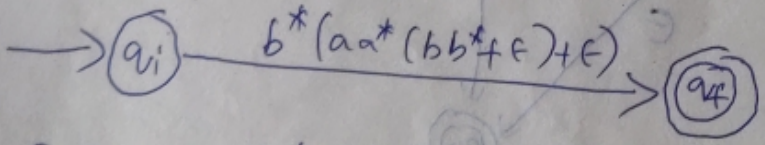
06 Eliminating state B, we get





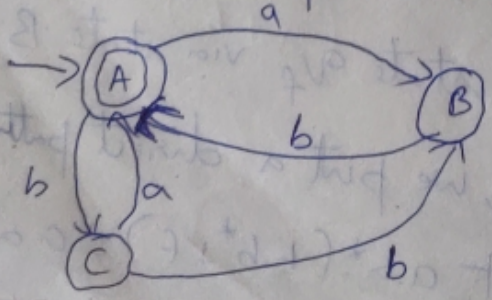
Step6 :- Now let us eliminate state A. There is a path going from state q_i to state q_f via state A.

So, after eliminating state A, we put a direct path from state q_i to state q_f having cost $\epsilon b^*(aa^*(bb^*+\epsilon)+\epsilon) = b^*(aa^*(bb^*+\epsilon)+\epsilon)$. Eliminating state A, we get

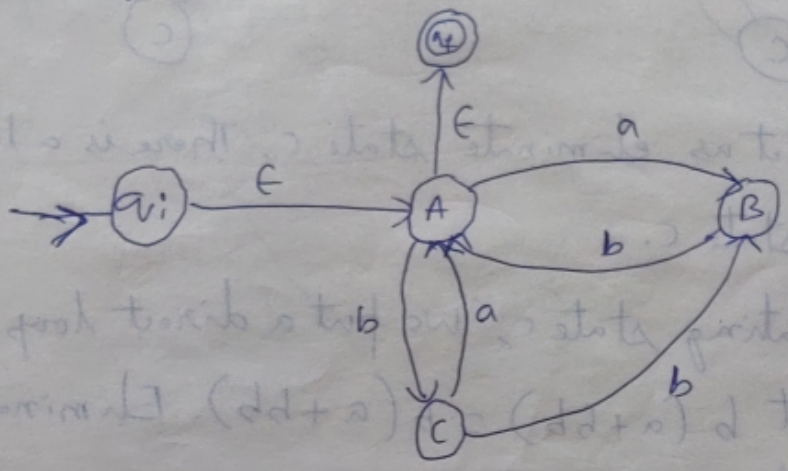


R.E = $b^*(aa^*(bb^*+\epsilon)+\epsilon)$

5) Find regular expression for the following DFA.

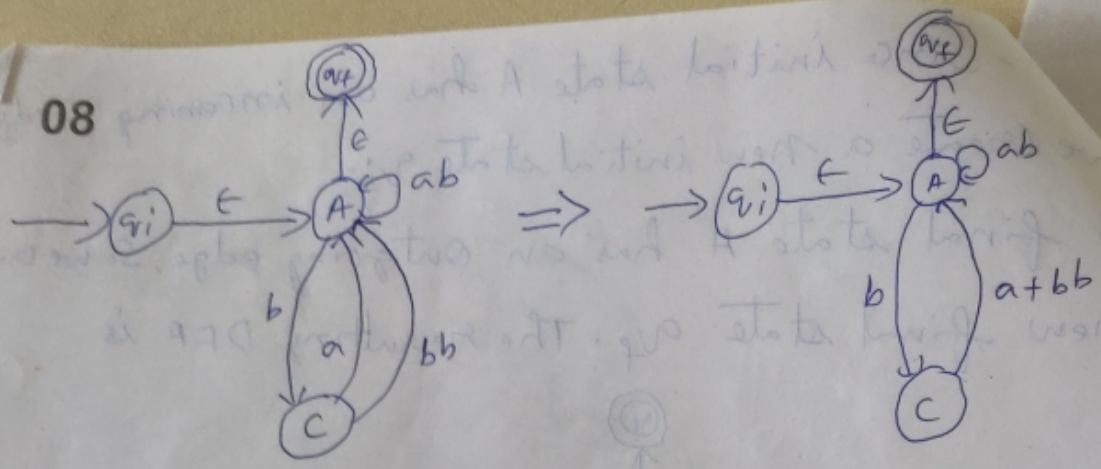


Step 1 :- Since initial state A has an incoming edge, we create a new initial state q_i.
Since final state A has an outgoing edge, so we create a new final state q_f. The resulting DFA is



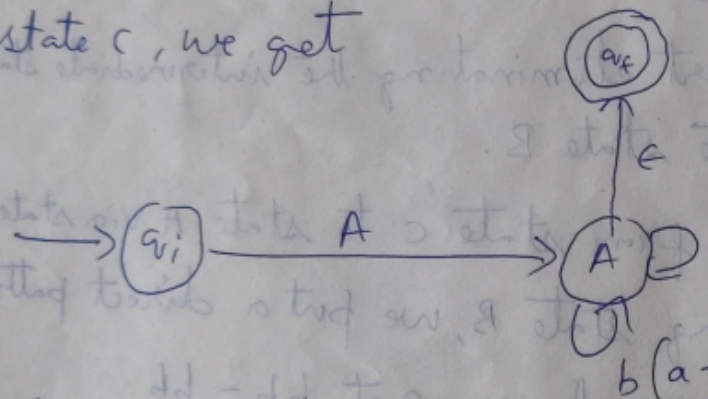
Step 2 :- Now, we start eliminating the intermediate state.
First, let us eliminate state B.
There is a path going from state C to state A via state B.
So, after eliminating state B, we put a direct path from state C to state A having cost $b.b = bb$.
There is a loop on state A using state B. So, after eliminating state B, we put a direct loop on state A having cost $a.b = ab$
Eliminating state B, we get,

08



Step 3 :- Now, let us eliminate state C, There is a loop on state A using state C.

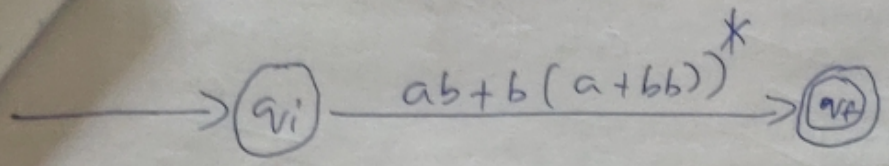
So, after eliminating state C, we put a direct loop on state A having cost $b.(a+bb) = b(a+bb)$. Eliminating state C, we get



Step 4 :- Now, let us eliminate state A. There is a path going from state q_i to state q_f via state A.

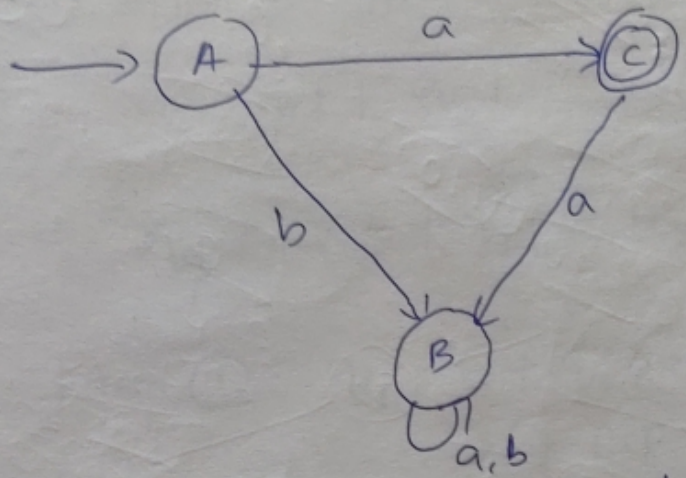
So, after eliminating state A, we put a direct path from state q_i to state q_f having cost $\epsilon.(ab+b(a+bb))^*\epsilon = (ab+b(a+bb))^*$

Eliminating state A, we get



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

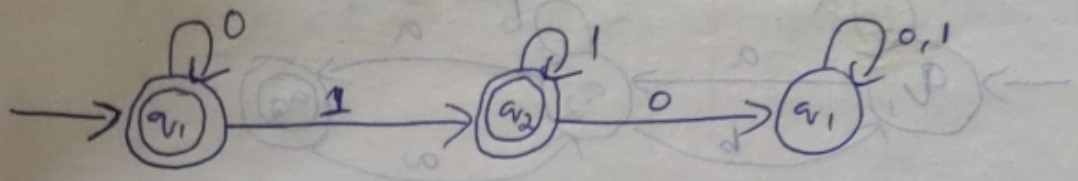
6) Find regular expression for the following DFA.



Step1 :- State B is a dead state as it does not reach to the final state. So, we eliminate state B and its associated edges. The resulting DFA is

$R.E = a$

Convert DFA to regular expression using Arden's theorem.



$q_1 = \epsilon + q_1 \cdot 0 \longrightarrow (1)$

$q_2 = q_1 1 + q_2 \cdot 1 \longrightarrow (2)$

$q_3 = q_2 \cdot 0 + q_3 \cdot 0 + q_3 \cdot 1 \longrightarrow (3)$

$R = q + RP$

$R = qP^*$

From (1) $q_1 = \epsilon + q_1 \cdot 0$ $\quad q = \epsilon$

$= q + R \cdot P$

$q_1 = \epsilon 0^* = 0^*$

From (2) $q_2 = q_1 \cdot 1 + q_2 \cdot 1$

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

$R = qP^*$

$q_2 = 0^* 1 1^*$

$= 0^* 1^*$

$RE_1 + RE_2$

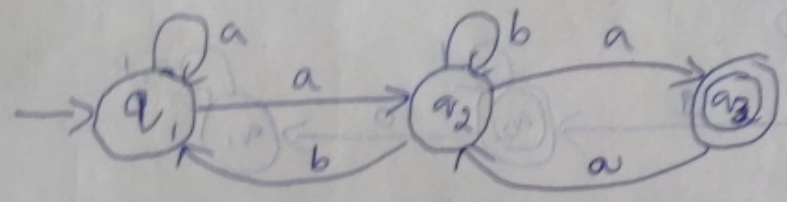
$= 0^* + 0^* 1^*$

$= 0^* (\epsilon + 11^*)$

$= 0^* 1^*$

$\epsilon + 11^* = 1^*$

2) 10 Convert finite automata to Regular Ex



$q_1 = q_1 a + q_2 b + \epsilon$ ——— (1)

$q_2 = q_2 b + q_1 a + q_3 a$ ——— (2)

$q_3 = q_2 a$ ——— (3)

From (2) $q_2 = q_2 b + q_1 a + q_2 aa$

$= q_1 a + q_2 b + q_2 aa$

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

$R = Q + R P$

$q_2 = q_1 a (b + aa)^*$ ——→ (4)

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

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

$R = Q + R P$

$q_1 = (a + a(b + aa)^* b)^*$ ——→ (5)

Theorem :- To construct the expression R_{ij}^{k}, use inductive definition starting at k=0 and k=n.

Basis :- K=0 i.e., path must have no intermediate states at all. There are two kinds of path:

i) An arc from node i to node j

ii) A path of length 0, that must consists of only some node i.

Case 1 :- $i \neq j$

DFA have input symbol 'a' such that there is transition from state i to state j on symbol a.

i) If there is no such symbol a, then $R_{ij}^{(0)} = \phi$

ii) If there is exactly one such symbol a, then $R_{ij}^{(0)} = a$

iii) If there are $a, a_2 - - a_k$ that label arc from state i to state j then $R_{ij}^{(0)} = a_1 + a_2 - - + a_k$

Case 2 :- $i = j$ i.e., path of length is zero and all loops from i to itself.

i) If there is such symbol 'a', then $R_{ij}^{(0)} = \phi + \epsilon$

ii) If there is exactly one such symbol a, then $R_{ij}^{(0)} = a + \epsilon$

iii) If there are $a_1, a_2 - - a_k$ that label arcs from state i to state j then $R_{ij}^{(0)} = a_1 + a_2 - - + a_k + \epsilon$

Induction :- Suppose there is a path from state ... that through no state higher than k.

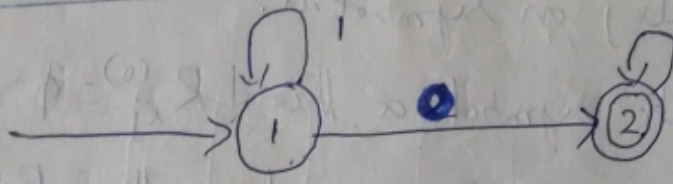
There are two Cases:

i) The path does not go through state k at all. The label of path is in the language $R_{ij}^{(k-1)}$

ii) The path goes through states k at least once. We break the path into several pieces:

$$R_{ij}^{k} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^{*} R_{kj}^{(k-1)}$$

1)



$k = 0$

$R_{11}^{(0)} = \epsilon + 1$

$R_{12}^{(0)} = 0$

$R_{21}^{(0)} = \phi$

$R_{22}^{(0)} = \epsilon + 0 + 1$

$k = 1$

$R_{11}^{(1)} = 1^{*}$

$R_{12}^{(1)} = 1^{*}0$

$R_{21}^{(1)} = \phi$

$R_{22} = 0 + 1 + \epsilon = (0+1)^{*}$

$$R_{11}^{(1)} = R_{11}^{0} + R_{11}^{0} \left(R_{11}^{0}\right)^{*} R_{11}^{0}$$

$$= (\epsilon + 1) + (\epsilon + 1)(\epsilon + 1)^{*}(\epsilon + 1)$$

$$= (\epsilon + 1) + (\epsilon + 1)\, 1^{*} (\epsilon + 1)$$

$$= \epsilon + 1 + 1^{*}$$

$R_{11}^{(1)} = 1^{*}$

$$R_{12}^{(1)} = R_{12}^{0} + R_{11}^{0}\,(R_{11}^{0})^{*} R_{12}^{0}$$

$$= 0 + (\epsilon+1)\,(\epsilon+1)^{*} 0$$

$$= 0 + (\epsilon+1)\,1^{*} 0$$

$$R_{12}^{(1)} = 0 + 1^{*} 0$$ ✓

$$R_{22}^{(1)} = R_{22}^{0} + R_{21}^{0}\,R_{11}^{0}\,R_{12}^{0}$$

$$= \epsilon + 0 + 1 + \phi\,(\epsilon+1)^{*} 0$$

$$= \epsilon + 0 + 1 + \phi$$

$$R_{22}^{(1)} = \epsilon + 0 + 1$$ ✓

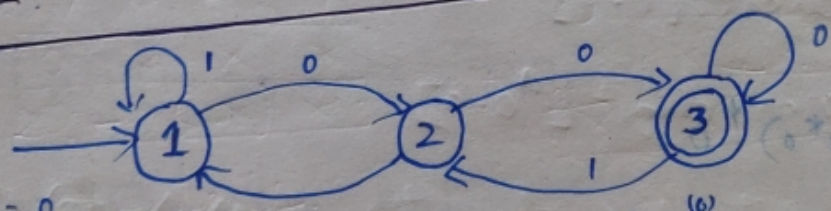
$$R_{12}^{(2)} = R_{12}^{1} + R_{12}^{1}\,(R_{22}^{1})^{*} R_{22}^{1}$$

$$= 1^{*}0 + 1^{*}0\,(\epsilon+0+1)^{*}(\epsilon+0+1)$$

$$= 1^{*}0 + 1^{*}0\,(0+1)^{*}$$

$$\boxed{R_{12}^{(2)} = 1^{*}0\,(0+1)^{*}}$$ ✓

ii)



$k = 0$

$R_{11}^{(0)} = \epsilon + 1$

$R_{12}^{(0)} = 0$

$R_{13}^{(0)} = \phi$

$R_{21}^{(0)} = 1$

$k = 1$

$R_{22}^{(0)} = \phi + \epsilon$

$R_{23} = 0$

$R_{31}^{(0)} = \phi$

$R_{32}^{(0)} = 1$

$R_{33}^{(0)} = \epsilon + 0$

10 $k=1$

$R_{11}^{(1)} = 1^*$ | $R_{21}^{(1)} = 11^*$ | $R_{31}^{(1)} = \phi$

$R_{12}^{(1)} = 1^*0$ | $R_{22}^{(1)} = 11^*0 + \epsilon$ | $R_{32}^{(1)} = 1$

$R_{13}^{(1)} = \phi$ | $R_{23}^{(1)} = 0$ | $R_{33}^{(1)} = 0 + \epsilon$

$k=2$

$R_{11}^{(2)} = 1^* + 1^*0\,(11^*0)^*11^*$

$R_{12}^{(2)} = 1^*0 + 1^*0\,(11^*0)^*11^*0$

$R_{13}^{(2)} = 1^*0\,(11^*0)^*0$

$R_{21}^{(2)} = -$

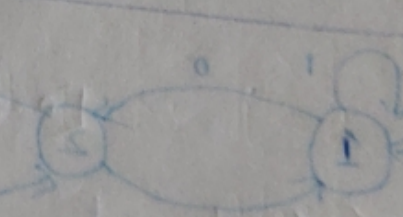
$R_{22}^{(2)} = -$

$R_{23}^{(2)} = -$

$R_{31}^{(2)} = \times$

$R_{32}^{(2)} = -$

$R_{33}^{(2)} = 0 + 1\,(11^*0)^*0$



$R_{32}^{(1)} = R_{32}^{(0)} + R_{31}^{(0)}\left(R_{11}^{(0)}\right)^* R_{12}^{(0)}$

$= 1 + \phi\,(\epsilon+1)^*0$

$= 1 + \phi$

$R_{11}^{(2)} = R_{11}^{(1)} + R_{12}^{(1)}\left(R_{22}^{(1)}\right)^* R_{21}^{(1)} = 1^* + 1^*0\,(11^*0)^*11^*$

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

$R_{12} = R_{12}^{(1)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{22}^{(1)}$

$= 1^*0 + 1^*0 (11^*0 + \epsilon)^* (11^*0 + \epsilon)$

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

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

Applications of Regular Expression :-

1) Regular expressions are used in UNIX operating System.
2) Regular expressions are used in Compilation (Lexical analysis).
3) Regular expressions are used in programming language like (Java).
4) Regular expressions are used in web search Engines.
5) Regular expressions are used in Software Engineering. 6) Text editors.

Regular expressions in Java :- Regular expressions are a language of string patterns built-in to most modern programming languages, including Java 1.4 onwards, they can be used for searching, extracting and modifying text.

The Java package Java.util.regex contains classes for working with regexps in Java.

Character classes :- The character classes are used to define the contents of the pattern.

Example, what should the pattern look for ?

→ \d : A digit: [0-9]

→ \D : A non-digit [^0-9]

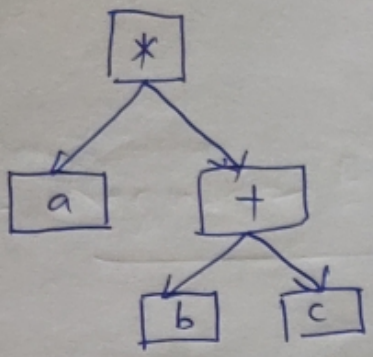
→ \s : A white space character [\t\n\x0B\f\r]

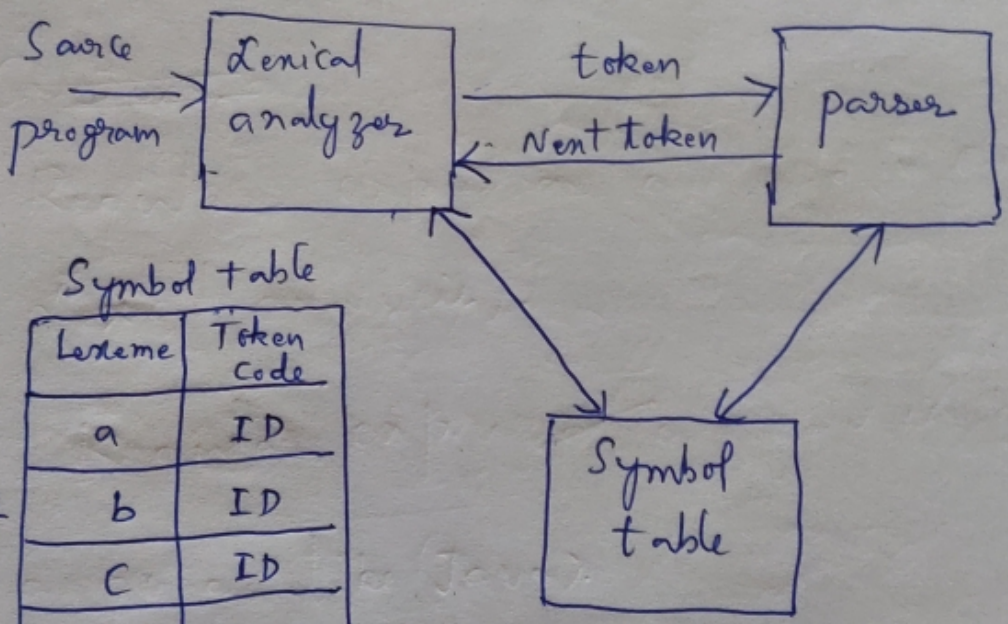
→ \S : A non-whitespace character. [^\s]

→ \w : A word character : [a-zA-Z_0-9]

→ \W : A non-word character [^\W]

Regular Expressions in Lexical analysis :- Interaction of lexical analyzer with parser.





	Lexeme	Token code
1	a	ID
2	b	ID
3	c	ID
	...	...

The lexical analyzer (lexer) reads source code & generates a stream of tokens

Token is a identifier, keyword, Number, operator, punctuations.

Tokens can be described using regular expressions.