

# Unit I – Part II

- 1. Regular Expressions*
- 2. Conversion of regular expression to NFA – Thompson's*
- 3. Converting Regular expression directly to DFA*

## *2. Regular Expressions*

# Regular expression

- Method to represent a language (regular expression)
- $L = \{\epsilon, a, aa, aaa, \dots\} = a^*$
- Let 'R' be a regular expression over alphabet  $\Sigma$  if R is-
  1.  $\epsilon$  is regular expression denoting the set  $\{\epsilon\}$ .
  2.  $\emptyset$  is regular expression denoting the empty set  $\{\}$ .
- (Phi means no string accepted i.e. no final state.

Epsilon means there is a string of length 0 & it is accepted i.e. there is a final state.)

3. For each symbol **a** belong to  $\Sigma$ , '**a**' is regular expression denoting set {a}.
4. Union of two regular expression is also regular.
5. Concatenation of two regular expression is also regular.
6. Kleene closure of two regular expression is also regular.
7. If R is regular language, then  $(R)^*$  is also regular.

Nothing else, repeat step 1 to 7 recursively.

Ex:  $(a + b)^*$

# Regular expression

1. Regular expression for finite language
2. Regular expression for infinite language

## Regular expression for Finite language:

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

- No string  $\{\}$  : -  $\emptyset$
- Length 0  $\{\epsilon\}$  :-  $\epsilon$
- Length 1  $\{a, b\}$  :-  $(a + b)$
- Length 2  $\{aa, ab, ba, bb\}$  :-  $(aa + ab + ba + bb) = a(a+b)+b(a+b) = (a+b)(a+b)$
- Length 3  $\{aaa, bbb, aab, aba, \dots\}$  :-  $(a+b)(a+b)(a+b)$
- Atmost 1 (0,1)  $\{\epsilon, a, b\}$  :-  $(\epsilon + a + b)$
- Atmost 2 :-  $(\epsilon + a + b)(\epsilon + a + b)$

# Regular expression for infinite language

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

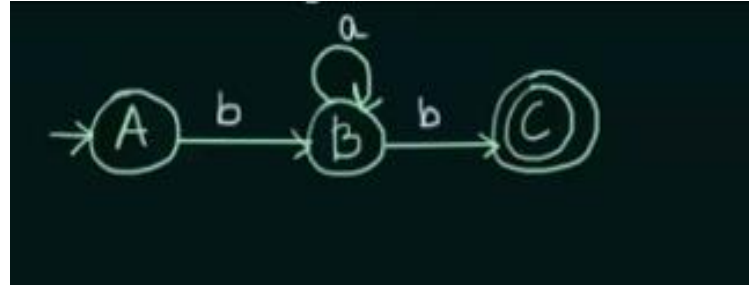
1. All strings having a single 'b' :-  $a^*b a^*$
2. All strings having at least one 'b' :-  $(a + b)^* b (a + b)^*$
3. All strings having 'bbbb' as substring :-  $(a + b)^* bbbb (a + b)^*$
4. All strings end with 'ab' :-  $(a + b)^* ab$
5. All strings start with 'ba' :-  $ba (a + b)^*$
6. All strings beginning and end with 'a' :-  $a (a + b)^* a$
7. All strings containing 'a' :-  $(a + b)^* a (a + b)^*$
8. All strings starting and end with different symbol :-  $a (a + b)^* b$  or  $b (a + b)^* a$

# Closure Properties of Regular Languages

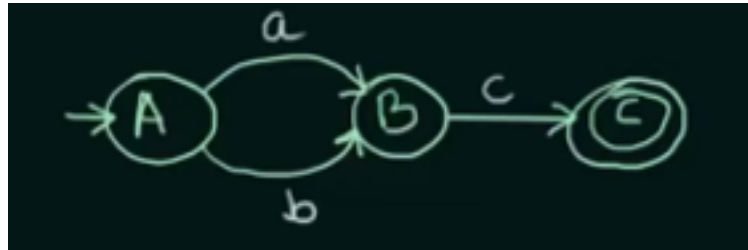
- **Union** : If  $L_1$  and  $L_2$  are two regular languages, their union  $L_1 \cup L_2$  will also be regular. For example,  $L_1 = \{a^n \mid n \geq 0\}$  and  $L_2 = \{b^n \mid n \geq 0\}$   
 $L_3 = L_1 \cup L_2 = \{a^n \cup b^n \mid n \geq 0\}$  is also regular.
- **Intersection** : If  $L_1$  and  $L_2$  are two regular languages, their intersection  $L_1 \cap L_2$  will also be regular. For example,  
 $L_1 = \{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$  and  $L_2 = \{a^m b^n \cup b^n a^m \mid n \geq 0 \text{ and } m \geq 0\}$   
 $L_3 = L_1 \cap L_2 = \{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$  is also regular.
- **Concatenation** : If  $L_1$  and  $L_2$  are two regular languages, their concatenation  $L_1.L_2$  will also be regular. For example,  
 $L_1 = \{a^n \mid n \geq 0\}$  and  $L_2 = \{b^n \mid n \geq 0\}$   
 $L_3 = L_1.L_2 = \{a^m . b^n \mid m \geq 0 \text{ and } n \geq 0\}$  is also regular.
- **Kleene Closure** : If  $L_1$  is a regular language, its Kleene closure  $L_1^*$  will also be regular. For example,  
 $L_1 = (a \cup b)$   
 $L_1^* = (a \cup b)^*$
- **Complement** : If  $L(G)$  is regular language, its complement  $L'(G)$  will also be regular. Complement of a language can be found by subtracting strings which are in  $L(G)$  from all possible strings. For example,  
 $L(G) = \{a^n \mid n > 3\}$   
 $L'(G) = \{a^n \mid n \leq 3\}$

# Converting Regular expression to DFA

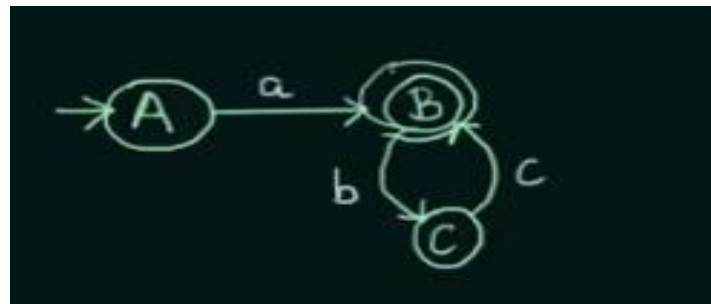
1.  $b a^* b = \{bb, bab, baab, baaab, \dots\}$



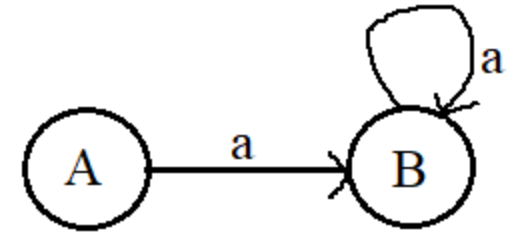
2.  $(a + b) c = \{ac, bc\}$



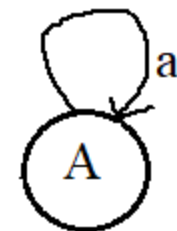
3.  $a(bc)^* = \{a, abc, abcbc, abcbcbc, \dots\}$



4.  $a^+ = \{a, aa, aaa, \dots\}$



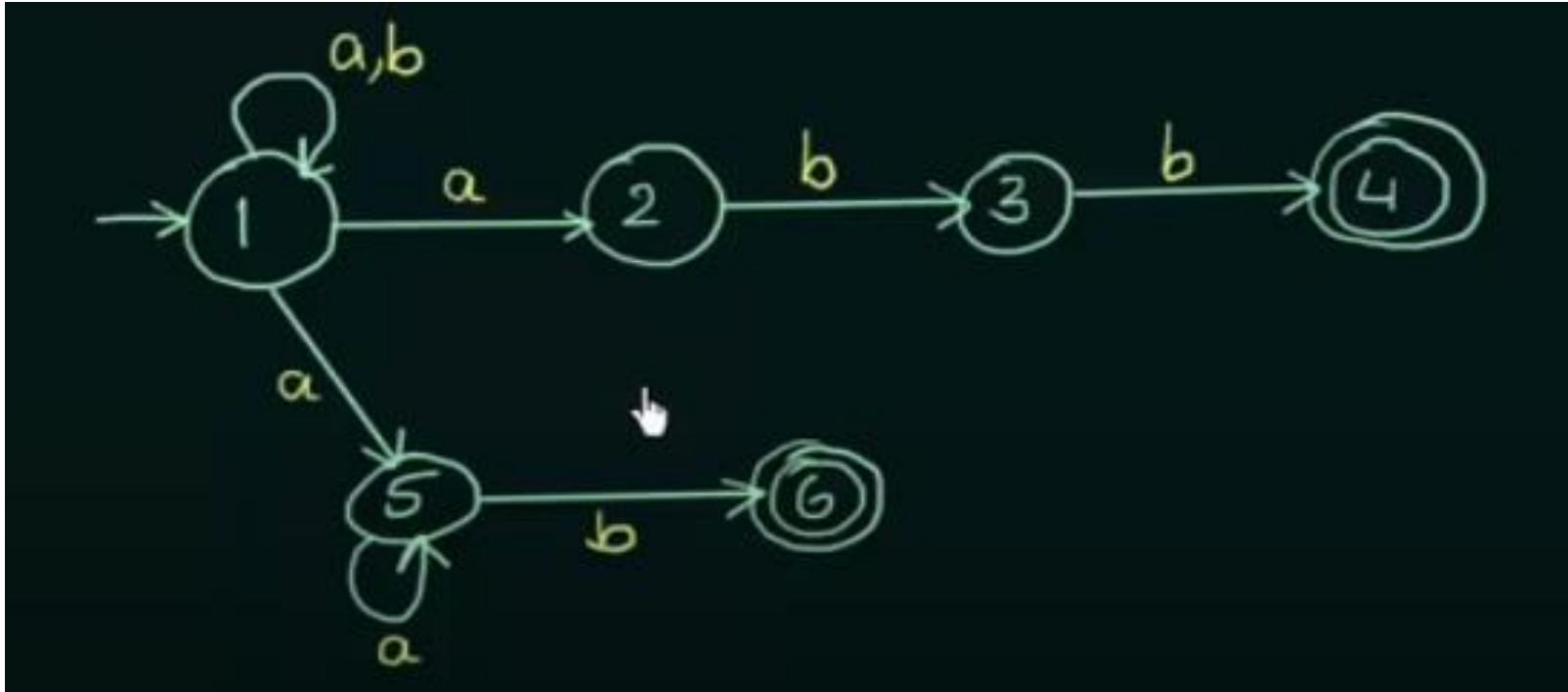
5.  $a^* = \{\epsilon, a, aa, aaa, \dots\}$





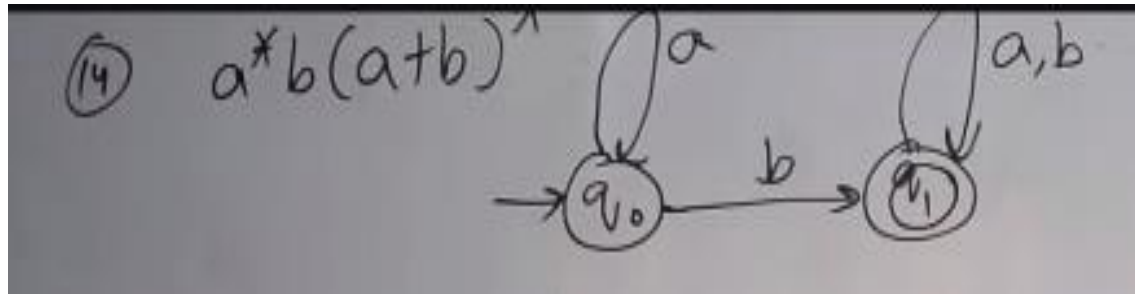
# Converting Regular expression to DFA

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

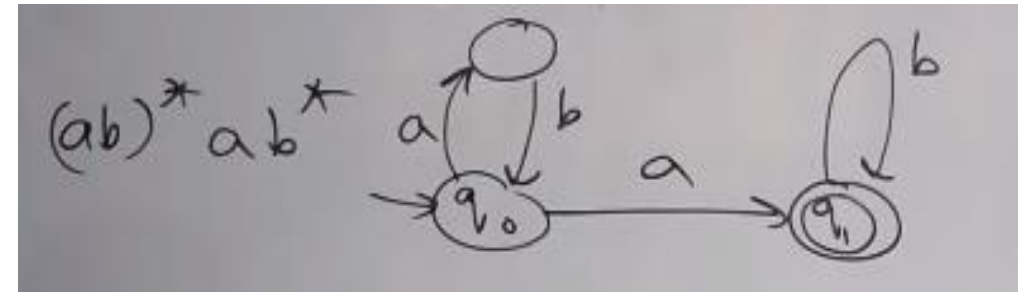
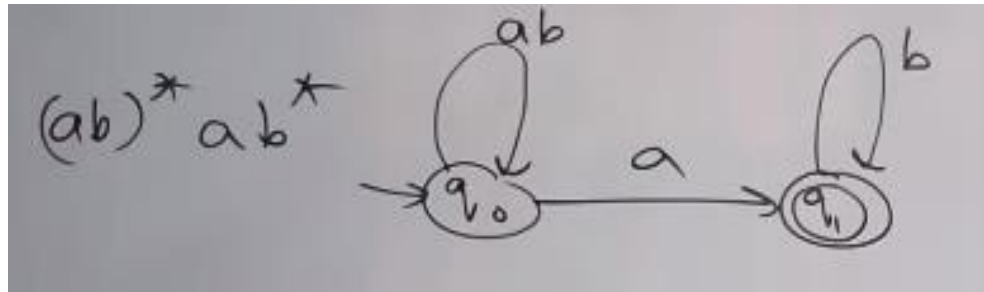


# Conversion of regular expression to NFA

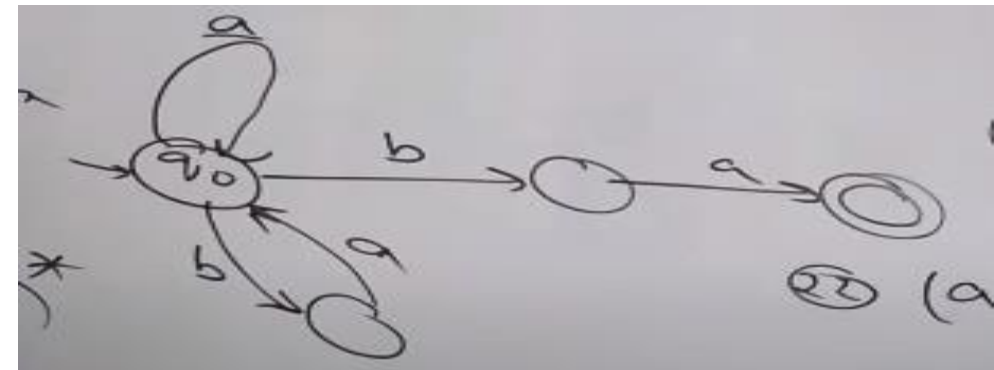
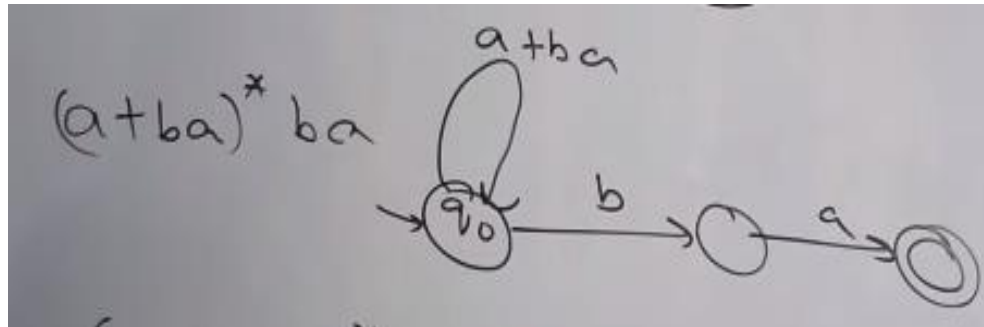
1



2



3



### 3. Conversion of regular expression to NFA

Regular expression can be converted into DFA by the following methods:

(i) Thompson's subset construction

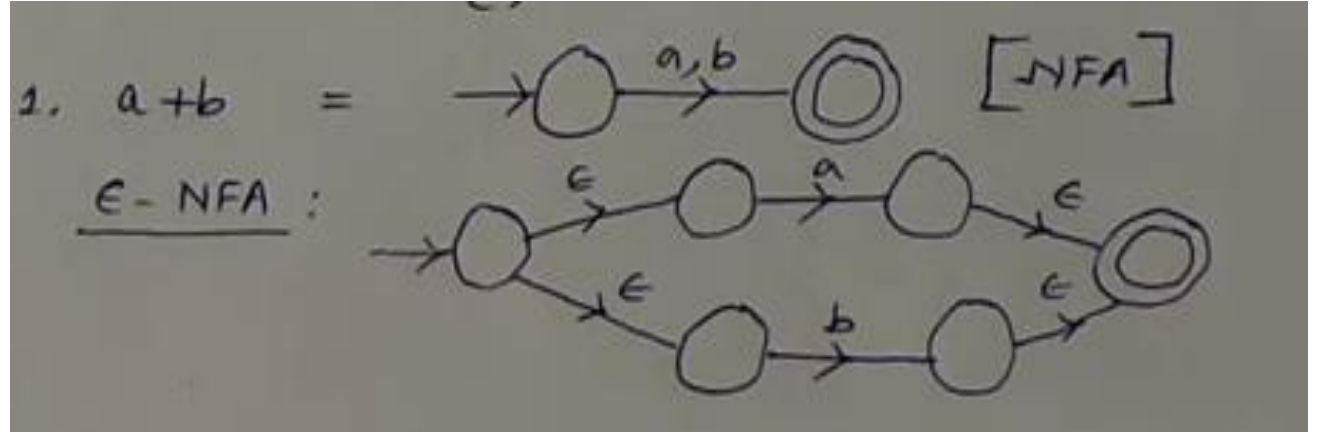
- Given regular expression is converted into NFA using thompson's construct.
- Resultant NFA is converted into DFA

(ii) Direct Method

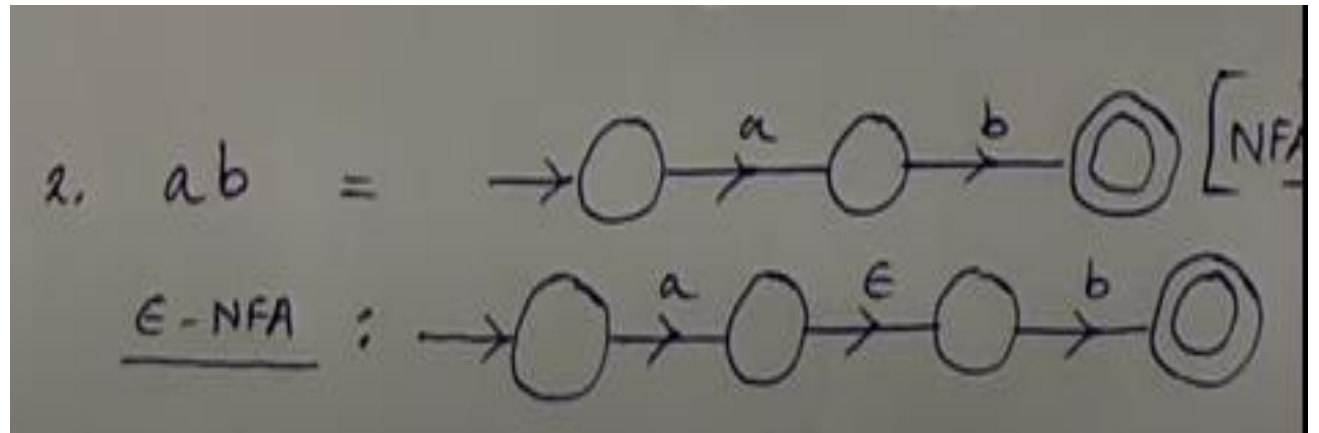
- In direct method, given regular expression is converted directly into DFA.

# Thompson's Construction for Conversion of Regular Expression to NFA

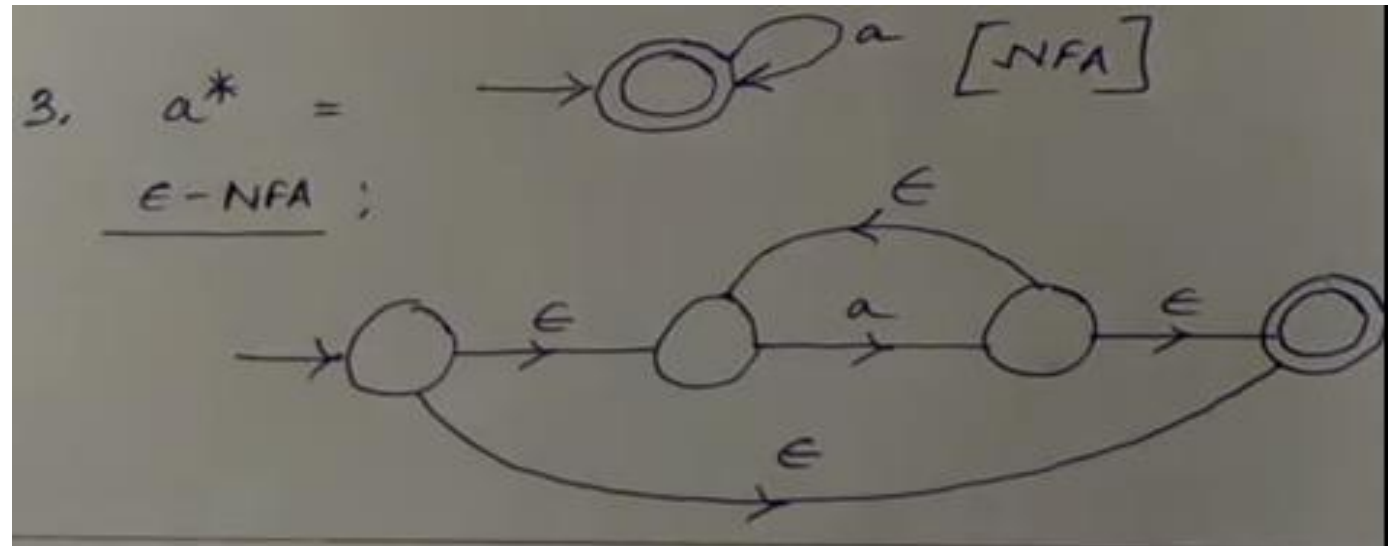
- Union:  $r = a + b$



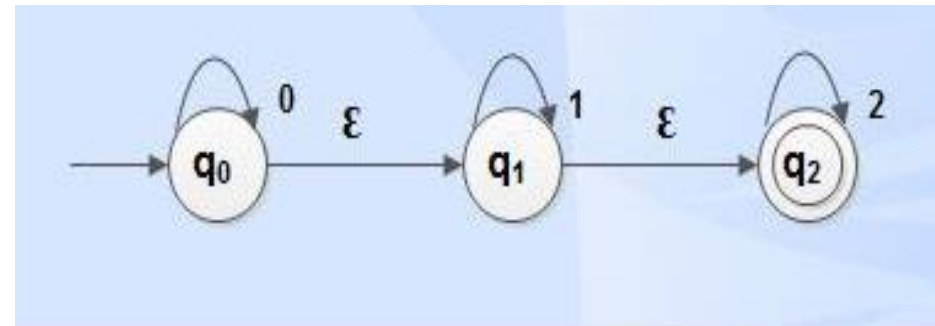
- Concatenation:  $r = r_1 r_2$



- **Closure:**  $r = a^*$

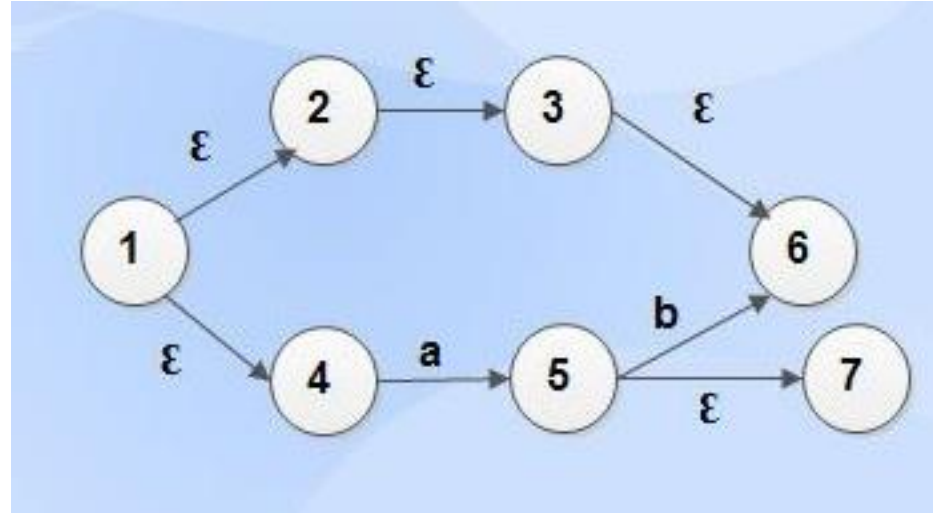


- **$\epsilon$  –closure:**  $\epsilon$  – Closure is the set of states that are reachable from the state concerned on taking empty string as input. It describes the path that consumes empty string ( $\epsilon$ ) to reach some states of NFA.
- $\epsilon$  -closure( $q_0$ ) = {  $q_0$ ,  $q_1$ ,  $q_2$  }
- $\epsilon$  –closure( $q_1$  ) = {  $q_1$ ,  $q_2$  }
- $\epsilon$  -closure( $q_2$ ) = {  $q_2$  }



Example:

- $\epsilon$ -closure (1) = {1, 2, 3, 4, 6}
- $\epsilon$ -closure (2) = {2, 3, 6}
- $\epsilon$ -closure (3) = {3, 6}
- $\epsilon$ -closure (4) = {4}
- $\epsilon$ -closure (5) = {5, 7}
- $\epsilon$ -closure (6) = {6}
- $\epsilon$ -closure (7) = {7}



## Sub-set Construction

- Given regular expression is converted into NFA.
- Then, NFA is converted into DFA.

## Steps:

1. Convert into  $\epsilon$ -NFA using above rules for operators (union, concatenation and closure) and precedence.
2. Find  $\epsilon$ -closure of all states.
3. Start with epsilon closure of start state of  $\epsilon$ -NFA.
4. Apply the input symbols and find its epsilon closure.

$Dtran[state, input\ symbol] = \epsilon\text{-closure}(\text{move}(\text{state}, \text{input symbol}))$

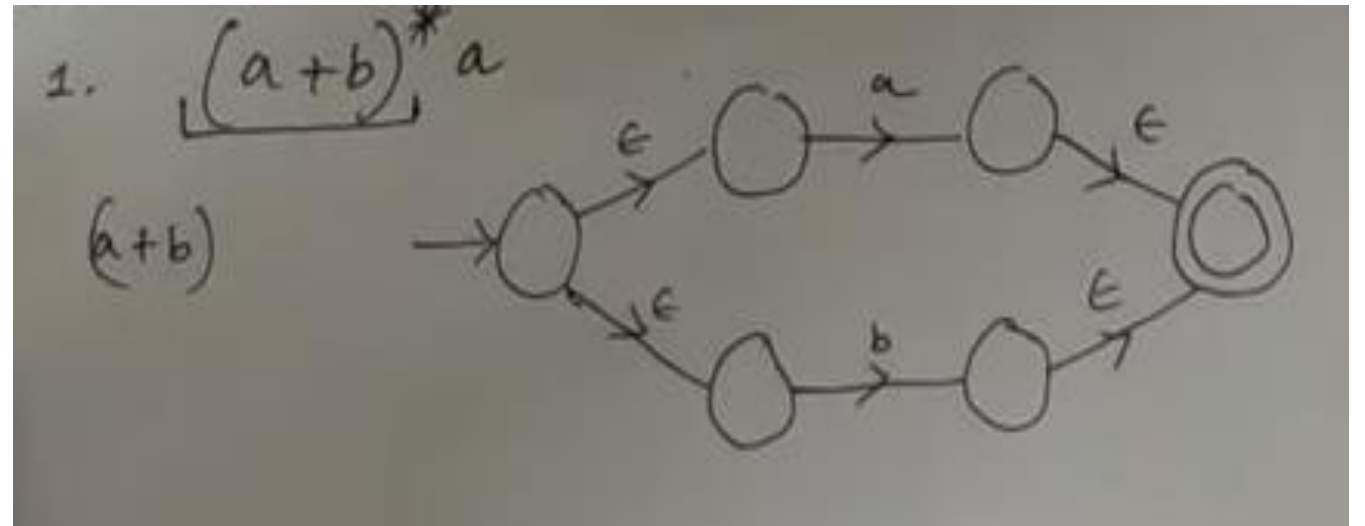
where  $Dtran$  is transition function of DFA

5. Analyze the output state to find whether it is a new state.
6. If new state is found, repeat step 4 and step 5 until no more new states are found.
7. Construct the transition table for  $Dtran$  function.
8. Draw the transition diagram with start state as the  $\epsilon$ -closure (start state of NFA) and final state is the state that contains final state of NFA drawn.

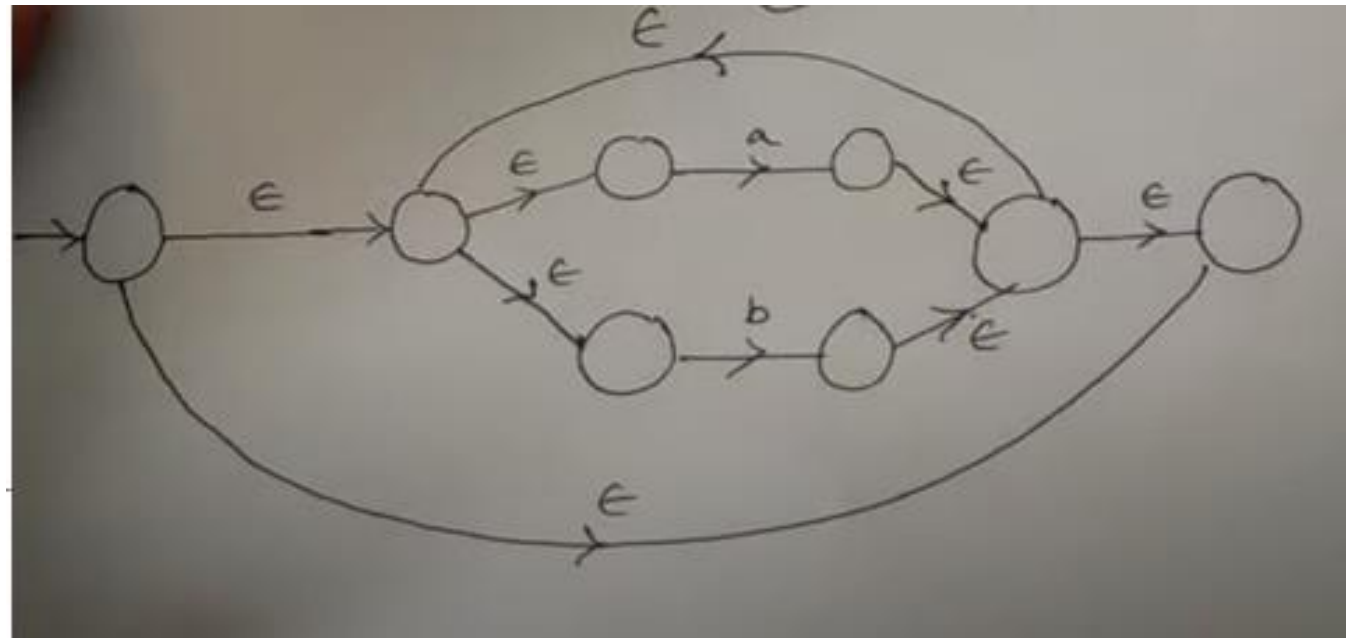


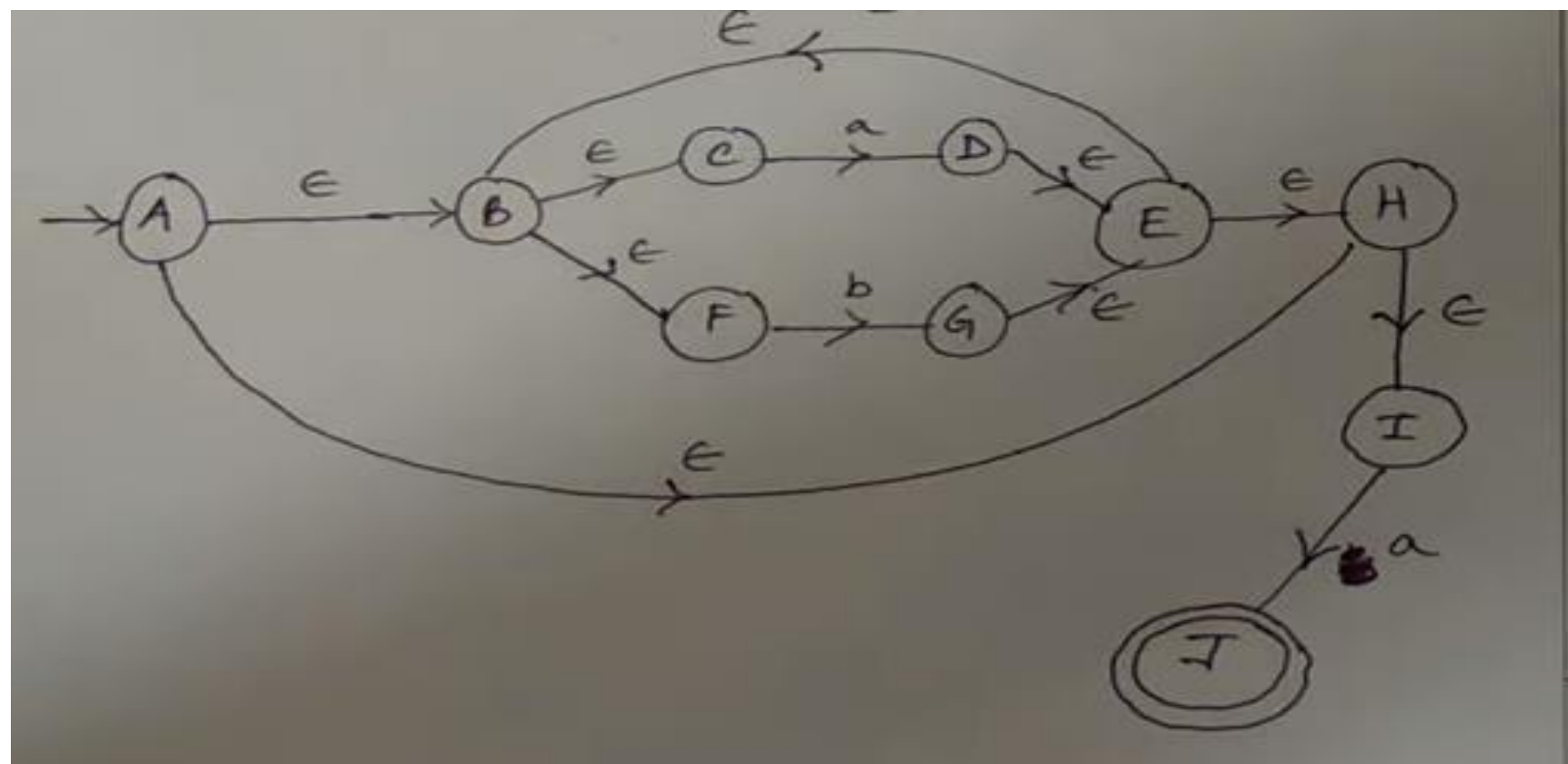
Ques:  $(a + b)^*a$

1



2





# Conversion:

Step 2: Find  $\epsilon$  -closure of all states.

$\epsilon$  -closure (A): {A, B, C, F, H, I}

$\epsilon$  -closure (B): {B, C, F}

$\epsilon$  -closure (C): {C}

$\epsilon$  -closure (D): {D, E, F, B, C, H, I}

$\epsilon$  -closure (E): {E, B, F, C, H, I}

$\epsilon$  -closure (F): {F}

$\epsilon$  -closure (G): {G, E, B, F, C, H, I}

$\epsilon$  -closure (H): {H, I}

$\epsilon$  -closure (I): {I}

$\epsilon$  -closure (J): {J}

**Step 3:** Start with epsilon closure of start state of NFA i.e. Closure of A = {A, B, C, F, H, I}, give a name to this state (here we have named it 1).

**Step 4:** Apply the input symbols and find its epsilon closure.

$$\text{Dtran}[\text{state}, \text{input symbol}] = \epsilon\text{-closure}(\text{move}(\text{state}, \text{input symbol}))$$

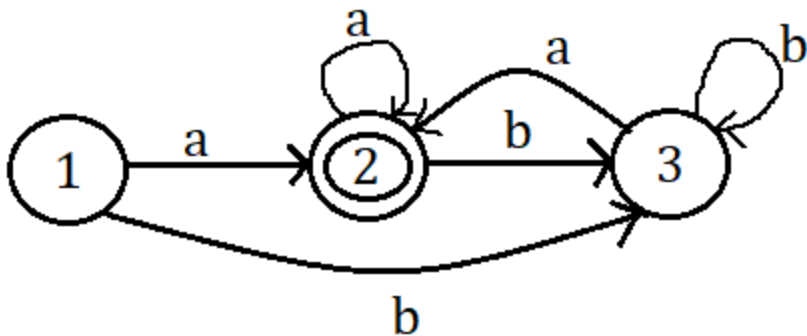
$\text{Dtran}[1, a] = \epsilon\text{-closure}(\text{move}(1, a)) = \epsilon\text{-closure}(D, J) = \{D, E, F, B, C, H, I, J\}$  (this will be termed as state 2 in new table).

$\text{Dtran}[1, b] = \epsilon\text{-closure}(\text{move}(1, b)) = \epsilon\text{-closure}(G) = \{G, E, B, F, C, H, I\}$  (this will be termed as state 3 in new table as this state is not previously received).

$Dtran[2, a] = \epsilon\text{-closure}(\text{move}(2, a)) = \epsilon\text{-closure}(D, J) = \{D, E, F, B, C, H, I, J\}$  (same state as 2).

$Dtran[2, b] = \epsilon\text{-closure}(\text{move}(2, b)) = \epsilon\text{-closure}(G) = \{G, E, B, F, C, H, I\}$  (same state as 3).

	NFA States	DFA State	a	b
Initial state as it contain A →	A, B, C, F, H, I	1	2	3
	D, E, F, B, C, H, I, J	2	2	3
Final state as it contain J ↗	G, E, B, F, C, H, I	3	2	3



## 4. Converting Regular expression directly to DFA

## Direct Method

- Direct method is used to convert given regular expression directly into DFA.
- Uses augmented regular expression  $r\#$ .
- Important states of NFA correspond to positions in regular expression that hold symbols of the alphabet.

Regular expression is represented as syntax tree where interior nodes correspond to operators representing union, concatenation and closure operations.

- Leaf nodes corresponds to the input symbols
- Construct DFA directly from a regular expression by computing the functions  $nullable(n)$ ,  $firstpos(n)$ ,  $lastpos(n)$  and  $followpos(i)$  from the syntax tree.


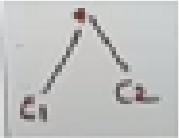
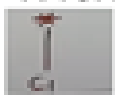
- **nullable** ( $n$ ): The subtree at node  $n$  generates languages including the empty string.
- **firstpos** ( $n$ ): Set of positions that can match the first symbol of a string generated by the subtree at node  $n$ .
- **lastpos** ( $n$ ): Set of positions that can match the last symbol of a string generated by the subtree at node  $n$ .
- **followpos** ( $i$ ): The set of positions that can follow position  $i$  in the tree.



# Steps:

- Step 1: Construct a syntax tree for  $r\#$ .
- Step 2: Traverse the tree to construct functions nullable, firstpos, lastpos and followpos.
- Step 3: Compute followpos.
- Step 4: Convert RE to DFA.

- Rules for computing nullable, firstpos, and lastpos:

Node n	nullable(n)	firstpos(n)	lastpos(n)
n is a leaf node labeled $\epsilon$	true	$\emptyset$	$\emptyset$
n is a leaf node labelled with position i	false	$\{i\}$	$\{i\}$
n is an or node with left child c1 and right child c2 	nullable(c1) or nullable(c2)	firstpos(c1) $\cup$ firstpos(c2)	lastpos(c1) $\cup$ lastpos(c2)
n is a cat node with left child c1 and right child c2 	nullable(c1) and nullable(c2)	If nullable(c1) then firstpos(c1) $\cup$ firstpos(c2) else firstpos(c1)	If nullable(c2) then lastpos(c2) $\cup$ lastpos(c1) else lastpos(c2)
n is a star node with child node c1 	true	firstpos(c1)	lastpos(c1)

## Rules for computing followpos:

1. If  $n$  is a cat-node with left child  $c1$  and right child  $c2$  and  $i$  is a position in  $\text{lastpos}(c1)$ , then all positions in  $\text{firstpos}(c2)$  are in  $\text{followpos}(i)$ .
2. If  $n$  is a star-node and  $i$  is a position in  $\text{lastpos}(n)$ , then all positions in  $\text{firstpos}(n)$  are in  $\text{followpos}(i)$ .

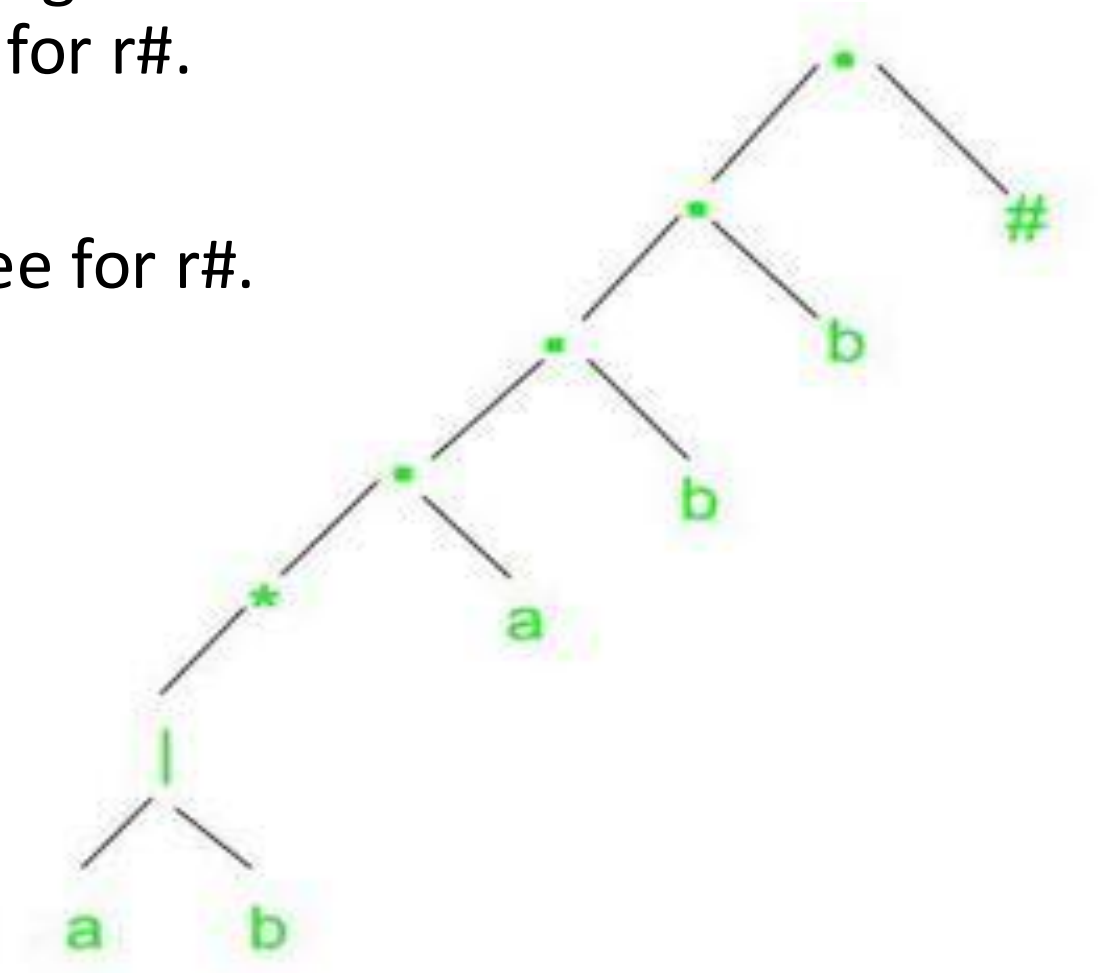
Now that we have seen the rules for computing  $\text{firstpos}$  and  $\text{lastpos}$ , we now proceed to calculate the values of the same for the syntax tree of the given regular expression  $(a|b)^*abb\#$ .

Example:  $r = (a|b)^*abb$

**Step 1:** Firstly, we construct the augmented regular expression for the given expression. By concatenating a unique right-end marker '#' to a regular expression  $r$ , we give the accepting state for  $r$  a transition on '#' making it an important state of the NFA for  $r\#$ .

So,  $r' = (a|b)^*abb\#$

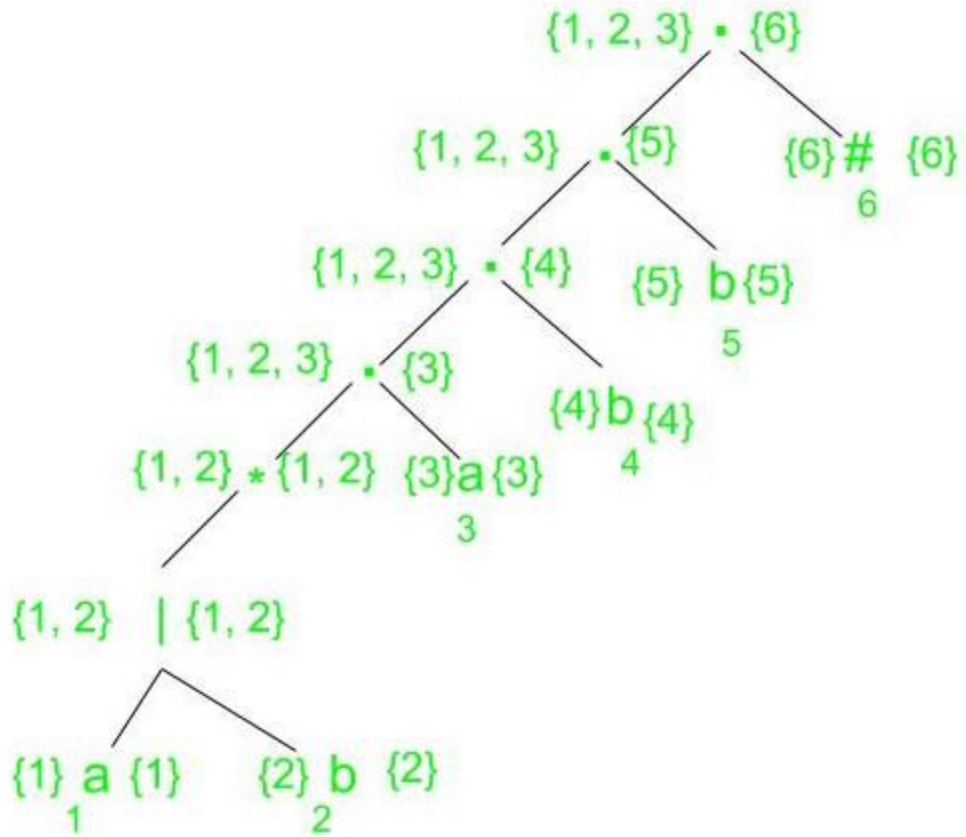
**Step 2:** Then we construct the syntax tree for  $r\#$ .



**Step 3:** Next we need to evaluate four functions nullable, firstpos, lastpos, and followpos.

- nullable( $n$ ) is true for a syntax tree node  $n$  if and only if the regular expression represented by  $n$  has  $\epsilon$  in its language.
- firstpos( $n$ ) gives the set of positions that can match the first symbol of a string generated by the subexpression rooted at  $n$ .
- lastpos( $n$ ) gives the set of positions that can match the last symbol of a string generated by the subexpression rooted at  $n$ .

We refer to an interior node as a cat-node, or-node, or star-node if it is labeled by a concatenation, | or \* operator, respectively.

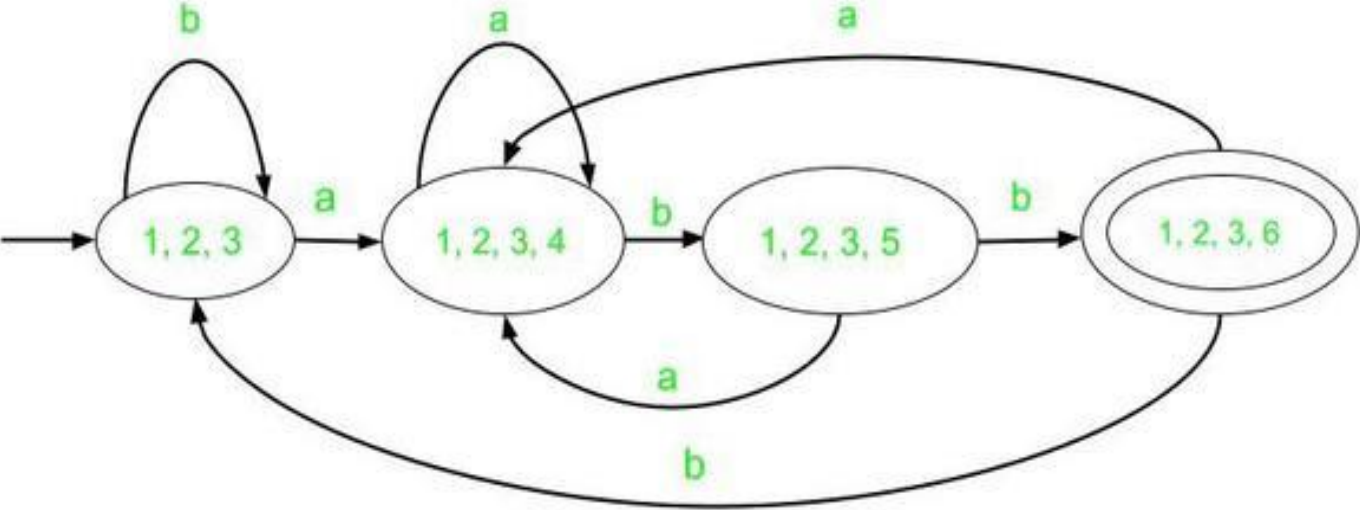


*firstpos and lastpos for nodes in syntax tree for  $(a|b)^*abb\#$*

NODE	followpos
1	$\{1, 2, 3\}$
2	$\{1, 2, 3\}$
3	$\{4\}$
4	$\{5\}$
5	$\{6\}$
6	$\emptyset$

4. Now we construct  $Dstates$ , the set of states of DFA  $D$  and  $Dtran$ , the transition table for  $D$ . The start state of DFA  $D$  is  $firstpos(root)$  and the accepting states are all those containing the position associated with the endmarker symbol  $\#$ .
- According to our example, the  $firstpos$  of the root is  $\{1, 2, 3\}$ . Let this state be  $A$  and consider the input symbol  $a$ . Positions 1 and 3 are for  $a$ , so let  $B = followpos(1) \cup followpos(3) = \{1, 2, 3, 4\}$ . Since this set has not yet been seen, we set  $Dtran[A, a] := B$ .
  - When we consider input  $b$ , we find that out of the positions in  $A$ , only 2 is associated with  $b$ , thus we consider the set  $followpos(2) = \{1, 2, 3\}$ . Since this set has already been seen before, we do not add it to  $Dstates$  but we add the transition  $Dtran[A, b] := A$ .
  - Continuing like this with the rest of the states, we arrive at the below transition table.

- Here, A is the start state and D is the accepting state.
5. Finally we draw the DFA for the above transition table.
- The final DFA will be :**



Input		
State	a	b
----> A	B	A
B	B	C
C	B	D
D	B	A