4.1 Regular Expressions

The language accepted by DFA or NFA and ε-NFA is called regular language. A regular language can be described using regular expressions consisting of the symbols such as alphabets in Σ, the operators such as '.', '+' and '*'. The three operators used to obtain a regular expression are:

- + operator can also be denoted by U) union operator (least precedence)
- (. operator) concatenation operator (next least precedence)
- (* operator) closure operator (highest precedence)

Note: The symbols '(' and ')' can be used in regular expressions. The order of evaluation of the regular expression is determined by the parenthesis and the priority associated with other operators.

Now, let us see **"What is a regular expression?"** A regular expression can be formally defined as follows.

Definition: A regular expression is recursively defined as follows.

- ϕ is a regular expression denoting an empty language.
- ε-(epsilon) is a regular expression indicating the language containing an empty string.
- *a* is a regular expression indicating the language containing only {a}
- If R is a regular expression denoting the language L_R and S is a regular expression denoting the language L_S, then
  - R + S is a regular expression corresponding to the language $L_R \cup L_S$.
  - R.S is a regular expression corresponding to the language $L_R.L_S$.
  - R* is a regular expression corresponding to the language $L_R{}^*$.
- The expressions obtained by applying any of the rules from 1 to 4 are regular expressions.

The following table shows some examples of regular expressions and the language corresponding to these regular expressions

Regular expressions	Meaning
a*	String consisting of any number of a's (or string consisting of zero or more a's).
a+	String consisting of at least one a (or string consisting of one or more a's).
(a+b)	String consisting of either one *a* or one *b*.
(a+b)*	Set of strings of a's and b's of any length including the NULL string.
(a+b)*abb	Set of strings of a's and b's ending with the string abb.
ab(a+b)*	Set of strings of a's and b's starting with the string ab.
(a+b)*aa(a+b)*	Set of strings of a's and b's having a sub string aa.

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 include empty string) followed by any number ofc's(may include empty string).
a+b+c+	Set of string consisting of at least one 'a' followed by string consisting of at least one 'b' followed by string consisting of at least one 'c'.
aa*bb*cc*	Set of s trings consisting of at least one 'a' followed by string consisting of at least one 'b' followed by string consisting of at least one 'c'.
(a+b)* (a + bb)	Set of strings of a's and b's ending with either *a* or *bb*.
(aa)*(bb)*b	Set of strings consisting of even number of a's followed by odd number of b's.
(0+1)*000	Set of strings of 0's and 1's ending with three consecutive zeros(or ending with 000).
(11)*	Set of strings consisting of even number of 1's.
01* + 1	The language represented is the string 1 plus the string consisting of a zero followed by any number of 1's possibly including none.
(01)* + 1	The language consists of a string 1 or strings of (01)'s that repeat zero or more times.
0(1* + 1)	Set of strings consisting of a zero followed by any number of 1's.
(1+ε)(00*1)*0*	Strings of 0's and 1's without any consecutive 1's.
(0+10)*1*	Strings of 0's and 1's ending with any number of 1's (possibly none).
(a+b)(a+b)	Strings of a's and b's whose length is 2.

Example 4.1: Obtain a regular expression representing strings of a's and b's having length 2

Solution: Strings of a's and b's with length two are: aa, ab, ba and bb.

So, RE is: (aa + ab + ba + bb)

or

So, R.E is (a+b)(a+b)

Example 4.2: Obtain regular expression representing strings of a's and b's of length ≤ 2

The strings of a's and b's whose length is ≤ 2 can be written as shown below:

$$\varepsilon + a + b + aa + ab + ba + bb$$

The above strings can be obtained using the regular expression:

$$(\varepsilon + a + b)\ (\varepsilon + a + b)$$

Using short hand notation, the above R.E can be written as:

$$(\varepsilon + a + b)^2$$

So, R.E is $(\varepsilon + a + b)^2$

Example 4.3: Obtain regular expression representing strings of a's and b's of length ≤ 10

By using the previous problem, we can write the expression as:

$$(\varepsilon + a + b)^{10}$$

So, R.E is $(\varepsilon + a + b)^{10}$

Example 4.4: Obtain a regular expression representing strings of a's and b's having even length

Solution: The regular expression is obtained by having zero or more combinations of strings aa, ab, ba, and bb.

So, RE is: $(aa + ab + ba + bb)^*$

Note: The above regular expression can also be written as:

So, R.E is $((a+b)(a+b))^*$

Example 4.5: Obtain a regular expression representing strings of a's and b's having odd length

Solution: The regular expression is obtained by prefixing or suffixing a or b denoted by (a+b) to the regular expression shown in previous problem as shown below:

So, RE is: $(aa + ab + ba + bb)^* (a+b)$

or

$(a+b)\ (aa + ab + ba + bb)^*$

Example 4.6: Obtain a regular expression representing a language consisting of strings of a's and b's with alternate a's and b's or no two consecutive letters are same.

Solution: The alternate a's and b's can be obtained by "concatenating the string ab zero or more times" which can be represented by the regular expression

$(ab)^*$

and adding an optional *b* to the front and adding an optional *a* at the end as shown below:

So, R.E is $(\varepsilon + b)\ (ab)^*\ (\varepsilon + a)$

or

R.E is $(\varepsilon + a)\ (ba)^*\ (\varepsilon + b)$ interchanging *a* and *b*

Example 4.7: Obtain a regular expression representing a language consisting of strings of 0's and 1's with at most one pair of consecutive 0's

Solution: Observe that the string may have any number of 1's with no 0's or with one 0 or with two 0's.

The string containing at most two 0's will have the following three cases:

- **Case 1:** No zeroes which may represent any number of 1's and given by the RE:

1^*

- **Case 2:** No two consecutive 0's can be obtained by any combination of 1's and 01's as shown below:

$$(1+01)^*$$

We can have an optional 0 at the end of the above regular expression and can be written as:

$$(1 + 01)^* (\varepsilon + 0)$$

- Two consecutive 0's preceded by any combination of 1's and 01's and followed by combination of 1's and 10's is given by the RE:

$$(1+01)^*00(1+10)^*$$

So, the complete regular expression is obtained by adding all the above regular expressions:

So, R.E = $1^* + (1+01)^* (\varepsilon + 0) + (1+01)^*00(1+10)^*$

Example 4.8: Obtain a regular expression representing a language containing at least one a and at least one b where $\Sigma = \{a, b, c\}$

Solution: Since the string should contain at least one *a* and at least one *b*, the regular expression for this minimum string can be written as:

RE: ab + ba

To the above expression, we can introduce any combinations of a's, b's and c's in all possible places. By introducing (a+b+c)* in all possible places for the string ab we obtain following REs:

(a+b+c)*a (a+b+c)* b (a+b+c)*

By introducing (a+b+c)* in all possible places for the string *ba* we obtain following REs:

(a+b+c)*b (a+b+c)* a (a+b+c)*

The final regular expression is obtained by adding above two REs.

So, R.E = (a+b+c)*a (a+b+c)* b (a+b+c)* + (a+b+c)*b (a+b+c)* a (a+b+c)*

Example 4.9: Obtain a regular expression for the language L(R) = {w | w ∈ {0, 1}* with at least three consecutive 0's

Solution: The regular expression representing 3 consecutive zero's is shown below:

000

The string 000 can be preceded by or followed by an arbitrary string consisting of 0's and 1's can be represented by the regular expression:

$$(0+1)^*$$

So, the regular expression can be written as

$$(0+1)^* 000 (0+1)^*$$

So, R.E is $(0+1)^* 000 (0+1)^*$

Note: The language corresponding to the regular expression can be written as

$L(R) = \{ (0+1)^m000(0+1)^n \mid m \geq 0 \text{ and } n \geq 0\}$

Example 4.10: Obtain a regular expression representing strings of a's and b's ending with 'b' and has no substring aa

Solution: The statement "strings of a's and b's ending with 'b' and has no substring aa" can be restated as "string made up of any combinations of either b or ab without ε".

So, R.E is (b+ab) (b+ab)*

Note: The language corresponding to the regular expression can be written as

$$L(R) = \{ (b + ab)^n \mid n \geq 1 \}$$

Example 4.11: Obtain a regular expression representing strings of 0's and 1's having no two consecutive zeros.

Solution: No two consecutive zeroes can be obtained using:

$$(01)^*$$

The above expression can be preceded by an optional 1 and followed by an optional zero as shown below:

$$(1 + \varepsilon)\ (01)^*\ (0 + \varepsilon)$$

The regular expression can also be written by interchanging 0's and 1's in the above regular expression and can be written as shown below:

$$(0 + \varepsilon)\ (10)^*\ (1 + \varepsilon)$$

The regular expression also can be written as shown below:

So, R.E is $(1 + 01)^*\ (0 + \varepsilon)$

Example 4.12: Obtain a regular expression representing string of a's and b's starting with 'a' and ending with 'b'.

Solution: Strings of a's and b's of arbitrary length can be written as

$$(a + b)^*$$

But, to get the string starting with 'a' and end with 'b' we have to precede the above RE with *a* and suffix the above RE with *b*.

So, R.E is a $(a + b)^*$ b

Example 4.13: Obtain a regular expression representing string of a's and b's whose second symbol from the right end is a.

Solution: The first symbol from the right end can be a or b which can be represented by the R.E

$$(a + b)$$

The second symbol from the right end should be a which can be represented by the RE

$$a$$

But, the third symbol from the right can start with any combinations of a's and b's denoted by the RE:

$$(a+b)^*$$

So, R.E is $(a + b)^* a (a+b)$

Note: On similar lines, strings of a's and b's whose third symbol from the right end is a can be written as

So, R.E is $(a + b)^* a(a+b) (a+b)$

Example 4.14: Obtain a regular expression representing string of a's and b's whose tenth symbol from the right end is a.

Solution: On similar lines to the previous problem, the RE representing strings of a's and b's whose tenth symbol from the right end is a can be written as shown below:

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

Since there are *nine* (a+b) expressions to the right of *a*, the above expression using short hand notation can be written as shown below:

So, R.E = $(a + b)^* a(a+b)^9$

Note: It is clear from this expression that all strings must be of length 10 or more. Here, we need to track the last 10 characters.

Example 4.15: Obtain a regular expression representing strings of a's and b's such that third symbol from the right is a and fourth symbol from the right is *b*

Solution: We are interested only in the third and fourth symbol so that third symbol from the right end is *a* and fourth symbol from the right end is *b* and the regular expression for this is given by

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

But, this substring can be preceded by any combinations of a's and b's which is given by the regular expression

$$(a + b)^*$$

By concatenating the above two regular expressions, we can write the regular expression.

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

Example 4.16: Obtain a regular expression representing the words with two or more letters but beginning and ending with the same letter where $\Sigma = \{a, b\}$

Solution: The string consisting of a's and b's can be denoted by the regular expression

$$(a + b)^*$$

Since, the string should start and end with the same letter, the above regular expression can be preceded and followed by *a* as shown below:

$$RE\ 1 = a(a+b)^*a$$

or preceded by and followed by *b* as shown below:

$$RE\ 2 = b(a+b)^*b$$

So, the final regular expression with two or more letters but beginning and ending with the same letter where $\Sigma = \{a, b\}$ can be written as:

$$\text{So, R.E} = a(a+b)^*a + b(a+b)^*b$$

Example 4.17: Obtain a regular expression representing strings of a's and b's whose length is either even or multiples of 3 or both.

Solution: The regular expression whose length is even can be obtained using

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

and the regular expression whose length is multiples of 3 can be obtained using

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

Thus, the regular expression whose length is either even or multiples of 3 or both is obtained by taking various combinations of strings obtained using the above regular expressions.

$$\text{So, R.E} = ((a + b) (a + b))^* + ((a + b) (a + b) (a + b))^*$$

Example 4.18: Obtain a regular expression representing strings of a's and b's such that every block of four consecutive symbols contains at least two a's.

Solution: It is required to have a block of 4 symbols with at least two a's which can be represented by the RE:

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

or

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

or

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

or

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

or

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

or

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

We can have any combinations of strings obtained from above REs. But, it is not possible to have star operator * (which indicates zero or more symbols) since it encompasses even the empty string. So, we have to use the + operator (which indicates one or more symbols) and can be expressed as shown below:

$$\text{So, RE} = (aa(a+b)(a+b) + a(a+b)a(a+b) + a(a+b)(a+b)a + (a+b)aa(a+b) + (a+b)a(a+b)a + (a+b)(a+b)aa)^+$$

Example 4.19: Obtain a regular expression for the language $L = \{anbm \mid m + n \text{ is even}\}$ (3 marks VTU July 2007)

Solution: It is given that

- n represent number of a's
- m represent number of b's

It is also given that m + n is even. This results in following two cases:

Case 1: (m + n is even) Even number of a's followed by even number of b's results in even number of symbols which is given by the regular expression:

RE = (aa)* (bb)*

Case 2: (m + n is even) Odd number of a's followed by odd number of b's results in even number of symbols which is given by the regular expression:

RE = a(aa)* b(bb)*

The final regular expression is obtained by adding the above two regular expressions.

Example 4.20: Obtain a regular expression for the L = $\{a^n b^m \mid m \geq 1, n \geq 1, nm \geq 3\}$
(3 marks VTU July 2007)

Solution: The possible cases to satisfy the given relations are shown below:

Case 1: Since nm ≥ 3, if m = 1 then n ≥ 3. The equivalent RE is given by:

RE = aaaa* b

Case 2: Since nm ≥ 3, if n = 1 then m ≥ 3. The equivalent RE is given by:

RE = a bbbb*

Case 3: Since nm ≥ 3, if m ≥ 2 then n ≥ 2. The equivalent RE is given by:

RE = aaa* bbb*

So, the final regular expression is obtained by adding all the above regular expressions.

So, R.E = aaaa* b + a bbbb* + aaa* bbb*

Example 4.21: Obtain a regular expression for the language L = $\{a^{2n} b^{2m} \mid n \geq 0, m \geq 0\}$
5 marks (VTU Jan/Feb 2005)

Solution: For every n ≥ 0, a^{2n} results in even number of a's and for every m ≥ 0, b^{2m} results in even number of b's. The regular expression representing even number of a's is given by:

RE = (aa)*

The regular expression representing even number of b's is given by:

RE = (bb)*

So, regular expression corresponding to L = $\{a^{2n} b2^m \mid n \geq 0$ and $m \geq 0\}$ is obtained by concatenating the above two regular expressions.

So, R.E = (aa)* (bb)*

Example 4.22: Obtain a regular expression for strings of a's and b's containing not more than three a's

Solution: Strings of a's and b's containing not more than three a's results in following cas

- ε indicating zero a's

  or

- one a

  or

- two a's

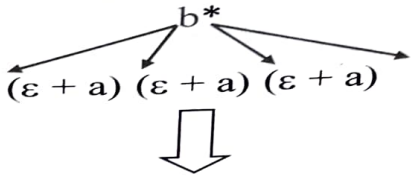
  or

- three a's

Note: At any point of time, number of a's should never exceed three a's. The regular expressi for all the above strings can be written as:

RE: (ε + a) (ε + a) (ε + a)

Now, we can insert any number of b's represented by b* at various positions in above expressio as shown below:



$$\text{So, R.E} = b^*(\varepsilon + a)\, b^*(\varepsilon + a)\, b^*(\varepsilon + a)b^*$$

Example 4.23: Obtain a regular expression for the language $L = \{a^n b^m : n \geq 4 , m \leq 3\}$

Solution: The given language consists of n number of a's followed by m number of b's where $n \geq 4$ and $m \leq 3$. This language can be interpreted as concatenation of two languages:

- The first language consists of strings of four 4's followed by any number of a's (since n ≥ 4). This is represented by the regular expression:

  RE = aaaa(a)* ...(1)

- The second language consist of strings of at most three b's i.e., it may have zero b's or one b, or two b's or three b's but not more than three b's. This can be represented by the regular expression:

  RE = (ε + b) (ε + b) (ε + b) ...(2)

So, the final regular expression is obtained by concatenating RE shown in (1) and RE shown in (2).

$$\text{So, R.E} = aaaa(a)^* (\varepsilon + b) (\varepsilon + b) (\varepsilon + b)$$

Example 4.24: Obtain a regular expression for strings of a's and b's whose lengths are multiples of 3

Solution: The strings of a's and b's whose length is 3 is given by the following regular expression:

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

The regular expression for strings of a's and b's whose lengths are multiples of 3 can be obtained by using the * operator.

So, R.E = ((a + b) (a + b) (a + b))*

Example 4.25: Obtain a regular expression for the language L = {w : |w| mod 3 = 0 where w ∈ (a,b)*} 5 marks (VTU Jan/Feb 2005) 3 marks VTU July 2006)

Solution: It is given that the language consists of strings whose length is multiple of 3 which can be written as shown below:

L = { ε, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaaaa, aaaaab, etc. }

The strings of a's and b's whose length is 3 is given by the following regular expression:

RE: (a + b) (a + b) (a + b)

Note that minimum string has to be ε whose length is 0 which is divisible by 3. The string whose length is multiples of 3 can be obtained using * operator for the above regular expression.

So, R.E = ((a + b) (a + b) (a + b))*

Example 4.26: Obtain a regular expression for the language L = {w : n_a(w) mod 3 = 0 where w ∈ (a,b)*}

Solution: Observe that even though the string is made up of a's and b's we are interested in only number of a's. The number of a's in the string should be divisible by 3. So, number of a's can be 0, 3, 6 and so on.

Note: There is no restriction on the number of b's. So, any number of b's denoted by b* can be inserted. The minimum string that correspond to the language is:

aaa

By incorporating any number of b's denoted by b* in all possible places in the above expression, we have the following regular expression:

RE: b*ab*ab*ab*

To incorporate ε in the above regular expression, we can use * operator.

So, R.E = (b*ab*ab*ab*)*

Example 4.27: Obtain a regular expression for set of all strings that do not end with 01, over {0, 1}*

Solution: It is given that the string should not end with 01. So, other strings whose length is 2 and that do not end with 01 are 00, 10 and 11. These strings can be preceded by any combinations of 0's and 1's denoted by (0 + 1)* and end with 00 or 10 or 11.

So, R.E = (0 + 1)* (00 + 10 + 11)

Example 4.28: Obtain a regular expression for $L = \{vuv : u, v \in \{a, b\}^* \text{ and } |v| = 2\}$

Solution: Since $|v| = 2$, $v \in \{a, b\}^*$ then $v = \{aa, ab, ba, bb\}$

Since there is no restriction on string *u*, the string *u* can be any combinations of a's and b's and is represented by the following regular expression:

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

So, the final regular expression to accept the language:

$$L = \{vuv : u, v \in \{a,b\}^* \text{ and } |v| = 2\}$$

is obtained by concatenating the regular expressions v, u and v as shown below:

$$aa(a+b)^*aa + ab(a+b)^*ab + ba(a+b)^*ba + bb(a+b)^*bb$$

Example 4.29: Give regular expressions for the following languages on $\Sigma = \{a, b, c\}$

(a) All strings containing exactly one a

(b) All strings containing no more than three a's

(c) All strings that contain at least one occurrence of each symbol in Σ

Solution: (All strings containing exactly one a) It is given that strings of a's and b's containing exactly one a has to be represented. So, the regular expression to represent one *a* is given by:

$$RE = a$$

But, one *a* can be preceded and followed by any combinations of b's and c's which is represented by the regular expression:

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

$$\text{So, R.E} = (b + c)^* \, a \, (b + c)^*$$

(All strings containing no more than three a's) This indicates that string may consist of no a's or string may have one a or string may have two a's or string may have three a's which is represented by the regular expression:

$$(\varepsilon + a)\ (\varepsilon + a)\ (\varepsilon + a)$$

Since string may consists of any combinations of b's and c's which is represented by the regular expression:

$$(b + c)^*$$

We can insert $(b + c)^*$ in various positions of $(\varepsilon + a)\ (\varepsilon + a)\ (\varepsilon + a)$ to get the resultant expression:

$$\text{So, R.E} = (b + c)^* \, (\varepsilon + a) \, (b + c)^* \, (\varepsilon + a) \, (b + c)^* \, (\varepsilon + a) \, (b + c)^*$$

(All strings that contain at least one occurrence of each symbol in Σ) Since $\Sigma = \{a, b, c\}$ and the string should consist of at least one occurrence of each symbol, the various strings which satisfy the condition are:

abc, acb, bac, bca, cab, cba

The regular expression corresponding to the above strings can be written as:

abc +

acb +

bac +

bca +

cab +

cba

But, we can incorporate any combinations of a's, b's and c's represented by (a + b + c)* in various positions in the above regular expression:

(a+b+c)* a (a+b+c)* b (a+b+c)* c (a+b+c)* +

(a+b+c)* a (a+b+c)* c (a+b+c)* b (a+b+c)* +

(a+b+c)* b (a+b+c)* a (a+b+c)* c (a+b+c)* +

(a+b+c)* b (a+b+c)* c (a+b+c)* a (a+b+c)* +

(a+b+c)* c (a+b+c)* a (a+b+c)* b (a+b+c)* +

(a+b+c)* c (a+b+c)* b (a+b+c)* a (a+b+c)*

Example 4.30: Obtain a regular expression for the language L = {w: the string ends with ab or ba where w ∈ {a, b}* }

Solution: The regular expression corresponding to the string ending with ab or ba can be written as:

RE = ab + ba

But, the strings of a's and b's denoted by the regular expression (a+b)* ending with ab or ba can be written as:

RE = (a+b)*(ab+ba)

Note: Now, let us see **"What are precedence of regular expression operators?"** In all above regular expressions that we have discussed so far have the operators +, * and concatenation operator with the following precedence of operators.

- The star operator called Kleene star (denoted by *) has the highest precedence.
- The concatenation operator (also called dot operator) has the next highest precedence.
- The union operator (denoted by + symbol) has the least precedence. The symbol + can also be replaced by union operator U.

Note: The precedence of all the above operators can be changed using parentheses. For example, (a+b)a* can also be written as (a U b)a*. In this expression.

- (a+b) or (a U b) has the highest precedence because of parentheses.
- Next preference is given for * operator.
- Next preference is given for concatenation operator.

Kleen's theorem Basics

K → intermediate nodes

K = 0

→ There are no intermediate nodes exist between i and j states

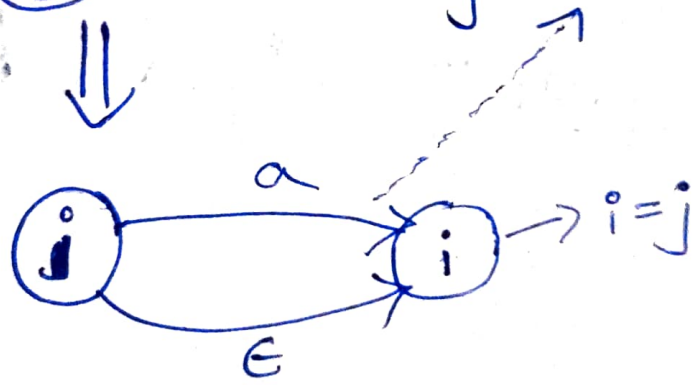
(i) i = j

(1) 

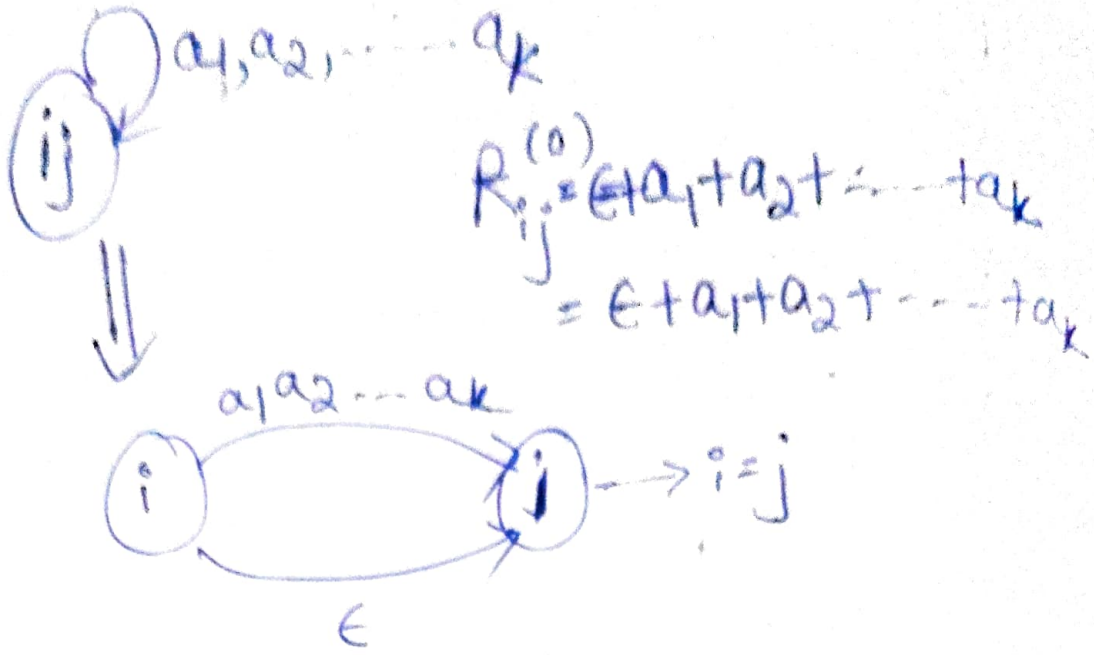
$R_{ij}^{(0)} = \epsilon$

(2) 

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



Every state reads 'ϵ' (null string or empty string) reaches its own state itself by default!

(3) 

(ii) $\underline{K = 0}$

$i \neq j$

(1) (i) (j) $R_{ij}^{(0)} = \emptyset$ (no path)

no path exist between i and j states.

(2) (i) —a→ (j) $R_{ij}^{(0)} = a$

path exists with symbol 'a' from i to j.

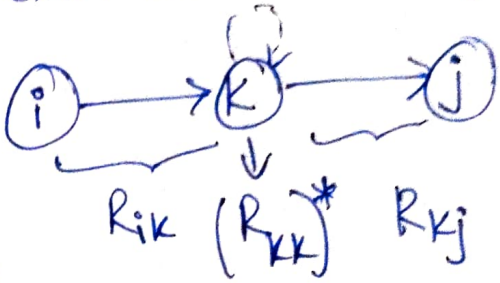
(3) (i) —a1, a2 ... ak→ (j) $R_{ij}^{(0)} = a_1 + a_2 + \cdots + a_k$

(iii)

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

Direct path between i and j

K as a intermediate node exist between i and j,



Rules of Regular expressions

(1) $\epsilon . R = R . \epsilon = R$

(2) $\phi . R = R . \phi = \phi$

(3) $(\epsilon)^* = \epsilon$

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

(5) $\phi + R = R$

(6) $R + R = R$

(7) $R . R^* = R^* . R = R^+$

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

(9) $\epsilon + R . R^* = \epsilon + R^* R = R^*$

(10) $(R+\epsilon) R^* = R^* (R+\epsilon) = R^*$

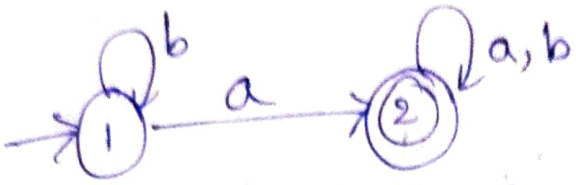
(11) $(R+\epsilon)(R+\epsilon)^* (R+\epsilon) = R^*$

(12) $R^* S + S = R^* S$

(13) $\phi + \epsilon = \epsilon$

Conversion of Finite Automata to Regular expression (Kleen's Theorem)

(1)



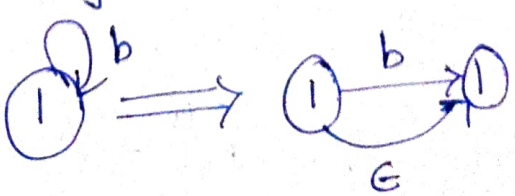
we need to convert the given finite Automata to Regular expression using Kleen's theorem. To get regular expression, we have to calculate, $R_{12}^{(2)} = ?$ → no. of intermediate nodes

	initial state K=0	K=1	K=2	← K-intermediate node.
$R_{11}^{(k)}$	$R_{11}^{(0)}$	$R_{11}^{(1)}$	$R_{11}^{(2)}$	
$R_{12}^{(k)}$	$R_{12}^{(0)}$	$R_{12}^{(1)}$	$R_{12}^{(2)}$	
$R_{21}^{(k)}$	$R_{21}^{(0)}$	$R_{21}^{(1)}$	$R_{21}^{(2)}$	
$R_{22}^{(k)}$	$R_{22}^{(0)}$	$R_{22}^{(1)}$	$R_{22}^{(2)}$	

K=0

(i) $R_{11}^{(0)\to k}$ => i=1, j=1, k=0

i & j



=> $b + \epsilon$

(Arrow marks are shown in the same direction, so '+' operator is used).

So, $\boxed{R_{11}^{(0)} = b+\epsilon}$ ⟶ (I)

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

(1) —a→ (2)

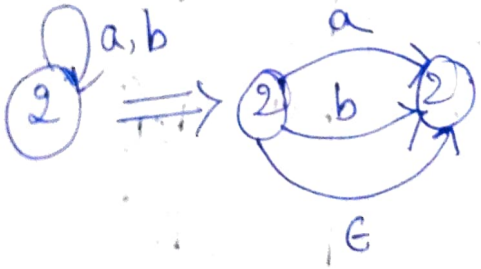
$\boxed{R_{12}^{(0)} = a}$ ⟶ (II)

(iii) $R_{21}^{(0)}$

(2) (1)

no path exists.. So, $\boxed{R_{21}^{(0)} = \phi}$ ⟶ (III)

(iv) $R_{22}^{(0)}$



$\Rightarrow \epsilon + a + b$

$\boxed{R_{22}^{(0)} = \epsilon + a + b}$ ⟶ (IV)

$k=0$,

$R_{11}^{(0)} = b+\epsilon$, $R_{21}^{(0)} = \phi$

$R_{12}^{(0)} = a$, $R_{22}^{(0)} = \epsilon + a + b$

$K=1$

(i) $R_{11}^{(1)}$

Apply the formula,

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

substitute, $i=1, j=1, k=1$

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

using (I)

$$= (\epsilon+b) + (\epsilon+b)\,(\epsilon+b)^*(\epsilon+b) \longrightarrow (V)$$

$$= (\epsilon + b) + \underbrace{(\epsilon + b)}_{R}\ \underbrace{(\epsilon + b)^*}_{R^*}(\epsilon + b)$$

$$\boxed{RR^* = R^+}$$

$$= (\epsilon + b) + (\epsilon + b)^+ (\epsilon + b)$$

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

$$\Downarrow$$

$$(\epsilon + b)^+ = \{\epsilon, b, \epsilon\epsilon, \epsilon b, b\epsilon, bb, \ldots\}$$

$$= \{\epsilon, b, bb, \ldots\}$$

$$= b^*$$

$$= (\epsilon + b) + b^* (\epsilon + b)$$

$$= (\epsilon + b) + b^*\epsilon + b^* b \qquad L(b^* b) \subseteq L(b^*)$$

$$= (\epsilon + b) + \underbrace{b^*}_{R} + \underbrace{b^*}_{R}$$

$$\boxed{R + R = R}$$

$$= (\epsilon + b) + b^*$$

$$= \epsilon + b + (\epsilon + b + bb + bbb + \ldots)$$

$$\boxed{R_{11}^{(1)} = b^*}$$

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

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

$$= a + (\epsilon + b)^+ a \quad [\because RR^* = R^+]$$

$$= a + (\epsilon + b + \epsilon.\epsilon + \epsilon b + b.\epsilon + bb + \cdots) a$$

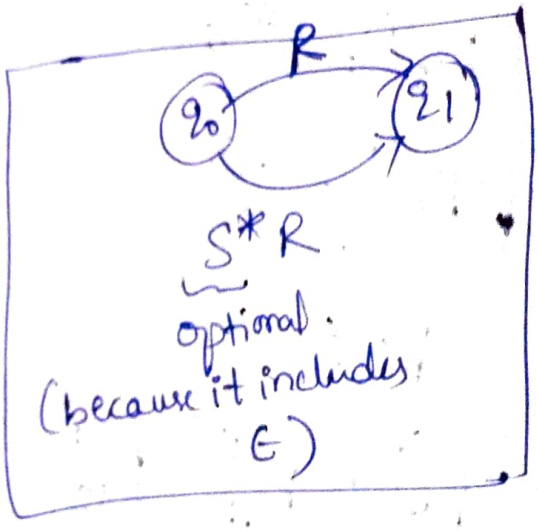
$$= a + (\epsilon + b + bb + \cdots) a$$

$$= a + b^* a$$

$$\boxed{R_{12}^{(1)} = b^* a}$$

$[\because R + S^* R = S^* R]$

$R \subseteq S^* R$



$$R_{21}^{(1)} = R_{21}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{11}^{(0)}$$

$$= \phi + \phi (\epsilon + b)^* (\epsilon + b)$$

$$= \phi + \phi \quad [\because \phi . R = \phi]$$

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

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

$$= (\epsilon + a + b) + \phi\ (\epsilon + b)^* . a$$

$$= (\epsilon + a + b) + \phi \quad [\because \phi . R = \phi]$$

$$\boxed{R_{22}^{(1)} = (\epsilon + a + b)}$$

K = 1

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

$$R_{12}^{(1)} = b^* a$$

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

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

K = 2

$$R_{11}^{(2)} = R_{11}^{(1)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{21}^{(1)}$$

$$= b^* + b^* a (\epsilon + a + b)^* \phi$$

$$= b^* + \phi \quad [\because \phi . R = \phi]$$

$$\boxed{R_{11}^{(2)} = b^*}$$

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

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

$$= b^*a + b^*a\,(\epsilon + a+b)^+ \quad [\because R R^* = R^+]$$

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

$$= b^*a + b^*a\,(\epsilon + a+b+\epsilon a+\epsilon b+a\epsilon+b\epsilon+ a\,b+b\,a+\epsilon\epsilon+aa+bb+\ldots)$$

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

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

$$= R + R\,S^*$$

$$\boxed{R_{12}^{(2)} = b^*a\,(a+b)^*}$$

$$L(b^*a) \subseteq L(b^*a\,(a+b)^*)$$

$$R_{22}^{(2)} = R_{22}^{(1)} + R_{22}^{(1)} \left(R_{22}^{(1)}\right)^* R_{22}^{(1)}$$

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

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

$$+ (\epsilon + a+b+aa+ab+ba+\ldots)(\epsilon + a+b)$$

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

$$\because L(\epsilon + a+b) \subseteq L(a+b)^*$$

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

$$\boxed{R_{22}^{(2)} = (a+b)^*}$$

$k=2$,

$R_{11}^{(2)} = b^*$ $\quad$ $R_{21}^{(2)} = \phi$

$R_{12}^{(2)} = b^* a (a+b)^*$ $\quad$ $R_{22}^{(2)} = (a+b)^*$

$$R_{21}^{(2)} = R_{21}^{(1)} + R_{22}^{(1)} (R_{22}^{(1)})^* R_{21}^{(1)}$$

$$= \phi + (\epsilon + a+b)(\epsilon + a+b)^* \phi$$

$$= \phi + \phi \qquad [\because \phi \cdot R = \phi]$$

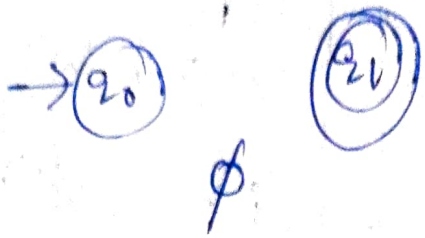
$$\boxed{R_{21}^{(2)} = \phi}$$

So, Final Regular expression,

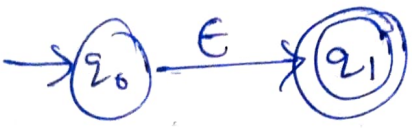
$$\boxed{R_{12}^{(2)} = b^* a (a+b)^*}$$

Conversion from Regular expression to Finite Automata

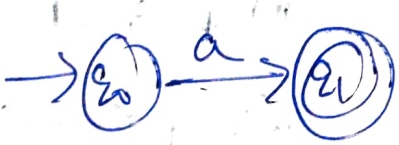
(i) Regular expression = ϕ



(ii) Regular expression = ϵ

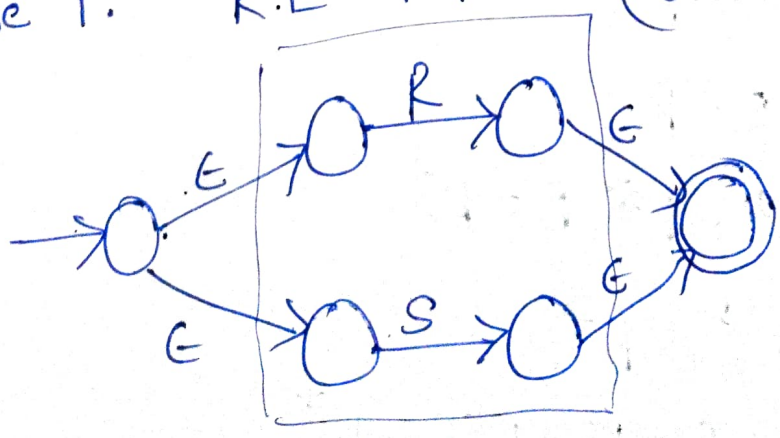


(iii) Regular expression = a

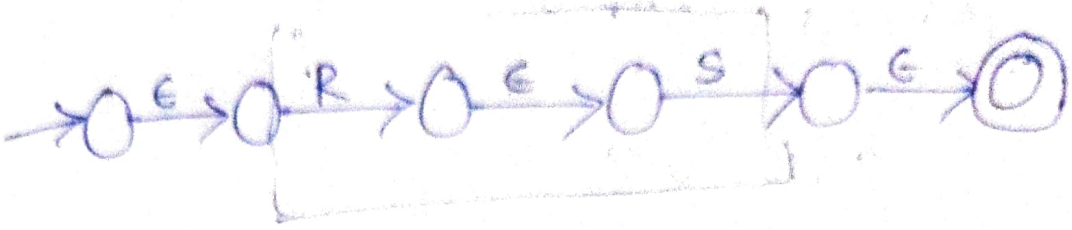


Let R is a R.E for given language L_1 then
S is a R.E for given language L_2.

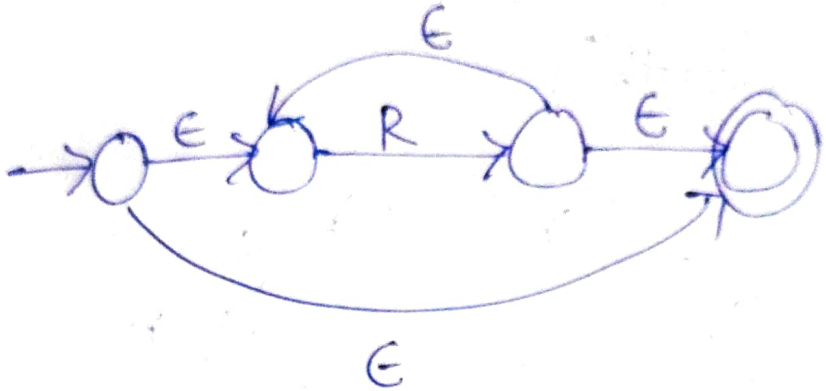
Case 1: R.E = R+S (Union)



Case 2: R.E = R.S (Concatenation)

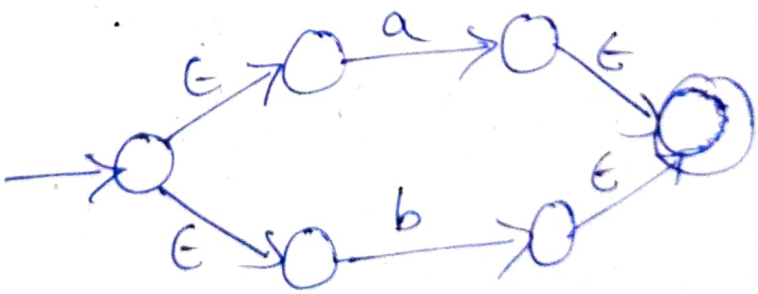


Case 3: R.E = R*

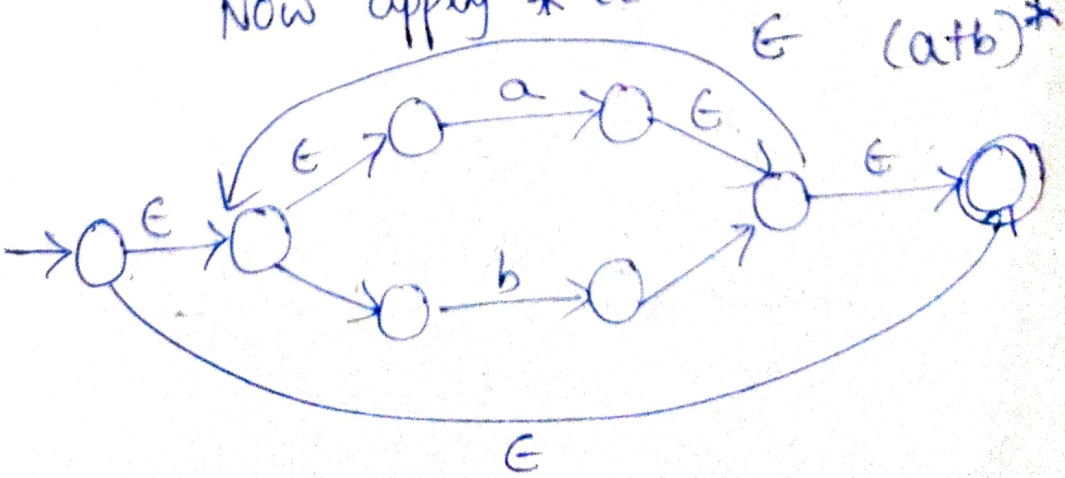


(1) Convert (a+b)* to Finite Automata

Sol^{n}:- First apply Union (a+b).

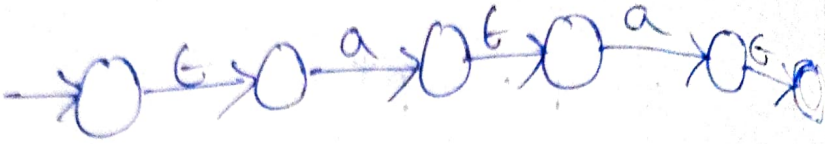


Now apply * (closure),

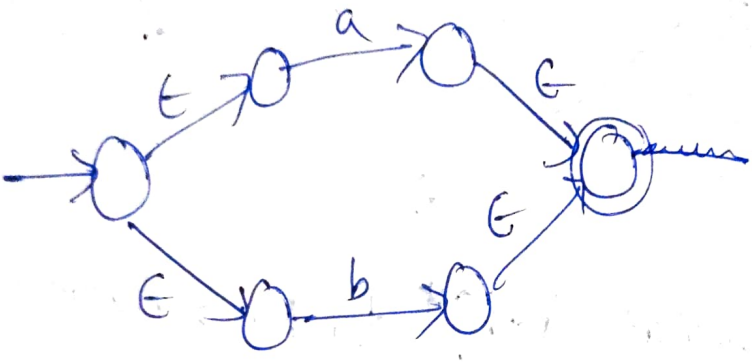


(2) aa(a+b)

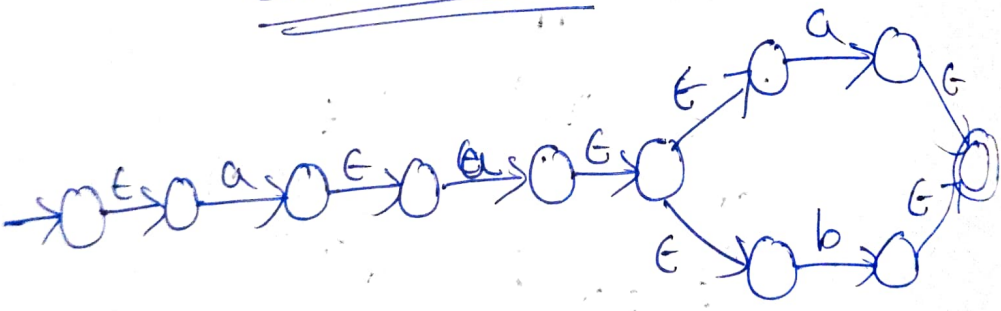
First, aa



(a+b)

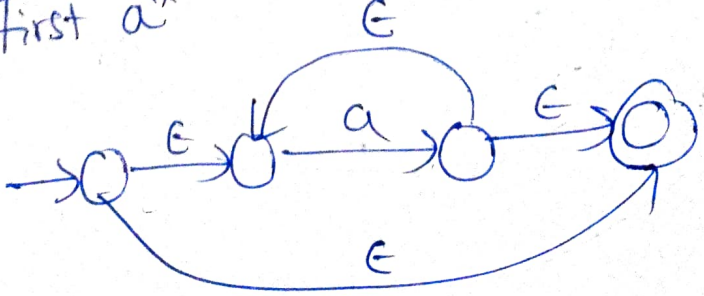


Combine both

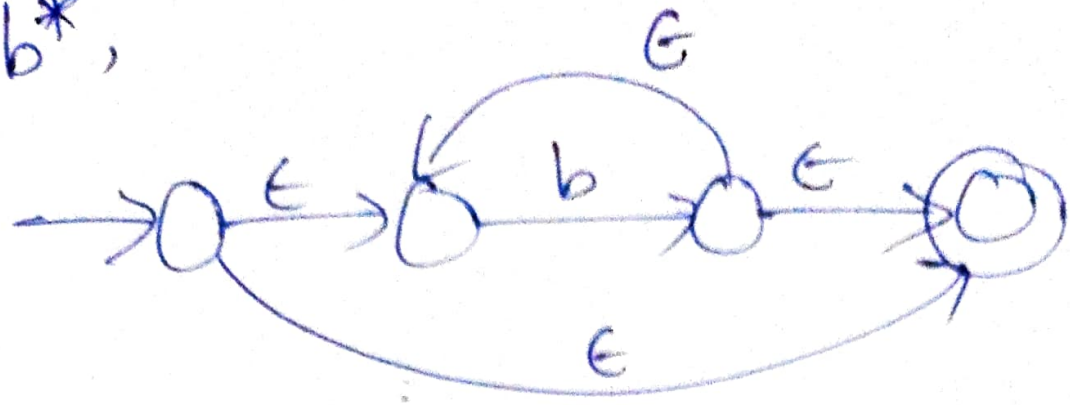


(3) $a^* + b^* + c^*$

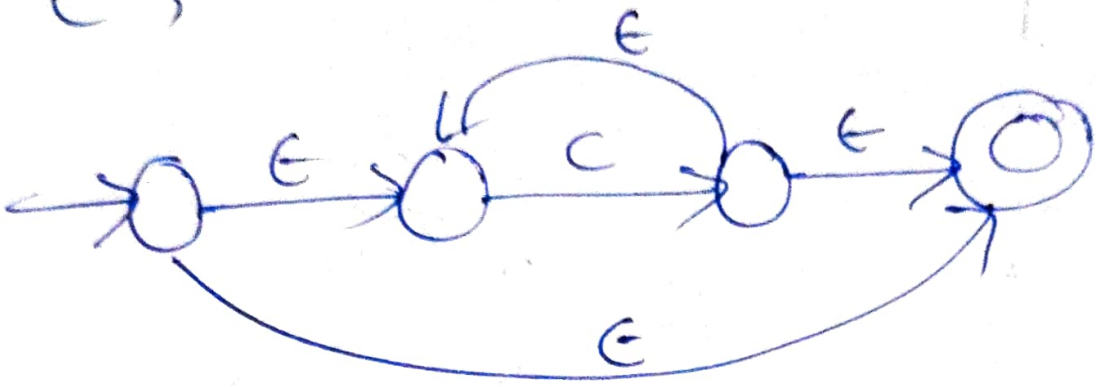
First a^*



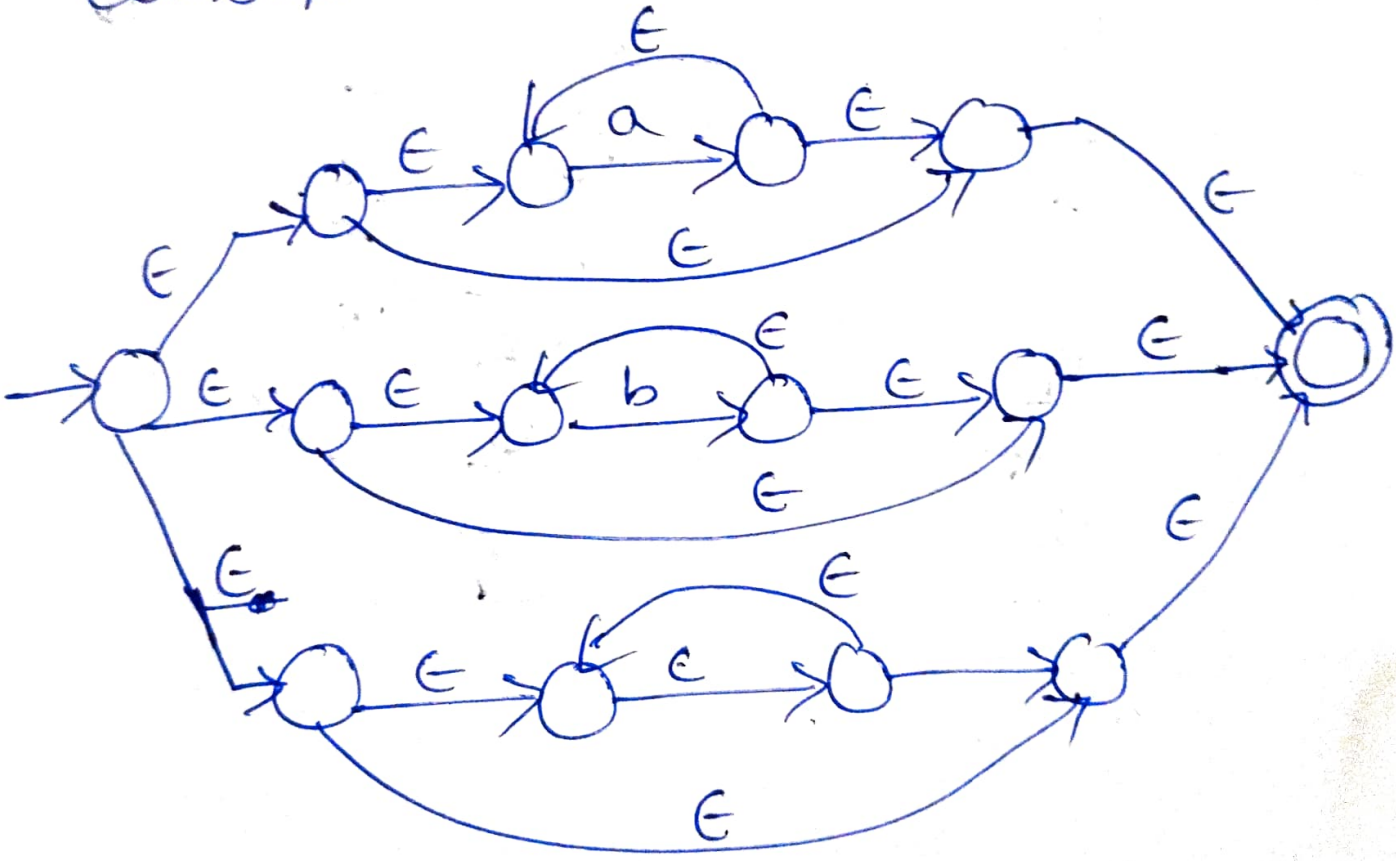
b^*,



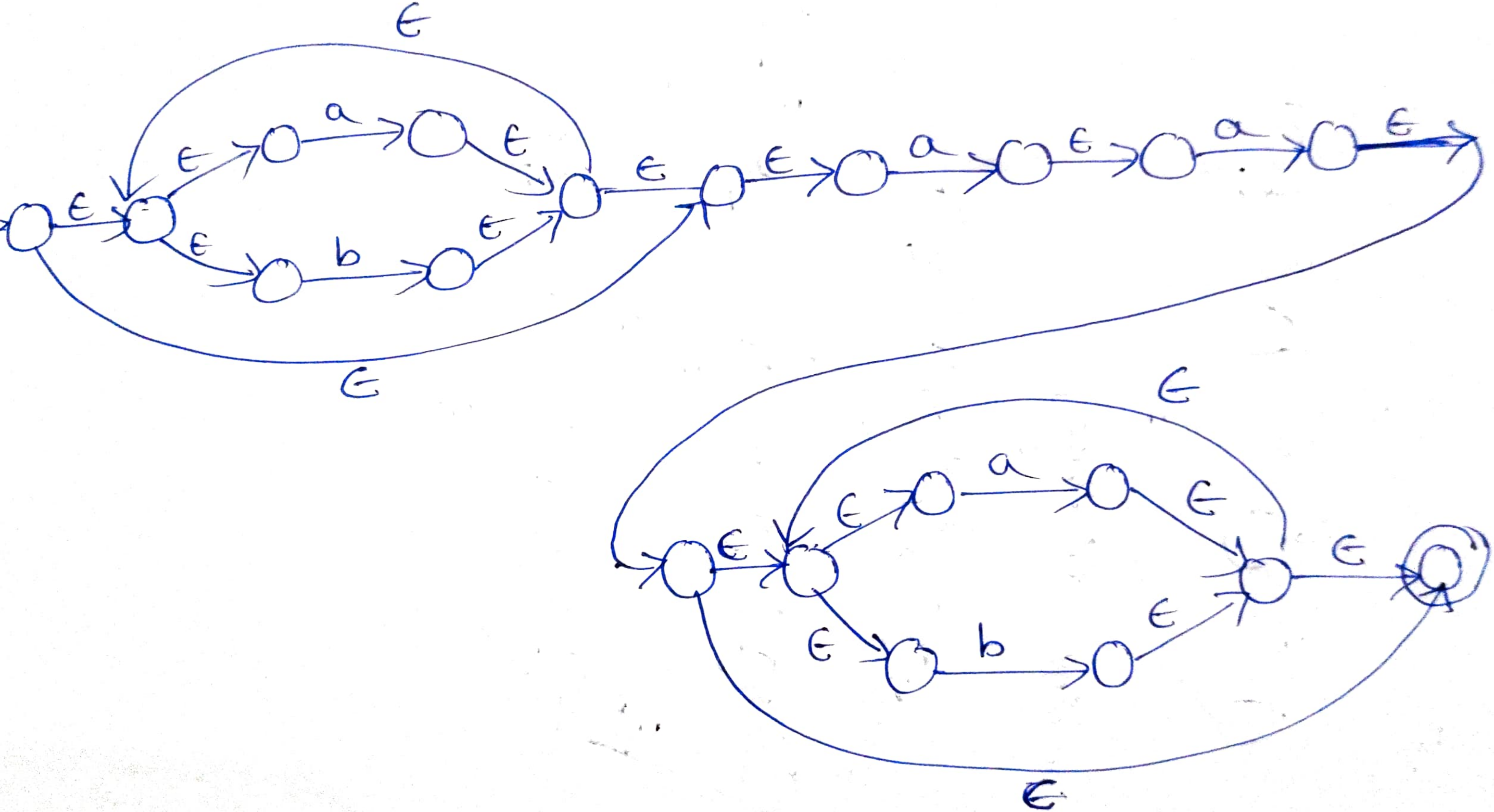
c^*,



Combine all these,



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



2.6.1. To Obtain ε–NFA from Regular Expression

Theorem: Let R be a regular expression. Then there exists a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ which accepts L(R).

or

Prove that there exists a finite automataon to accept the language L(R) corresponding to the regular expression R.

Proof: By definition, ϕ, ε and *a* are regular expressions. So, the corresponding machines to recognize the language for the respective expressions are shown in Figure 2.2.

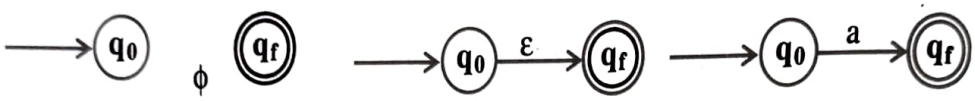


Figure 2.2. ε-NFAs to accept ϕ, ε and a

The schematic representation of a regular expression R to accept the language L(R) is shown in Figure 2.3, where q is the start state and f is the final state of machine M.

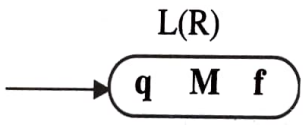


Figure 2.3. Schematic representation of FA accepting L(R)

In the definition of a regular expression it is clear that if R and S are regular expression, then R+S and R.S and R* are regular expressions which clearly uses three operators '+', '*' and '.'. Let us take each case separately and construct equivalent machine.

Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, f_1)$ be a machine which accepts the language $L(R_1)$ corresponding to the regular expression R_1. Let $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, f_2)$ be a machine which accepts the language $L(R_2)$ corresponding to the regular expression R_2. Then the various machines corresponding to the regular expressions $R_1 + R_2$, $R_1 . R_2$ and R* are shown below:

Case 1: $R = R_1 + R_2$. We can construct an NFA which accepts either $L(R_1)$ or $L(R_2)$ which can be represented as $L(R_1 + R_2)$ as shown in Figure 2.4.

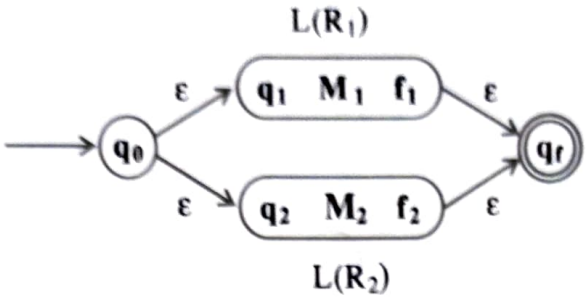


Figure 2.4. To accept the language L(R1 + R2)

It is clear from Figure 2.4 that the machine can either accept $L(R_1)$ or $L(R_2)$. Here, q_0 is the start state of the combined machine and q_f is the final state of combined machine M.

Case 2: $R = R_1 . R_2$. We can construct an NFA which accepts $L(R_1)$ followed by $L(R_2)$ which can be represented as $L(R_1 . R_2)$ as shown in Figure 2.5.

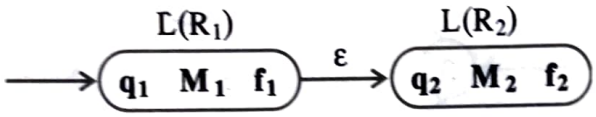


Figure 2.5. To accept the language L(R1 . R2)

It is clear from Figure 2.5 that the machine after accepting $L(R_1)$ moves from state q_1 to f_1. Since there is a ε-transition, without any input there will be a transition from state f_1 to state q_2. In state q_2, upon accepting $L(R_2)$, the machine moves to f_2 which is the final state. Thus, q_1 which is the start state of machine M_1 becomes the start state of the combined machine M and f_2 which is the final state of machine M_2, becomes the final state of machine M and accepts the language $L(R_1.R_2)$.

Case 3: $R = (R_1)^*$. We can construct an NFA which accepts either $L(R_1)^*$) as shown in Figure 2.6.

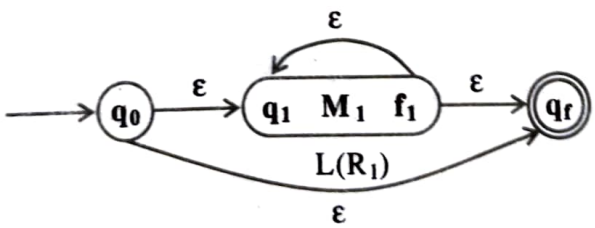


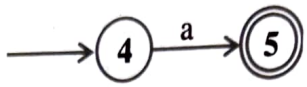
Figure 2.6. To accept the language L(R1)*

It is clear from Figure 2.6 that the machine can either accept ε or any number of $L(R_1)$s thus accepting the language $L(R_1)^*$. Here, q_0 is the start state q_f is the final state.

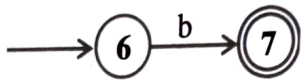
(1) Obtain an NFA which accepts strings of a's and b's starting with the string ab.

The regular expression corresponding to this language is ab(a+b)*.

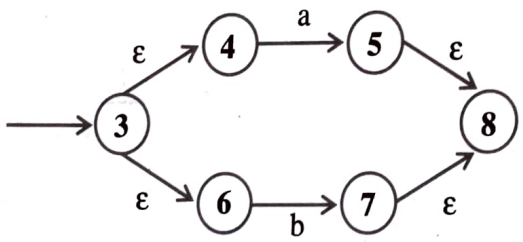
Step 1: The machine to accept 'a' is shown below.



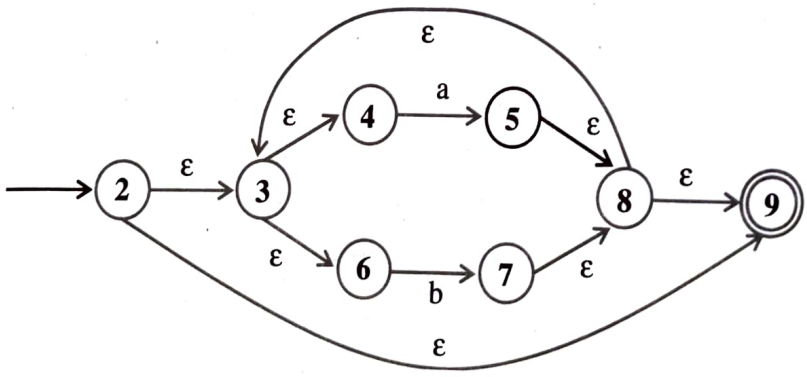
Step 2: The machine to accept 'b' is shown below.



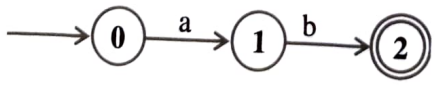
Step 3: The machine to accept (a + b) is shown below.



Step 4: The machine to accept (a+b)* is shown below.



Step 5: The machine to accept ab is shown below.



Step 6: The machine to accept ab(a+b)* is shown below.

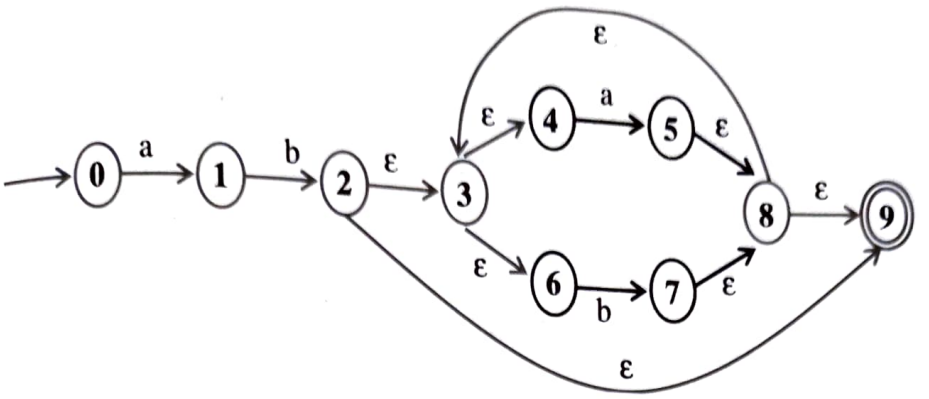
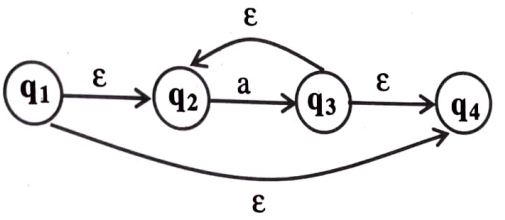


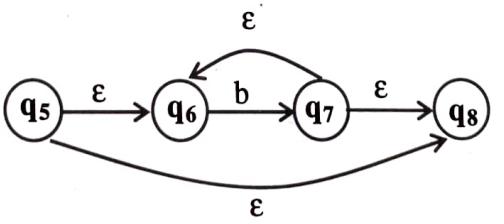
Figure 2.7. To accept the language L(ab(a+b)*)

2) **Obtain an NFA for the regular expression a* + b* + c*.**

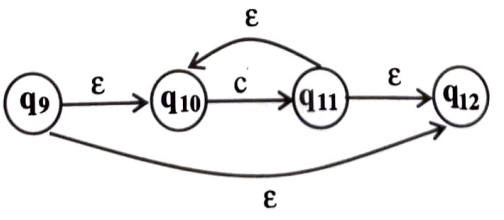
The machine corresponding the regular expression a* can be written as



The machine corresponding the regular expression b* can be written as



The machine corresponding the regular expression c* can be written as



The machine corresponding the regular expression a* + b* + c* is shown in Figure 2.8.

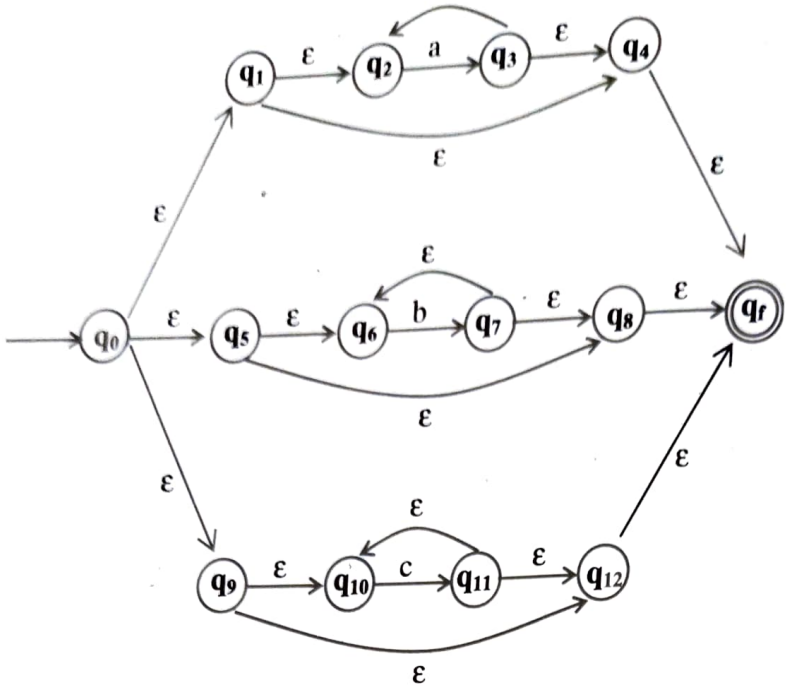
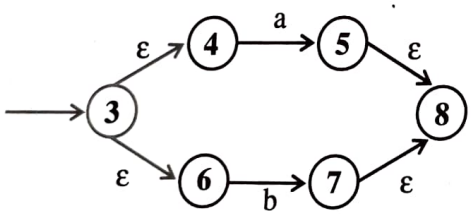


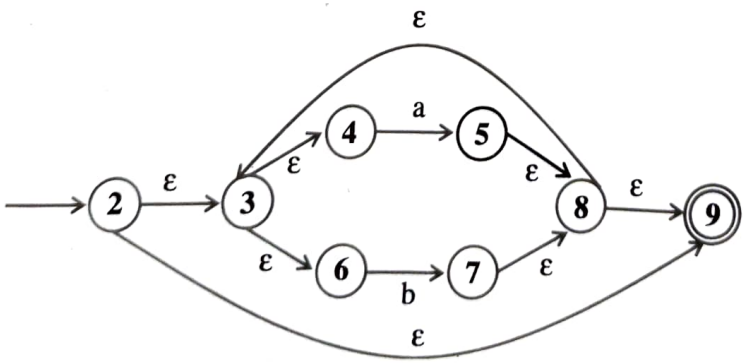
Figure 2.8. To accept the language L(a* + b* + c*)

(3) Obtain an NFA for the regular expression (a+b)*aa(a+b)*.

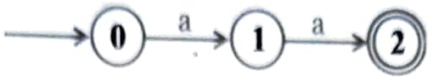
Step 1: The machine to accept (a + b) and (a+b)* are shown below:



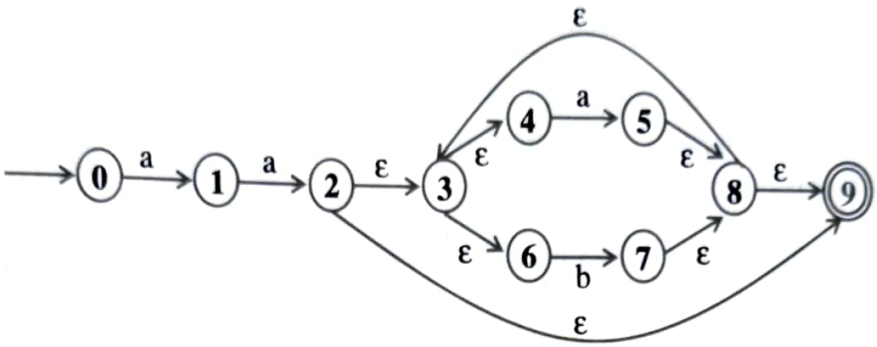
Step 2: The machine to accept (a+b)* is shown below:



Step 3: The machine to accept aa is shown below:



Step 4: The machine to accept aa(a+b)* is shown below:



Step 5: The machine to accept (a+b)*aa(a+b)* is shown in Figure 2.9.

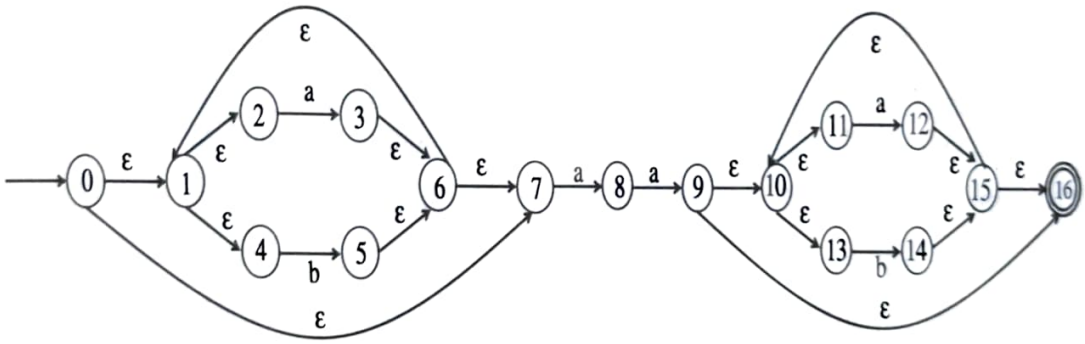


Figure 2.9. NFA to accept (a+b)*aa(a+b)*

2.6.2. To Obtain RE from FA (Kleene's Theorem)

In this section let us see how to obtain a regular expression from FA using Kleene's theorem.

Theorem: Let $M = (Q, \Sigma, \delta, q_1, F)$ be an FA recognizing the language L. Then there exists an equivalent regular expression R for the regular language L such that $L = L(R)$.

or

If the language L is accepted by a DFA then prove that there is a regular expression R for the language L so that $L = L(R)$

or

Prove that if L = L(A) for some DFA A, then there is a regular expression R such that L = L(R).

Note: The proof along with the procedure to obtain a regular expression from finite automaton is shown below:

Proof: Let $Q = \{q_1, q_2, \ldots, q_n\}$ are the states of machine M where *n* is the number of states. The path from state to *i* to state *j* through an intermediate state whose number is not greater than *k* is given by the regular expression R_{ij}^{k} as shown below:

$R_{ij}^{k} = \{w \in \Sigma^* \mid$ w is label (path) from *i* to state *j*
that goes through an intermediate state
whose number is not greater than *k*$\}$

where $i > k$ and $j > k$. The string *w* can be written as

$$w = xy$$

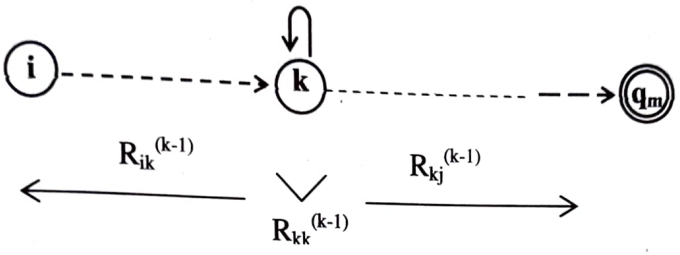
where $|x| > 0$ and $|y| > 0$ and $\hat{\delta}(i, x) = k$ and $\hat{\delta}(k,y) = j$.

Basis: $k = 0$. This indicates that there is no intermediate state and the path from state *i* to state *j* is given by the following two conditions:

1. There is a direct edge from state *i* to state *j*. This is possible when $i \neq j$. Here, a DFA M with all input symbols *a* such that there is a transition from state *i* to state *j* is considered with following cases:
   a) No input symbol and the corresponding regular expression is given by
      $$R_{ij}^{(0)} = \phi$$
   b) Exactly one input symbol *a* on which there is a transition from state *i* to state to *j* and the corresponding regular expression is given by
      $$R_{ij}^{(0)} = a$$
   c) There are multiple inputs $a_1, a_2, a_3 \ldots a_k$ where there is a transition from each symbol from state *i* to state to *j* and the corresponding regular expression is given by
      $$R_{ij}^{(0)} = a_1 + a_2 + a_3 + \ldots + a_k$$
2. There is only one state such that i = j and there exists a path from *i* to itself on input symbol *a* forming a self loop or a path of length 0 (i.e., if no loop path from state *i* to state *i* then there is a path of length 0) and is denoted by ε. Thus the regular expressions corresponding to various cases for 1.a, 1.b and 1.c is given by
   $$R_{ij}^{(0)} = \phi + \varepsilon$$
   $$R_{ij}^{(0)} = a + \varepsilon$$
   $$R_{ij}^{(0)} = a_1 + a_2 + a_3 + \ldots + a_k + \varepsilon$$

Induction: Suppose, there exists a path from i to j through a state which is not higher than k. This situation leads to two cases:

1. There exists a path from state i to state j which does not go through k and so the language accepted is $R_{ij}^{(k-1)}$.
2. There exists a path from state i to state j through k as shown below:



The path from state i to state j can be broken into several pieces as shown below:

1. The path from state i to state k and not passing through a state higher than k is given by

   $R_{ik}^{(k-1)}$

2. The path from state k to state k and not passing through a state higher than k (may exist zero or more times) is given by

   4. $(R_{kk}^{(k-1)})^*$

3. The path from state k to state j and not passing through a state higher than k is given by

   $R_{kj}^{(k-1)}$

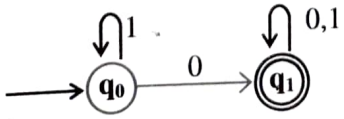
So, the regular expression for the path from state i to state j through no state higher than k is given by the concatenation of above 3 regular expressions as shown below:

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

By constructing the regular expressions in increasing order of subscripts, each $R_{ij}^{(k)}$ depends only on expressions with a smaller superscript and all the regular expressions are available whenever there is a need. Finally, we will have $R_{ij}^{(n)}$ for all i and j.

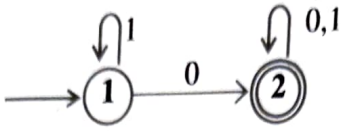
Note: Assume that state 1 is the initial state and the regular expression for the language is the sum of all regular expressions $R_{ij}^{(n)}$ where state j is an accepting state.

(1) Obtain a regular expression for the FA shown below:



What is the language corresponding to the regular expression?

Solution: Let $q_0 = 1$ and $q_1 = 2$. By renaming the states, the above FA can be written as shown below:



By following the procedure shown in Kleene's theorem (section 2.4) we have:

Basis: when k = 0

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

Note: If the beginning and ending states are same add ε which denotes the length 0

Induction: The regular expression corresponding to the path from state *i* to state *j* through a state which is not higher than *k* is given by

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} [R_{kk}^{(k-1)}]^* R_{kj}^{(k-1)}$$

When k = 1 (i.e., path from state *i* to state *j* through a state not higher than 1): The various regular expressions obtained are shown below:

$$\begin{aligned} R_{11}^{(1)} &= R_{11}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{11}^{(0)} \\ &= (\varepsilon + 1) + (\varepsilon + 1)(\varepsilon + 1)^* (\varepsilon + 1) \\ &= (\varepsilon + 1) + (\varepsilon + 1) 1^* (\varepsilon + 1) \\ &= (\varepsilon + 1) + 1^* \\ &= 1^* \end{aligned}$$

$$\begin{aligned} R_{12}^{(1)} &= R_{12}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \\ &= 0 + (\varepsilon + 1)(\varepsilon + 1)^* 0 \\ &= 0 + 1^* 0 \\ &= 1^* 0 \end{aligned}$$

$$\begin{aligned} R_{21}^{(1)} &= R_{21}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{11}^{(0)} \\ &= \phi + \phi (\varepsilon + 1)^* (\varepsilon + 1) \\ &= \phi \end{aligned}$$

$$\begin{aligned} R_{22}^{(1)} &= R_{22}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \\ &= (\varepsilon + 0 + 1) + \phi (\varepsilon + 1)^* 0 \\ &= (\varepsilon + 0 + 1) \end{aligned}$$

When k = 2 (i.e., path from state *i* to state *j* through a state not higher than 2): The various regular expressions are given by

$$R_{11}^{(2)} = R_{11}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{21}^{(1)}$$
$$= 1^* + 1^*0 (\varepsilon + 0 + 1)^* \phi$$
$$= 1^*$$
$$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)}$$
$$= 1^*0 + 1^*0 (\varepsilon + 0 + 1)^* (\varepsilon + 0 + 1)$$
$$= 1^*0 + 1^*0 (0 + 1)^* (\varepsilon + 0 + 1)$$
$$= 1^*0 + 1^*0 (0 + 1)^*$$
$$= 1^*0 (0 + 1)^*$$
$$R_{21}^{(2)} = R_{21}^{(1)} + R_{22}^{(1)} [R_{22}^{(1)}]^* R_{21}^{(1)}$$
$$= \phi + (\varepsilon + 0 + 1) (\varepsilon + 0 + 1)^* \phi$$
$$= \phi$$
$$R_{22}^{(2)} = R_{22}^{(1)} + R_{22}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)}$$
$$= (\varepsilon + 0 + 1) + (\varepsilon + 0 + 1) (\varepsilon + 0 + 1)^* (\varepsilon + 0 + 1)$$
$$= (\varepsilon + 0 + 1) + (0 + 1)^*$$
$$= (0 + 1)^*$$

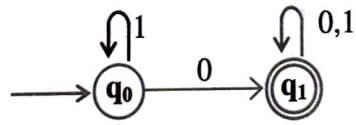
Note: Since the total number of states in the given machine is 2, maximum value of k should be 2.

Since the start state is 1 and final state is 2, the regular expression is given by

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

So, the regular expression for the given DFA is $1^*0 (0 + 1)^*$ which is the language consisting of any number of 1's followed by a zero and then followed by strings of 0's and 1's. This can also be expressed as strings of 0's and 1's with at least one zero.

(2) Obtain a regular expression for the FA shown below:



What is the language corresponding to the regular expression?

Note: The solution for this problem is already given in previous example. Another approach to solve this problem is that, instead of calculating the regular expressions for R_{ij} from k = 0 to n, start from k = n, and then work back to the case when k = 0.

In the current example, number of states $n = 2$ and hence to start with assume k = 2. The regular expression is given by

$$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)} \qquad (1)$$

It is clear from the above expression that $R_{12}^{(1)}$ and $R_{22}^{(1)}$ are required and are obtained using the following regular expressions:

$$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \quad (2)$$
$$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \quad (3)$$

The regular expressions for these two equations can be obtained if we know $R_{11}^{(0)}$, $R_{12}^{(0)}$, $R_{21}^{(0)}$ and $R_{22}^{(0)}$ which are obtained when k = 0 as shown below:

When k = 0:

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

Substituting these values in Eq. (2) and Eq. (3) we have,

$$\begin{aligned} R_{12}^{(1)} &= R_{12}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \\ &= 0 + (\varepsilon + 1)(\varepsilon + 1)^* 0 \\ &= 0 + 1^* 0 \\ &= 1^* 0 \end{aligned}$$

$$\begin{aligned} R_{22}^{(1)} &= R_{22}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \\ &= (\varepsilon + 0 + 1) + \phi (\varepsilon + 1)^* 0 \\ &= (\varepsilon + 0 + 1) \end{aligned}$$

Substituting for $R_{12}^{(1)}$ and $R_{22}^{(1)}$ in Eq. (1) we have

$$\begin{aligned} R_{12}^{(2)} &= R_{12}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)} \\ &= 1^* 0 + 1^* 0 (\varepsilon + 0 + 1)^* (\varepsilon + 0 + 1) \\ &= 1^* 0 + 1^* 0 (0 + 1)^* (\varepsilon + 0 + 1) \\ &= 1^* 0 + 1^* 0 (0 + 1)^* \\ &= 1^* 0 (0 + 1)^* \end{aligned}$$

So, the regular expression for the given DFA is $1^*0\ (0 + 1)^*$ which is the language consisting of any number of 1's followed by a zero and then followed by strings of 0's and 1's. This can also be expressed as strings of 0's and 1's with at least one zero.

(3) Consider the DFA shown below:

	Σ	
States	0	1
$\rightarrow q_1$	q_2	q_1
q_2	q_3	q_1
$*q_3$	q_3	q_2

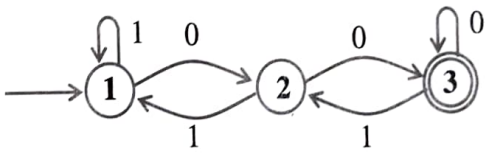
Obtain the regular expressions $R_{ij}^{(0)}$, $R_{ij}^{(1)}$ and simplify the regular expressions as much as possible.

Note: The symbol * preceding state q_3 indicates that q_3 is the final state.

Solution: Rename the states 1, 2 and 3 in order and the resulting DFA is shown below:

	Σ	
States	0	1
→1	2	1
2	3	1
*3	3	2

where state 3 is the final state and state 1 is the start state. The corresponding transition diagram is shown below:



By following the procedure shown in Kleene's theorem (Section 2.4) we have:

Basis: when k = 0

$R_{11}^{(0)} = \varepsilon + 1$
$R_{12}^{(0)} = 0$
$R_{13}^{(0)} = \phi$
$R_{21}^{(0)} = 1$
$R_{22}^{(0)} = \phi + \varepsilon = \varepsilon$
$R_{23}^{(0)} = 0$
$R_{31}^{(0)} = \phi$
$R_{32}^{(0)} = 1$
$R_{33}^{(0)} = \varepsilon + 0$

Note: If the beginning and ending states are same add ε which denotes the length 0 (i.e., whenever i = j)

Induction: The regular expression corresponding to the path from state *i* to state *j* through a state which is not higher than *k* is given by

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} [R_{kk}^{(k-1)}]^* R_{kj}^{(k-1)}$$

When k = 1 (i.e., path from state *i* to state *j* through a state not higher than 1): The various regular expressions obtained are shown below:

$$R_{11}^{(1)} = R_{11}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{11}^{(0)}$$
$$= (\varepsilon + 1) + (\varepsilon + 1)(\varepsilon + 1)^* (\varepsilon + 1)$$
$$= (\varepsilon + 1) + (\varepsilon + 1) 1^* (\varepsilon + 1)$$
$$= (\varepsilon + 1) + 1^*$$
$$= 1^*$$

$$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)}$$
$$= 0 + (\varepsilon + 1)(\varepsilon + 1)^* 0$$
$$= 0 + 1^* 0$$
$$= 1^* 0$$

$$R_{13}^{(1)} = R_{13}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{13}^{(0)}$$
$$= \phi + (\varepsilon + 1)^* (\varepsilon + 1) \phi$$
$$= \phi$$

$$R_{21}^{(1)} = R_{21}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{11}^{(0)}$$
$$= 1 + 1 (\varepsilon + 1)^* (\varepsilon + 1)$$
$$= 1 + 11^* = 11^*$$

$$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)}$$
$$= \varepsilon + 1 (\varepsilon + 1)^* 0$$
$$= \varepsilon + 11^* 0$$

$$R_{23}^{(1)} = R_{23}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{13}^{(0)}$$
$$= 0 + 1 (\varepsilon + 1)^* \phi$$
$$= 0$$

$$R_{31}^{(1)} = R_{31}^{(0)} + R_{31}^{(0)} [R_{11}^{(0)}]^* R_{11}^{(0)}$$
$$= \phi + \phi (\varepsilon + 1)^* (\varepsilon + 1)$$
$$= \phi$$

$$R_{32}^{(1)} = R_{32}^{(0)} + R_{31}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)}$$
$$= 1 + \phi (\varepsilon + 1)^* 0$$
$$= 1$$

$$R_{33}^{(1)} = R_{33}^{(0)} + R_{31}^{(0)} [R_{11}^{(0)}]^* R_{13}^{(0)}$$
$$= (\varepsilon + 0) + \phi (\varepsilon + 0)^* \phi$$
$$= (\varepsilon + 0)$$

When k = 2 (i.e., path from state *i* to state *j* through a state not higher than 2): The various regular expressions are given by

$$R_{11}^{(2)} = R_{11}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{21}^{(1)}$$
$$= 1^* + 1^* 0 (\varepsilon + 11^* 0)^* 11^*$$
$$= 1^* + 1^* 0 (11^* 0)^* 11^*$$

$$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)}$$
$$= 1^* 0 + 1^* 0 (\varepsilon + 11^* 0)^* (\varepsilon + 11^* 0)$$
$$= 1^* 0 + 1^* 0 (11^* 0)^* (\varepsilon + 11^* 0)$$

$$R_{13}^{(2)} = R_{13}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{23}^{(1)}$$
$$= \phi + 1^*0 (\varepsilon + 11^*0)^* 0$$
$$= 1^*0(11^*0)^*0$$
$$R_{21}^{(2)} = R_{21}^{(1)} + R_{22}^{(1)} [R_{22}^{(1)}]^* R_{21}^{(1)}$$
$$= 11^* + (\varepsilon + 11^*0) (\varepsilon + 11^*0)^* 11^*$$
$$= 11^* + (\varepsilon + 11^*0) (11^*0)11^*$$
$$R_{22}^{(2)} = R_{22}^{(1)} + R_{22}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)}$$
$$= (\varepsilon + 11^*0) + (\varepsilon + 11^*0) (\varepsilon + 11^*0)^* (\varepsilon + 11^*0)$$
$$= (\varepsilon + 11^*0) + (\varepsilon + 11^*0) (11^*0)^* (\varepsilon + 11^*0)$$
$$R_{23}^{(2)} = R_{23}^{(1)} + R_{22}^{(1)} [R_{22}^{(1)}]^* R_{23}^{(1)}$$
$$= 0 + (\varepsilon + 11^*0) (\varepsilon + 11^*0)^* 0$$
$$= 0 + (\varepsilon + 11^*0) (11^*0)^* 0$$
$$R_{31}^{(2)} = R_{31}^{(1)} + R_{32}^{(1)} [R_{22}^{(1)}]^* R_{21}^{(1)}$$
$$= \phi + 1(\varepsilon + 11^*0)^* 11^*$$
$$= 1(11^*0)^*11^*$$
$$R_{32}^{(2)} = R_{32}^{(1)} + R_{32}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)}$$
$$= 1 + 1(\varepsilon + 11^*0)^* (\varepsilon + 11^*0)$$
$$= 1 + 1(11^*0)^* (\varepsilon + 11^*0)$$
$$R_{33}^{(2)} = R_{33}^{(1)} + R_{32}^{(1)} [R_{22}^{(1)}]^* R_{23}^{(1)}$$
$$= (0 + \varepsilon) + 1(11^*0)^* 0$$

The regular expression is given by R_{13} can be calculated as shown below:

$$R_{13}^{(3)} = R_{13}^{(2)} + R_{13}^{(2)} [R_{33}^{(2)}]^* R_{33}^{(2)}$$
$$= 1^*0(11^*0)^*0 + 1^*0(11^*0)^*0 [(0 + \varepsilon) + 1(11^*0)^* 0]^* (0 + \varepsilon) + 1(11^*0)^* 0$$

2.6.3. To Obtain RE from FA (By Eliminating States)

In this section let us see how to obtain a regular expression from FA.

Theorem: Let $M = (Q, \Sigma, \delta, q_0, F)$ be an FA recognizing the language L. Then there exists an equivalent regular expression R for the regular language L such that $L = L(R)$.

The general procedure to obtain a regular expression from FA is shown below. Consider the generalized graph:

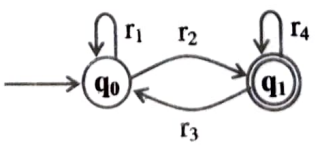


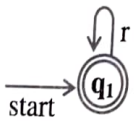
Figure 2.10. Generalized transition graph

where r_1, r_2, r_3 and r_4 are the regular expressions and correspond to the labels for the edges. The regular expression for this can take the form:

$$r = r_1^* r_2 (r_4 + r_3 r_1^* r_2)^* (2.1)$$

Note:

1. For each final state (accepting state) q, apply the reduction procedure and bring the graph to the form shown in Figure 2.10.
2. If q is not the start state, the reduced graph obtained will be of the form shown in figure 2.10. and use the equation 2.1 to obtain the regular expression.
3. If the start state is also an accepting state, the state elimination process has to be performed so that we should get rid of every state except the start state. The final automaton will be of the form



4. The final regular expression is the sum of all the regular expressions obtained from the reduced automata for each accepting state.

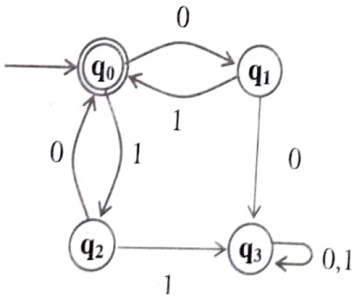
For example, If r_3 is not there in Figure 2.10, the regular expression can be of the form

$$r = r_1^* r_2 r_4^* (2.2)$$

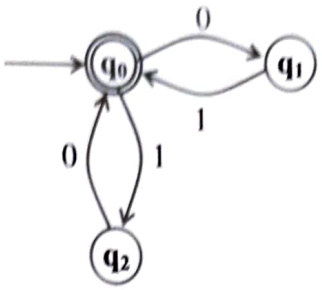
If q_0 and q_1 are the final states, then the regular expression can be of the form

$$r = r_1^* + r_1^* r_2 r_4^* (2.3)$$

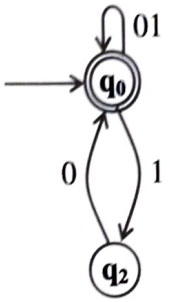
(1) Obtain a regular expression for the FA shown below:



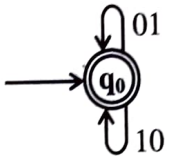
It is clear from the FA that q_3 is the dead state (i.e., once the state q_3 is reached irrespective of the input, the machine stays in q_3 only and there is no way to reach the final state) and so all the edges connected to q_3 can be removed and the resulting figure is shown below:



It is clear from the figure that if we input the string 01, the machine goes to state q_1 and comes back to q_0 and the process can be repeated. So, instead of q_1, we can loop back on the string 01 as shown below:



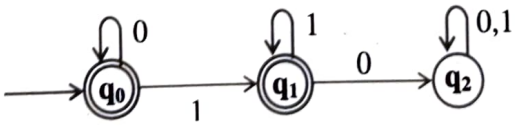
Similarly on 10, the machine comes back to q_0 and so we can replace it by another loop with the edge labeled 10 as shown:



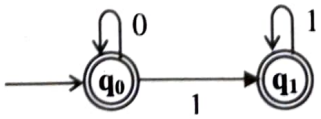
It is clear from this figure that the machine accepts strings of 01's and 10's of any length and the regular expression can be of the form

$$(01 + 10)^*$$

(2) What is the language accepted by the following FA.



Since, state q_2 is the dead state, it can be removed and the following FA is obtained.



The state q_0 is the final state and at this point it can accept any number of 0's which can be represented using notation as

$$0^*$$

q_1 is also the final state. So, to reach q_1 one can input any number of 0's followed by 1 and followed by any number of 1's and can be represented as

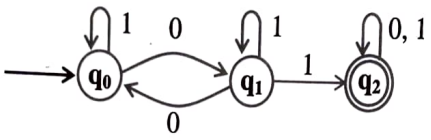
$$0^*11^*$$

So, the final regular expression is obtained by adding 0^* and 0^*11^*. So, the regular expression is

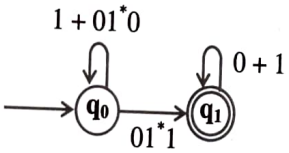
$$\begin{aligned} R.E &= 0^* + 0^*11^* \\ &= 0^*(\varepsilon + 11^*) \\ &= 0^*(\varepsilon + 1^+) \\ &= 0^*(1^*) = 0^*1^* \end{aligned}$$

It is clear from the regular expression that language consists of any number of 0's (possibly ε) followed by any number of 1's (possibly ε).

(3) Obtain a regular expression for the FA shown below:



The graph should be converted into generalized graph (shown in Figure 2.10) by eliminating state q_1 as shown below.



By comparing this figure with Figure 2.10, we have

$$\begin{aligned} r_1 &= 1 + 01^*0 \\ r_2 &= 01^*1 \\ r_3 &= \phi \\ r_4 &= 0 + 1 \end{aligned}$$

By substituting these in Eq. (2.1) we have

$$\begin{aligned} r &= (1 + 01^*0)^* 01^*1[\,(0+1) + \phi(1 + 01^*0)^*01^*1]^* \\ &= (1 + 01^*0)^* 01^*1[\,(0+1) + \phi]^* \\ &= (1 + 01^*0)^* 01^*1\,(0+1)^* \end{aligned}$$

So, the regular expression for the given FA is $(1 + 01^*0)^* 01^*1\,(0+1)^*$

2.7. Applications of Regular Expressions

Regular expressions in UNIX: Regular expressions are extensively used in UNIX operating system. But, certain short hand notations are used in UNIX platform using which complex regular expressions are avoided. For example, the symbol '.' stands for any character, the sequence [abcde...] stands for the regular expression "a + b + c + d + e...", the operator | is used in place of +, the operator ? means "zero or one of" etc. Most of the commands are invoked invariably uses regular expressions. For example, grep (**G**lobal search for **R**egular **E**xpression and **P**rint) used to search for a pattern of string.

Pattern Matching refers to a set of objects with some common properties. We can match an identifier or a decimal number or we can search for a string in the text.

Lexical analysis: Regular expressions are extensively used in the design of lexical analyzer phase (This is the first phase of the compiler design). This phase scans the source program and recognizes all the tokens which are logically together. The UNIX commands such as *lex* accepts regular expressions as the input and produces the lexical analyzer generator. This generator takes a high-level description of a lexical analyzer as the input and produces lexical analyzer.

Obtain a regular expression to identify an identifier.

An identifier stats with a letter. This letter can be followed by combination of zero or more letters and digits i.e., an identifier can be a single letter followed by strings of letters and digits of any length and can be represented as

letter (letter + digit)*

Obtain a regular expression to identify an integer.

An integer can start with any of the signs +, - or ε (null string means no sign) followed by one or more digits ranging from 0 to 9. This can be represented using a regular expression as

sd^+ or sdd*

where *s* stands for sign and *d* stands for the digits from 0 to 9.

An application of regular expression in UNIX editor ed.

In UNIX operating system, we can use the editor *ed* to search for a specific pattern in the text. For example, if the command specified is

/acb*c/

then the editor searches for a string which starts with *ac* followed by zero or more b's and followed by the symbol *c*. Note that the editor ed accepts the regular expression and searches for that particular pattern in the text. As the input can vary dynamically, it is challenging to write programs for string patters of these kinds.