2.4. Difference Between DFA, NFA and ε-NFA

Now, let us see "What are the difference between DFA, NFA and ε-NFA?" Strictly speaking, difference between DFA and NFA lies only in the definition of δ. Using this difference some more points can be derived and can be written as shown:

DFA	NFA	ε-NFA
The DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where • Q is set of finite states • Σ is set of input alphabets • $\delta : Q \times \Sigma$ to Q • q_0 is the start state • $F \subseteq Q$ is set of final states	An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where • Q is set of finite states • Σ is set of input alphabets • $\delta : Q \times \Sigma$ to 2^Q • q_0 is the start state • $F \subseteq Q$ is set of final states	An ε-NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where • Q is set of finite states • Σ is set of input alphabets • $\delta : Q \times (\Sigma \cup \varepsilon)$ to 2^Q • q_0 is the start state • $F \subseteq Q$ is set of final states
• There can be zero or one transition from a state on an input symbol	• There can be zero, one or more transitions from a state on an input symbol	• There can be zero, one or more transitions from a state with or without giving any input
• More number of transitions	• Less number of transitions	• Relatively more transitions when compared with NFA
• Difficult to construct	• Easy to construct	• Easy to construct using regular expressions
• Less powerful since at any point of time it will be in only one state	• More powerful than DFA since at any point of time it will be in more than one state	• More powerful than NFA since at any point of time it will be in more than one state with or without giving any input

2.5. 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) 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.

1. ϕ is a regular expression denoting an empty language.
2. ε-(epsilon) is a regular expression indicates the language containing an empty string.
3. *a* is a regular expression which indicates the language containing only {a}.
4. If R is a regular expression denoting the language L_R and S is a regular expression denoting the language L_S, then
   a) R+S is a regular expression corresponding to the language $L_R \cup L_S$.
   b) R.S is a regular expression corresponding to the language $L_R.L_S$.
   c) R* is a regular expression corresponding to the language L_R^*.
5. 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 of c'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 strings 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 or 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

(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)

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

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

(2) Obtain a regular expression to accept 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 $(\varepsilon + a + b)^2$

(3) Obtain a regular expression to accept 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 $(\varepsilon + a + b)^{10}$

(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))*

(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)

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

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

or

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

(6) Obtain a regular expression such that L(R) = {w | w ∈ {0, 1}* with at least three consecutive 0's.

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

$L(R) = \{ (0+1)^m 000 (0+1)^n \mid m \geq 0 \text{ and } n \geq 0 \}$

(7) Obtain a regular expression to accept strings of 0's and 1's having no two consecutive zeros.

The first observation from the statement is that whenever a 0 occurs it should be followed by 1. But, there is no restriction on the number of 1's. So, it is a string consisting of any combinations

of 1's and 01's. So, the partial regular expression for this can be of the form

$(1 + 01)^*$

No doubt that the above expression is correct. But, suppose the string ends with a 0. What to do? For this, the string obtained from above regular expression may end with 0 or may end with ε which can be represented by the regular expression:

$(0 + \varepsilon)$

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

(8) Obtain a regular expression to accept string of a's and b's starting with 'a' and ending with 'b'.

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 following the above RE with *b*.

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

(9) Obtain a regular expression to accept string of a's and b's whose second symbol from the right end is a.

The first symbol from the right end can be *a* or *b* which can be represented by the RE

$(a + b)$

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

(10) Obtain a regular expression representing string of a's and b's whose tenth symbol from the right end is a.

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:

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

Since there are ten (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.

(11) Obtain a regular expression to accept strings of a's and b's such that third symbol from the right is *a* and fourth symbol from the right is *b*.

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

(12) Obtain a regular expression to accept the words with two or more letters but beginning and ending with the same letter where $\Sigma = \{a, b\}$.

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:

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

(13) Obtain a regular expression to accept strings of a's and b's whose length is either even or multiples of 3 or both.

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:

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

(14) Obtain a regular expression to accept strings of a's and b's such that every block of four consecutive symbols contains at least two a's.

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

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

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:

So, R.E 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)+

(15) Obtain a regular expression for the language L = {$a^n b^m$ | m + n is even}.

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.

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

(16) Obtain a regular expression for the language $L = \{a^n b^m \mid m \geq 1, n \geq 1, nm \geq 3\}$.

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

m = 1 n ≥ 3

n = 1 m ≥ 3

m, n ≥ 2

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

$RE = aaaa^* b$

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

$RE = a\ bbbb^*$

Case 3: Since $nm \geq 3$, if $m \geq 2$ then $n \geq 2$. 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 = $aaaa^*\ b + a\ bbbb^* + aaa^*\ bbb^*$

(17) Obtain a regular expression for the language $L = \{a^{2n} b^{2m} \mid n \geq 0, m \geq 0\}$.

Solution: For every $n \geq 0$, a^{2n} results in even number of a's and for every $m \geq 0$, 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 $L = \{a^{2n} b^{2m} \mid n \geq 0$ and $m \geq 0\}$ is obtained by concatenating the above two regular expressions.

So, R.E = $(aa)^*\ (bb)^*$

(18) 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 cases:

- ε indicating zero a's

or

- one a

or

- two a's

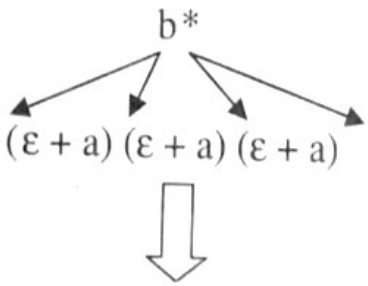
or

- three a's

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

RE: $(\varepsilon + a)\ (\varepsilon + a)\ (\varepsilon + a)$

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



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

(19) 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 $n \geq 4$). 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:

  $RE = (\varepsilon + b)\ (\varepsilon + b)\ (\varepsilon + b)$ (2)

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

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

(20) 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:

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

(21) Obtain a regular expression for the language L = {w : |w| mod 3 = 0 where w ∈ (a,b)*}.

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.

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

Note: The above regular expression and the regular expression obtained in previous problem are same.

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

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*)*

(23) Obtain a regular expression for the set of all strings that do not end with 01 over {0, 1}*.

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 $(0 + 1)^*$. These strings can end with 00 or 10 and 11:

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

(24) Obtain a regular expression for L = {vuv : u, v ∈ {a, b}* and |v| = 2}.

Since |v| = 2, the string having length 2 can be represented by the regular expression:

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 ∈ {a,b}* and |v| = 2}

is obtained by concatenating the regular expressions V, U and V.

So, R.E = aa(a + b)*aa + ab(a + b)*ab + ba(a + b)*ba + bb) + bb(a + b)*bb

(25) Obtain a regular expression for L = {w: string ends with ab or ba where w ∈ {a, b}* }.

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 (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

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

- (a+b) has the highest precedence.
- Next preference is given for * operator.
- Next preference is given for concatenation operator.

Theorem 3.4

If $L = L(N)$ for some DFA A, then there is a regular expression E such that $L = L(E)$

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 given by the regular expression R_{ij}^k as shown below:

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

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

$$w = xy$$

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

Basis: Take $K = 0$.

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 ways:

(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 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 j & corresponding regular expression is

$$R_{ij}^{(0)} = a_1 + a_2 + a_3 + \ldots + a_k$$

2 There is only one state such that $i=j$ & there exists a path from i to itself on input symbol a forming a self loop or a path of length one (i.e., of 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 expression corresponding to various cases for 1.a, 1.b and 1.c is given by

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

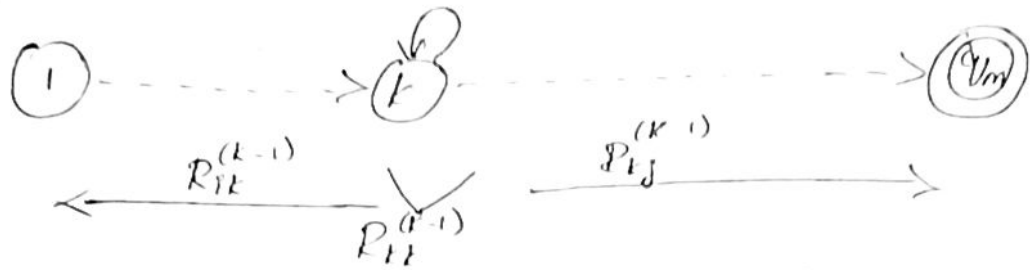
$$R_{ij}^{(0)} = a + \epsilon$$

$$R_{ij}^{(0)} = a_1 + a_2 + a_3 + \ldots + a_k + \epsilon$$

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 $P_{ij}^{(k-1)}$

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 in to 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

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

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

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

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

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

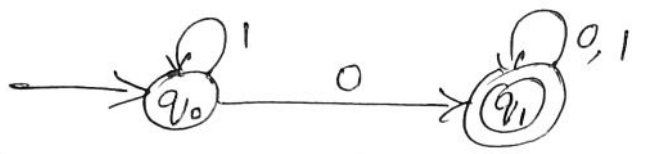
So, the regular expression for the path from state i to state j through no state higher than k is given by the concatination of above 3 regular expressions as shown below :

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

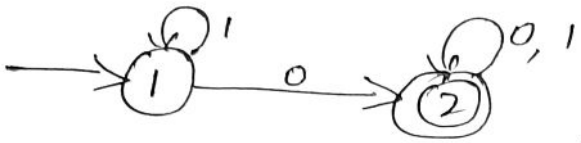
By constructing the regular expression in increasing order of subscripts, each $R_{ij}^{(k)}$ depends only on expression with a smaller superscript and all the regular expression are available whenever there is a need.

Finally, we will have $R_{ij}^{(n)}$ for all i and j

1) Obtain a regular expression for the DFA shown below



Soln: Let $q_0 = 1$ and $q_1 = 2$. By renaming the states, the above DFA can be written as



By following the procedure shown in Kleene's theorem (formula method)

$R_{ij}^{(k)} = R_{12}^{(2)} = ?$

i = initial state = 1
j = final = 2
k = Total no of states = 2
└ Total intermediate states

Formula:

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

Basis: when $k = 0$

$R_{11}^{(0)} = \epsilon + 1$ (either 1 or empty (NULL) whenever there is a self loop we can write ϵ)

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

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

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

*Some Minimization rules

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

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

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

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

$$= (\varepsilon+1) + (\varepsilon+1)(\varepsilon+1)^* (\varepsilon+1)$$

$$= (\varepsilon+1) + (\varepsilon+1)\, 1^* (\varepsilon+1)$$

$$= (\varepsilon+1) + 1^*$$

$$= 1^*$$

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

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

$$= 0 + 1^* 0$$

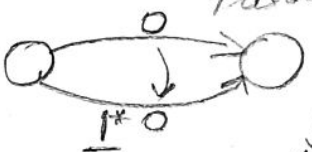
$$= 1^* 0$$

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

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

$$= 0 + 1^* 0$$

$$= 1^* 0$$



Parallel edge condition

1* includes ε also without the condition as ε + the P/p is 0 alone so we can write 0 only once

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

$$= (\varepsilon+0+1) + \phi(\varepsilon+1)^* 0$$

$$= (\varepsilon+0+1)$$

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

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

$$= 1^* + 1^* 0 (\varepsilon+0+1)^* \phi$$

$$= 1^*$$

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

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

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

$$= 1^*0 + 1^*0 (0+1)^* \quad \text{— parallel rule applied}$$

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

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

$$= \phi + (\varepsilon + 0 + 1)(\varepsilon + 0 + 1)^* \phi$$

$$= \phi$$

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

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

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

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

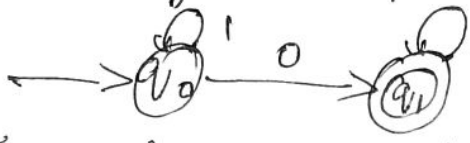
Note: Since the total number of states in the given machine is 2, maximum value of k should be 2.

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

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

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 & then followed by strings of 0's and 1's. This can also be expressed as strings of 0's and 1's with atleast one zero.

2) obtain a regular expression for FA shown below



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

In current example, no of states $n=2$ & hence to start with assume $k=2$. The Regular expression is given by

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

It is clear from above exp that $R_{12}^{(1)}$ & $R_{22}^{(1)}$ are reg and are obtained using R.E:

$$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)}[R_{11}^{(0)}]^* R_{12}^{(0)} \quad \text{——— (2)}$$

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

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

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

Substituting these values in Eq (2) & Eq (3)

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

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

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

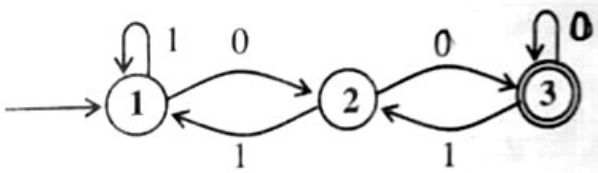
$$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)}[R_{22}^{(1)}]^* R_{22}^{(1)}$$
$$= 1^* 0 + 1^* 0(\varepsilon+0+1)^* (\varepsilon+0+1)$$
$$= 1^* 0 + 1^* 0(0+1)^* (\varepsilon+0+1)$$
$$= 1^* 0 + 1^* 0(0+1)^*$$
$$= 1^* 0(0+1)^*$$

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 regul 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

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

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

us regula

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

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

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

3) Consider the DFA shown below:

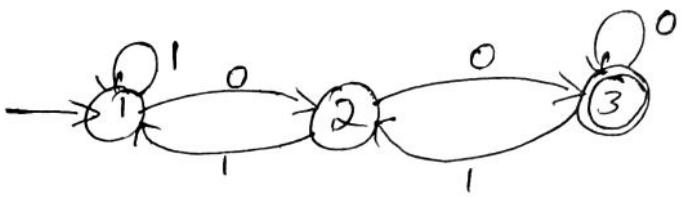
	Σ	
States	0	1
→q_1	q_2	q_1
q_2	q_3	q_1
*q_3	q_3	q_2

Obtain the regular expression for given DFA.

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

	Σ	
States	0	1
→ 1	2	1
2	3	1
* 3	3	2

Transition diagram



In this example, no of state $n = 3$ & hence $k = 3$.
The regular expression is given by

By following kleene's theorem we get

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

$$\boxed{R_{13}^{(3)} = R_{13}^{(2)} + R_{13}^{(2)} \left[R_{33}^{(2)}\right]^* R_{33}^{(2)}} \longrightarrow (1)$$

initial state $i = 1$
final state $j = 3$
No of state $k = 3$

It is clear from the above (1) eqn that $R_{13}^{(2)}$ and $R_{33}^{(2)}$ are required & are obtained using the following regular expression

$$R_{13}^{(2)} = R_{13}^{(1)} + R_{12}^{(1)} \left[R_{22}^{(1)}\right]^* R_{23}^{(1)} \longrightarrow (2)$$

$$R_{33}^{(2)} = R_{33}^{(1)} + R_{32}^{(1)} \left[R_{22}^{(1)}\right]^* R_{23}^{(1)} \longrightarrow (3)$$

For eqn (2) we need to obtain the RE are $R_{13}^{(1)}$, $R_{12}^{(1)}$, $R_{22}^{(1)}$ and $R_{23}^{(1)}$ & $R_{33}^{(1)}$, $R_{32}^{(1)}$ are obtained using all RE

$$R_{13}^{(1)} = R_{13}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{13}^{[0]} \longrightarrow (4)$$

$$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)} [R_{11}^{[0]}]^* R_{12}^{(0)} \longrightarrow (5)$$

$$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{12}^{[0]} \longrightarrow (6)$$

$$R_{23}^{(1)} = R_{23}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)})^* R_{13}^{[0]} \longrightarrow (7)$$

$$R_{33}^{(1)} = R_{33}^{(0)} + R_{31}^{(0)} [R_{11}^{(0)}]^* R_{13}^{[0]} \longrightarrow (8)$$

$$R_{32}^{(1)} = R_{32}^{(1)} + R_{31}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \longrightarrow (9)$$

The eqn (4), (5), (6), (7), (8), (9) to obtain we need to find the R.E when $k=0$.

when $k=0$:

$R_{11}^{(0)} = \varepsilon + 1$ $\quad R_{12}^{(0)} = 0$ $\quad R_{13}^{(0)} = \phi$

$R_{21}^{(0)} = 1$ $\quad R_{22}^{(0)} = \phi + \varepsilon = \varepsilon$ $\quad R_{23}^{(0)} = 0$

$R_{31}^{(0)} = \phi$ $\quad R_{32}^{(0)} = 1$ $\quad R_{33}^{(0)} = \varepsilon + 0$

Substituting the values in Eq. (4), (5), (6), (7) (8) & (9)

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

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

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

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

$$= \varepsilon + 11^* 0$$

$$= 11^* 0$$

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

$$= 0 + 1 [\varepsilon + 1]^* \phi$$

$$= 0 + \phi$$

$$= 0$$

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

$$= (\varepsilon + 0) + \phi [\varepsilon + 1] \phi$$

$$= \varepsilon + 0$$

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

$$= 1 + \phi [\varepsilon + 1] 0$$

$$= 1$$

Substituting these values in eqn (2) & eq (3)

$$R_{13}^{(2)} = R_{13}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{23}^{(1)}$$

$$= \phi + 1^* 0 [11^* 0]^* 0$$

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

$$R_{33}^{(2)} = R_{33}^{(1)} + R_{32}^{(1)} [R_{22}^{(1)}]^* R_{23}^{(1)}$$

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

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

Substituting these values in eqn (1).

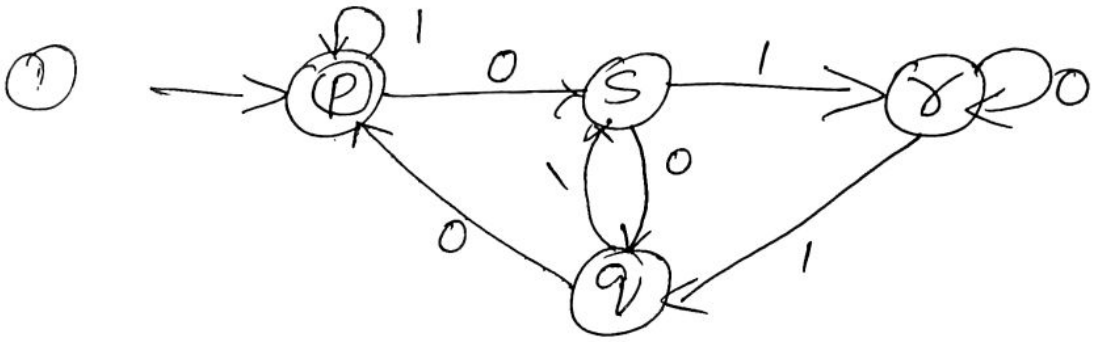
$$R_{13}^{(3)} = R_{13}^{(2)} + R_{13}^{(2)} [R_{33}^{(2)}]^* R_{33}^{(2)}$$

$$= 1^* 0 (11^* 0)^* 0 + 1^* 0 (11^* 0)^* 0 [(\varepsilon + 0) + 1 (11^* 0)^* (\varepsilon + 0 + 1 (11^* 0)^* 0)$$

Convert the DFA to regular expression using state elimination method.

State elimination method: we have to eliminate all the intermediate states expect the start state and the final state.

Formula: $R_{ij} + Q_i S^* P_j$



Step 1:
Eliminate the intermediate state 's' from

State p:

Formula: $R_{ij} + Q_i S^* P_j$

R_{ij} → Direct / Self loop

$Q_i S^* P_j$ → Indirect transition ⟹ through the intermediate state S

From p:

$p - s - p \Rightarrow 1 + \phi \Rightarrow 1$ (from p to s we have 0 but from s to p we dont have so ϕ)

$p - s - q \Rightarrow \phi + 0.\phi^*.0 = 00$

$p - s - r \Rightarrow \phi + 0.\phi^*.1 = 01$

$\phi^* = \epsilon$

From q:

$q - s - q \Rightarrow \phi + 10 = 10$

$q - s - p \Rightarrow 0 + \phi = 0$

$q - s - r \Rightarrow 0 + \phi = \phi 1$

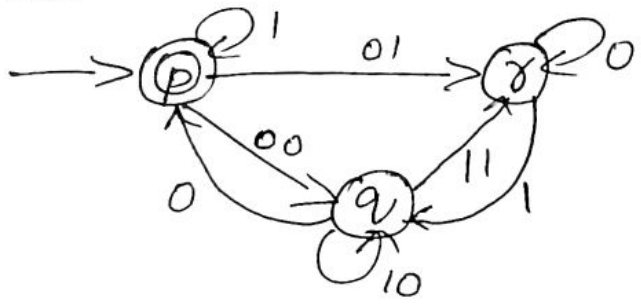
From r:

$r - s - r \Rightarrow 0 + \phi = 0$

$r - s - p \Rightarrow \phi + \phi = \phi$

$r - s - q \Rightarrow 1 + \phi = 1$

After eliminating the state 's' the automata will be



Step 2: Eliminate the intermediate state 'q'

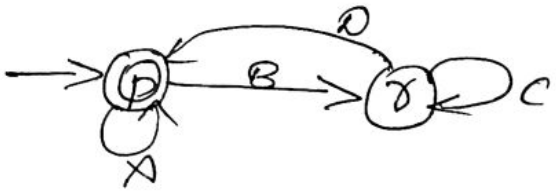
Renaming

$p \to q - p \Rightarrow 1 + 00.(10)^* 0 \Rightarrow A$

$p - q - r \Rightarrow 01 + 00.(10)^* 11 \Rightarrow B$

$r - q - r \Rightarrow 0 + 1(10)^* 11 \Rightarrow C$

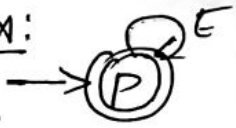
$r - q - p \Rightarrow \phi + 1.(10)^*.0 \Rightarrow D$



Step 3: Eliminate the state 'r'

$p - r - p \Rightarrow A + B.C^*.D \Rightarrow E$

DFA:



~~This~~ is the final regular expression

$E \Rightarrow A + B\,C^*\,D$

$P.E \Rightarrow 1 + 00.(10)^* 0 + 01 + 00.(10)^* 11.0 + 1(10)^* 11 . \phi + 1.(10)^*.0$

2)

step 1: Eliminate the intermediate state q_2.

(i) $q_1 - q_2 - q_3$

$R = \phi, Q = 1, S = 1, P = 0$

Action

$\emptyset + 11^* 0$

$11^* 0$

(ii) $q_3 - q_2 - q_1$

$R=0,\ Q=1,\ S=1,\ P=\emptyset$

$R + Qs^* P$

$0 + 11^* \emptyset = 0$

(iii) $q_1 - q_2 - q_1$

$R=0, Q=1, S=1, P=\emptyset$

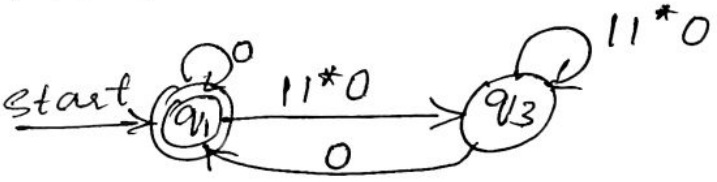
$0 + 11^* \emptyset = 0$

(iv) $q_3 - q_2 - q_3$

$R=\emptyset,\ Q=1,\ S=1,\ P=0$

$\emptyset + 11^* 0 = 11^* 0$

DFA:



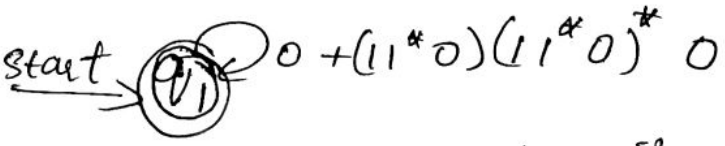
Step 2: Eliminate the state q_3

(i) $q_1 - q_3 - q_1$

$R=0,\ Q=11^*0,\ S=11^*0,\ P=0$

$R + QS^* P$

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



The final regular expression is

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

Theorem - 3.7 (Thomson construction Theorem)

Theorem: Every language defined by a regular expression is also defined by a finite Automata.

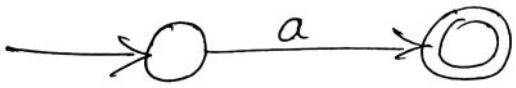
To prove: $L(M) = L(R)$ (set of strings accepted by deterministic finite automata will also be accepted by regular language)

Basis: There are three parts to the basis

(a) we see how to handle the expression ϵ. The language of the automaton is easily seen to be $\{\epsilon\}$, since the only path from the start state to an accepting state is labeled ϵ

(b) shows the construction for ϕ. clearly there are no paths from start state to accepting state, so ϕ is the language of this automation.

(c) It shows the automaton for a regular exp a. The language of this automaton evidently consists of the one string a, which is also $L(a)$.



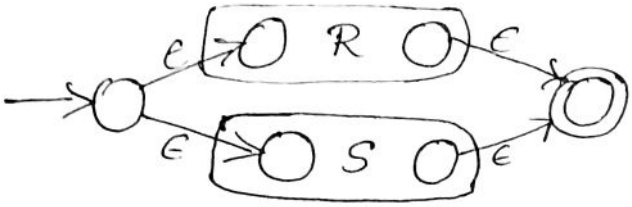
It is easy to check that these automata all satisfy conditions (1), (2) and (3) of the inductive hypothesis

Induction: We assume that the statement of the theorem is true for the immediate subexpressions of a given regular expression; ie, the language of these subexpressions are also the languages of ϵ-NFA's with a single accepting state.

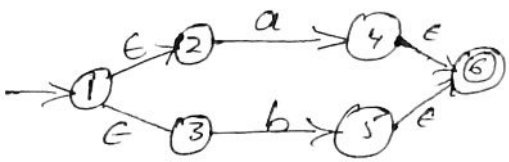
The 4 cases are :-

Case 1 : The expression is R+S for some smaller exp R & S.

The automaton for union is



example : R = a + b



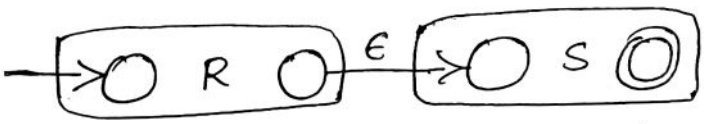
ie, starting at the new start state, we can go to the start state of either the automaton R or the automaton S. We then reach the accepting state of one of these automata, following a path labeled by some string L(R) or L(S) respectively.

Once we reach the accepting state of the automaton for R or S, we can follow one of the ε-arcs to the accepting state of the new automaton.

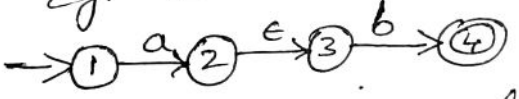
Thus the language of the automaton is L(R) ∪ L(S)

Case 2 : The expression is RS for some smaller exp R and S.

The automaton for the concatenation is



Eg: R = ab



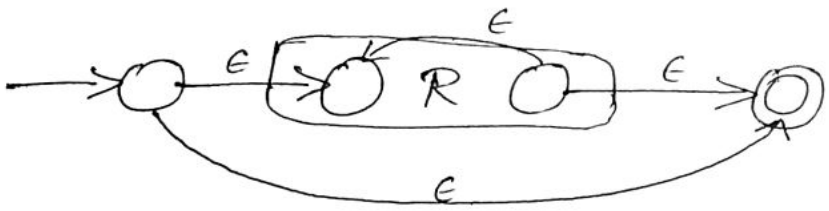
The start state of the first automaton becomes the start state of the whole, and the accepting state of the second automaton becomes the accepting state of the whole.

The idea is that only paths from start to accepting state go first through the automaton for R, where it must follow a path labeled by a string in L(R), & then through the automaton for S, where it follows a path labeled by a string in L(S)

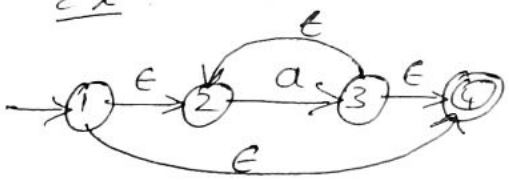
Thus, the paths in automaton are all and only labeled by strings in L(R)L(S).

Case 3: The expression is R^* for some smaller exp R.

Then we use the automaton of



Ex: $R = a^*$



This automaton allows us to go either:

(a) Directly from the start state to the accepting state along a path labeled ϵ. That path lets us accept ϵ, which is in $L(R^*)$ no matter what expression R is

(b) To the start state of the automaton for R, through that automaton one or more times, & then to the accepting state.

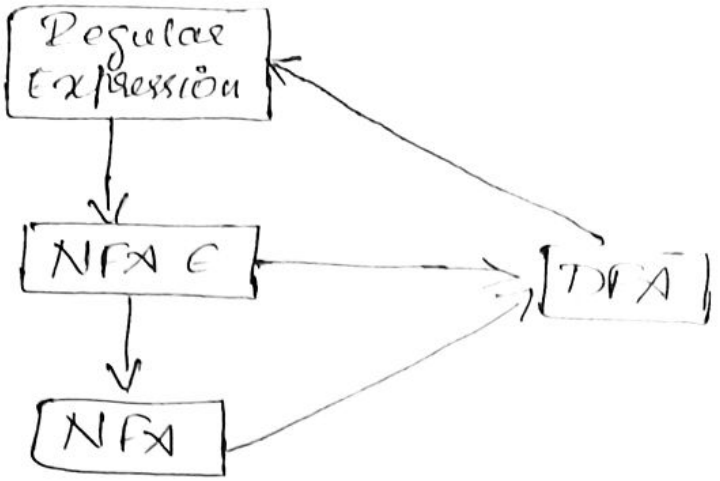
This set of paths allows us to accept strings in $L(R)$, $L(R)L(R)$, $L(R)L(R)L(R)$, & so on, thus covering all strings in $L(R^*)$ except perhaps ϵ,

Case 4: The expression is (R) for some smaller exp R.

The automaton for R also serves as the automaton for (R), since the parentheses do not change the language defined by the expression.

Hence pt is proved from the inductive hypotheses — one accepting state, with no arcs into the initial state or out of the accepting state.

Equivalence of FA & RE



1) Conversion of Regular expression to NFA ϵ
   ↳ Thompson's method

2) Conversion of Finite Automata to RE
   - Formula method (K)
   - Rij
   - State elimination method

Equivalence means is it possible to convert from one machine to the another & which automata is the subset of another.

We know,
DFA is the subset of NFA
NFA is the subset of NFA ϵ

& It is possible to convert NFA machine in to DFA
11/4 NFA ϵ can be directly converted to DFA or the NFA ϵ can be converted to NFA & then to DFA

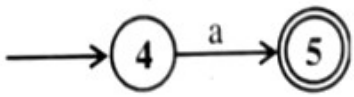
Whereas,
Conversion of DFA to NFA or ϵNFA is Not possible.

Note: • RE can be converted to NFA ϵ only the NFA ϵ again can be converted to NFA or DFA
• Finite automata (DFA, NFA, ϵNFA) can be converted to the Regular expression.

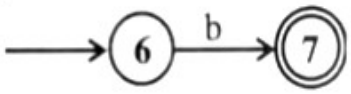
(1) Obtain an NFA which accepts strings of a's and b's starting with the string ab.

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

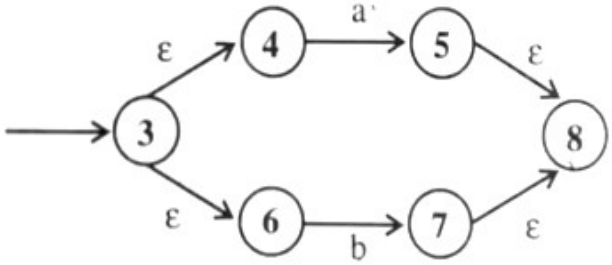
Step 1: The machine to accept 'a' is shown below.



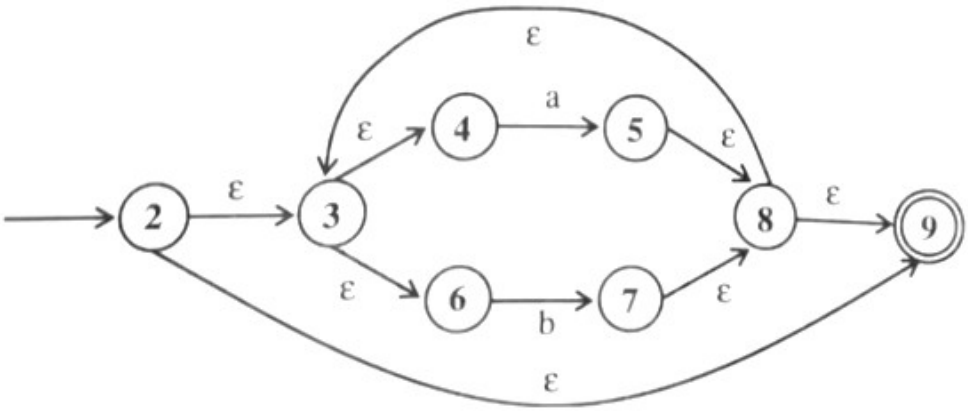
Step 2: The machine to accept 'b' is shown below.



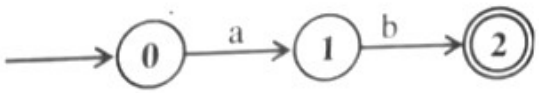
Step 3: The machine to accept (a + b) is shown below.



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.

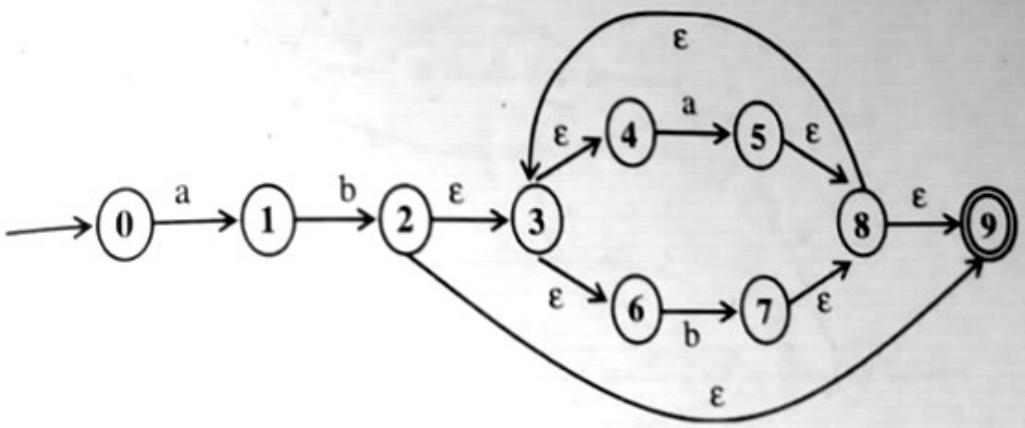
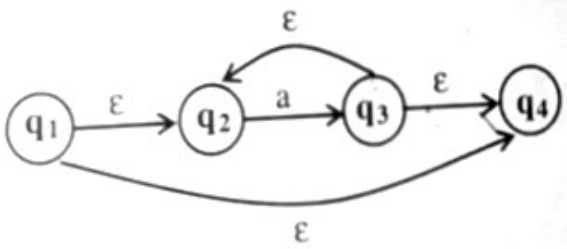


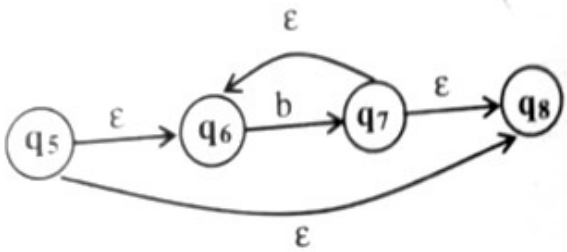
Figure 2.7. To accept the language L(ab(a+b)*)

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

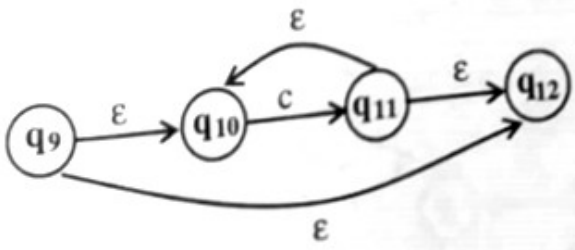
The machine corresponding the regular expression a* can be written as



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.

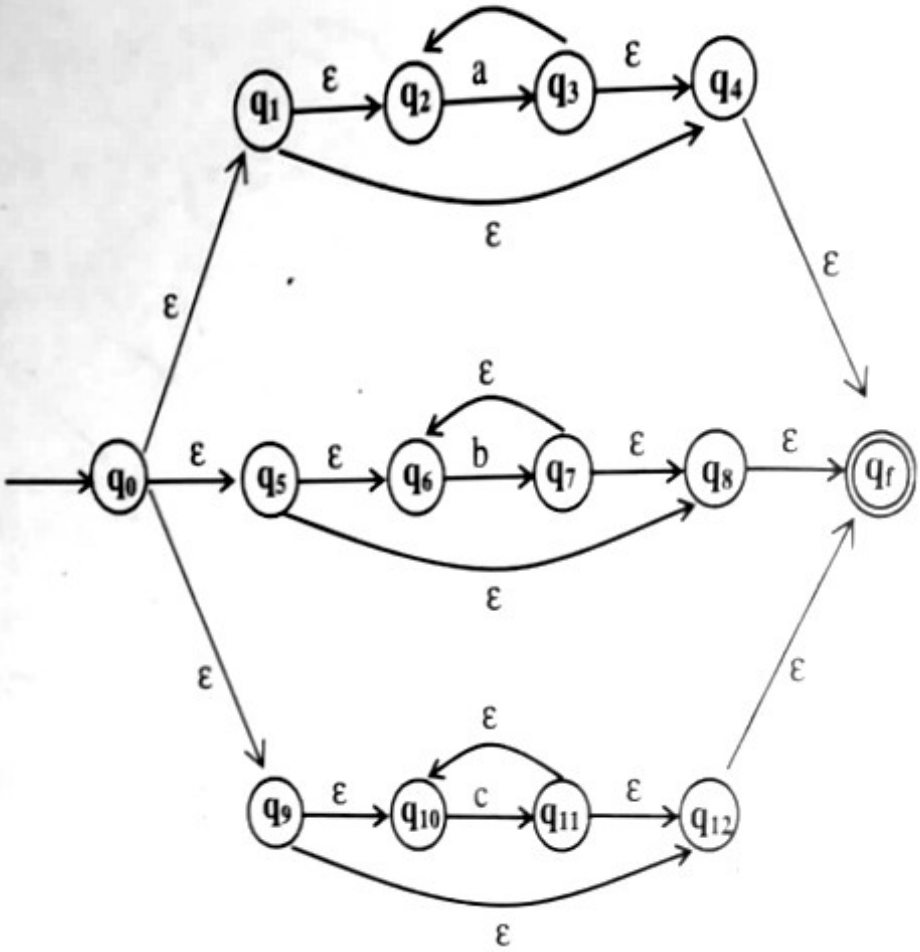
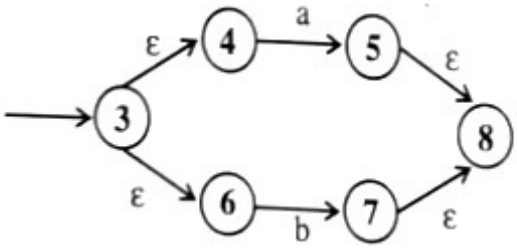


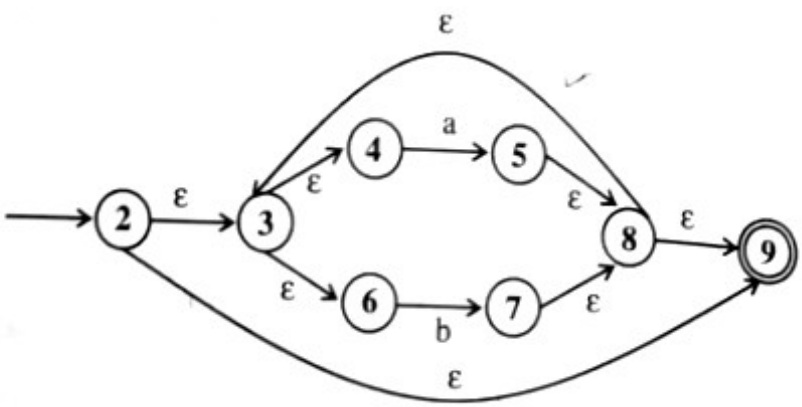
Figure 2.8. To accept the language L(a* + b* + c*)

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

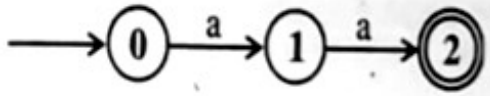
Step 1: The machine to accept (a + b) and (a+b)* are shown below:



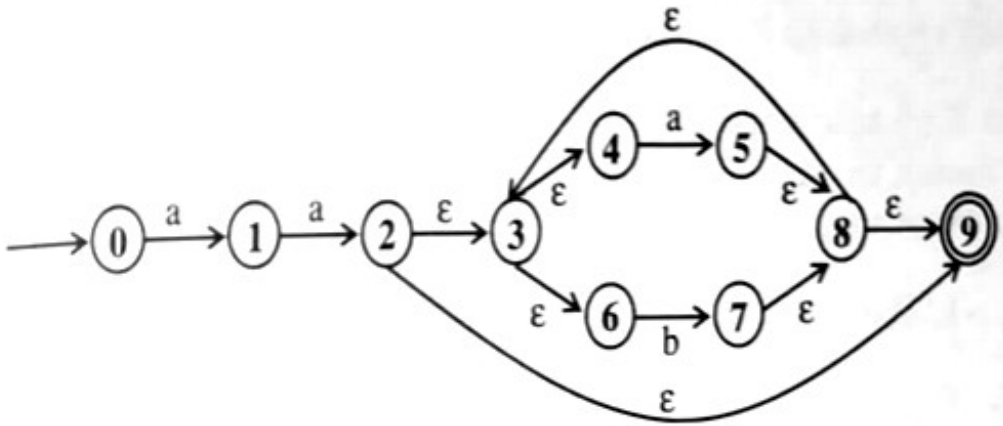
Step 2: The machine to accept (a+b)* is shown below:



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.

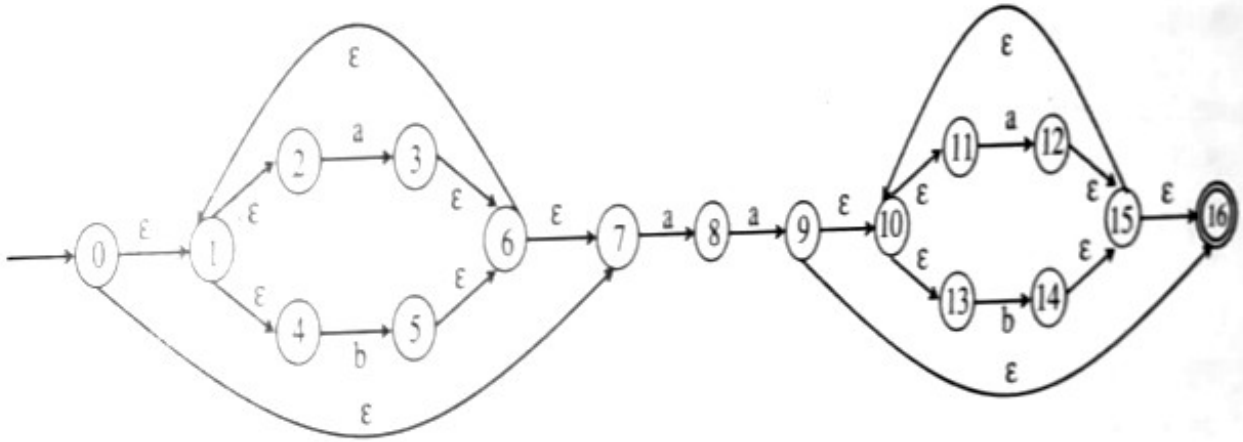


Figure 2.9. NFA to accept (a+b)*aa(a+b)*

Regular Languages and Properties of Regular Languages

What are we studying in this chapter . . .

- *Regular languages*
- *Proving languages not to be regular languages*
- *Closure properties of regular languages*
- *Decision properties of regular languages*
- *Equivalence and minimization of automata*

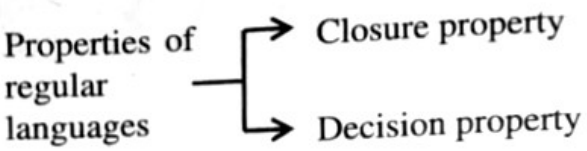
3.1. Proving Languages Not to be Regular

We have already described regular languages in previous chapters. For easy understanding let us see once again "What is a regular language?"

❖ **Definition:** The class of languages known as regular languages have four different descriptions. The regular languages are the languages accepted by DFA's, NFA's and ε-NFA's and defined by regular expressions. We know that each FA can be expressed in terms of other and all of them

accept the same languages. Thus, we can define regular languages as the languages accepted by DFA's or NFA's or ε-NFA's.

Once we know the definition of regular language, let us see "What are the various properties of regular languages?" The two important properties of regular languages are shown below:



- Closure property of regular languages: This property is a useful tool for building complex automata. Using the closure property we can build language recognizers. The closure property helps us to construct more complex finite automata.

  For example, the intersection of two regular languages is also regular. So, if there are two automata recognizing two different languages, we can construct an automaton that recognizes the intersection of these two languages.

- Decision property of regular languages: Using this property, we can decide whether two automata define the same language. If so, we can minimize the states of automata with as few states as possible. The minimization of automaton is very important in the design of switching circuits. This is because as the number of states of automaton decreases the size of the circuit and hence the cost decreases.

Any finite language can be expressed using regular expressions and we can construct finite automata (DFA or NFA or ε-NFA). Though the regular languages are important, there are non-regular languages which are very interesting and important. For example, following are some of the non-regular languages:

1. $L = \{w : w \in \{0, 1\}^*$ and has equal number of 0's and 1's $\}$
2. $L = \{0^n 1^n$ for $n \geq 0\}$
3. $L = \{a^p \in$ where $p \geq 2$ is a prime number $\}$
4. Language consisting of matching parentheses

Apart from above four languages, there are so many languages which are not regular. Now, the question is "How to prove that certain languages are not regular?" We can prove that certain languages are not regular using one powerful tool called pumping lemma.

3.2. Pumping Lemma for Regular Languages

Now, let us "State and prove pumping lemma for regular languages".

Theorem 4.1: (The *pumping lemma for regular languages*) Let L be a regular language. Then there exists a constant n (which depends on L) such that for every string w in L such that $|w| \geq n$, we can break w into three strings, $w = xyz$, such that:

1. $y \neq \epsilon$.
2. $|xy| \leq n$.
3. For all $k \geq 0$, the string $xy^k z$ is also in L.

That is, we can always find a nonempty string y not too far from the beginning of w that can be "pumped"; that is, repeating y any number of times, or deleting it (the case $k = 0$), keeps the resulting string in the language L.

PROOF: Suppose L is regular. Then $L = L(A)$ for some DFA A. Suppose A has n states. Now, consider any string w of length n or more, say $w = a_1 a_2 \cdots a_m$, where $m \geq n$ and each a_i is an input symbol. For $i = 0, 1, \ldots, n$ define state p_i to be $\hat{\delta}(q_0, a_1 a_2 \cdots a_i)$, where δ is the transition function of A, and q_0 is the start state of A. That is, p_i is the state A is in after reading the first i symbols of w. Note that $p_0 = q_0$.

By the pigeonhole principle, it is not possible for the $n+1$ different p_i's for $i = 0, 1, \ldots, n$ to be distinct, since there are only n different states. Thus, we can find two different integers i and j, with $0 \leq i < j \leq n$, such that $p_i = p_j$. Now, we can break $w = xyz$ as follows:

1. $x = a_1 a_2 \cdots a_i$.
2. $y = a_{i+1} a_{i+2} \cdots a_j$.
3. $z = a_{j+1} a_{j+2} \cdots a_m$.

That is, x takes us to p_i once; y takes us from p_i back to p_i (since p_i is also p_j), and z is the balance of w. The relationships among the strings and states are suggested by Fig. 4.1. Note that x may be empty, in the case that $i = 0$. Also, z may be empty if $j = n = m$. However, y can not be empty, since i is strictly less than j.

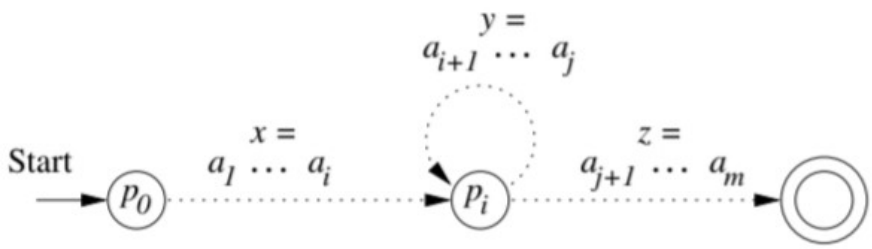


Figure 4.1: Every string longer than the number of states must cause a state to repeat

Now, consider what happens if the automaton A receives the input $xy^k z$ for any $k \geq 0$. If $k = 0$, then the automaton goes from the start state q_0 (which is also p_0) to p_i on input x. Since p_i is also p_j, it must be that A goes from p_i to the accepting state shown in Fig. 4.1 on input z. Thus, A accepts xz.

If $k > 0$, then A goes from q_0 to p_i on input x, circles from p_i to p_i k times on input y^k, and then goes to the accepting state on input z. Thus, for any $k \geq 0$, $xy^k z$ is also accepted by A; that is, $xy^k z$ is in L. $\square$

Pumping lemma problems

1) prove that $L = \{0^n 1^n / n \geq 1\}$ is not regular

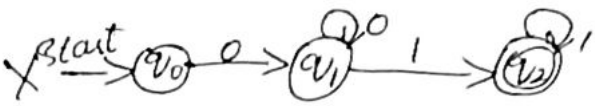
* Assume L is regular
* Let n be a constant
* Let $w = 0^n 1^n$ $w \geq n$
* Split $w = xyz$ such that
  1. $y \neq \epsilon$
  2. $|xy| \leq n$
  3. For all $k \geq 0$, $xy^k z \in L$

Assume $k = 2$, $xy^2z = 00011 \notin L$

Contradiction!

Hence $L = \{0^n 1^n / n \geq 1\}$ is not regular

01, 0011, 000111...



if $L = \{0^n 1^m / n > 1\}$ then this DFA was correct

$w = 0^n 1^n$

if $n = 2$ $w = 0011$

$|xy| \leq n$ - $xy = 00$ - rule 2

$y = 0$ as $|y| > 0$ - rule 3

$x = 0$, $y = 0$

z -> remaining symbols of w $z = 11$

2) prove that $L = \{a^i b^j / i \leq j\}$ is not regular

* Assume that L is regular
* Let n be the number of states of DFA
* Let $w = a^n b^{n+1}$, $|w| \geq n$
* Split $w = xyz$ such that
  1. $y \neq \epsilon$
  2. $|xy| \leq n$
  3. for all $k \geq 0$, $xy^k z \in L$

$\{aabb, aabbb, aabbbb$
$abb..\}$

(n number of a's followed by atleast n+1 no of steings

Let $n = 2$, $w = aabbb$

$xy = aa$

$x = a$, $y = a$, $z = bb$

prove for all $k \geq 0$, $xy^k z \in L$

Assume $k = 2$, $xy^2z = aaaabbb \in L$

$k = 3$, $xy^3z = aaaaabbb \notin L$

Contradiction!

Hence L is not regular

3) prove $L = \{1^p / p$ is prime number$\}$ is not regular using pumping lemma

length of 1 should be prime no
$\{1, 3, 7, 13, \ldots\}$
$\{1, 111, 1111111$

* assume that given language is regular
* Let n be a constant
* Let $w = xyz$, $|xyz| = p$ $|y| \geq 1$

proof: $|xy^i z| \neq p$ assume $i = p+1$

* assume $|xy^{p+1}z| = |xyz| + |y|^p$

$= p + p[|y|]$

$= p[1 + |y|]$ $p * 2/3$ shows it is not prime no should be multiply by 1 or itself

$|xy^i z| = |xyz| + |y^{i-1}|$

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

$\leq p + (i-1)m$

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

$= q + qm - q(1+m)$

4) prove that $L = \{0^{j^2} / j \geq 1\}$ is not regular

if $j = 1^2, 2^2 = 4, 3^2 = 9$

$L = \{0, 0000, 000000000 \ldots\}$

* assume L is regular
* Let n be a constant
* $w = 0^{p^2} = 0^n$ where $n = p^2$
* split $w = xyz$ with
  1. $y \neq \epsilon$
  2. $|xy| \leq n$
  3. for all $k \geq 0$, $xy^k z \in L$

* $w = 0^n$

$xy = 0^m$ where $m \leq n$

$y = 0^r$ where $r < m$

$z = 0^{n-m}$

* $xy^k z = x\, y\, y^{k-1} z$

$= 0^m (0^r)^{k-1} 0^{n-m}$

$= 0^{m + rk - r + n - m}$

$= 0^{r(k-1) + n}$

Pick $k = 1$, $xy^1 z = 0^n = 0^{p^2} \in L$

Pick $k = 2$, $xy^2 z = 0^{r+n} = 0^{r+p^2} \notin L$ Contradiction. Hence $L = \{0^{j^2} / j \geq 1\}$ is not regular

4.2 Closure Properties of Regular Languages

In this section, we shall prove several theorems of the form "if certain languages are regular, and a language L is formed from them by certain operations (e.g., L is the union of two regular languages), then L is also regular." These theorems are often called *closure properties* of the regular languages, since they show that the class of regular languages is closed under the operation mentioned. Closure properties express the idea that when one (or several) languages are regular, then certain related languages are also regular. They also serve as an interesting illustration of how the equivalent representations of the regular languages (automata and regular expressions) reinforce each other in our understanding of the class of languages, since often one representation is far better than the others in supporting a proof of a closure property. Here is a summary of the principal closure properties for regular languages:

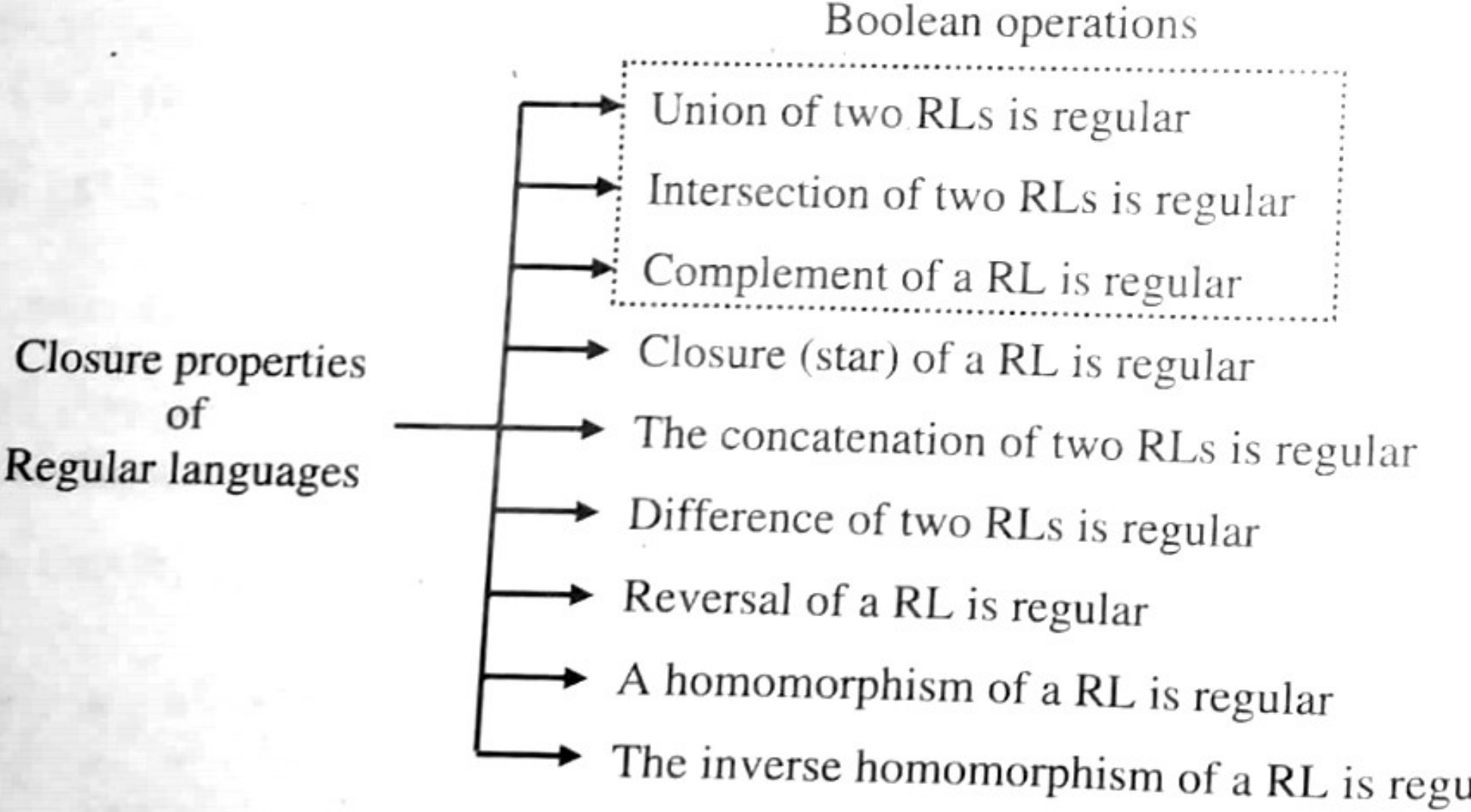
1. The union of two regular languages is regular.
2. The intersection of two regular languages is regular.
3. The complement of a regular language is regular.
4. The difference of two regular languages is regular.
5. The reversal of a regular language is regular.
6. The closure (star) of a regular language is regular.
7. The concatenation of regular languages is regular.
8. A homomorphism (substitution of strings for symbols) of a regular language is regular.
9. The inverse homomorphism of a regular language is regular.

4.2.1 Closure of Regular Languages Under Boolean Operations

Our first closure properties are the three boolean operations: union, intersection, and complementation:

1. Let L and M be languages over alphabet Σ. Then $L \cup M$ is the language that contains all strings that are in either or both of L and M.
2. Let L and M be languages over alphabet Σ. Then $L \cap M$ is the language that contains all strings that are in both L and M.
3. Let L be a language over alphabet Σ. Then $\overline{L}$, the *complement* of L, is the set of strings in Σ^* that are not in L.

It turns out that the regular languages are closed under all three of the boolean operations. The proofs take rather different approaches though, as we shall see.


Boolean operations
Union of two RLs is regular
Intersection of two RLs is regular
Complement of a RL is regular
Closure properties
of
Regular languages
Closure (star) of a RL is regular
The concatenation of two RLs is regular
Difference of two RLs is regular
Reversal of a RL is regular
A homomorphism of a RL is regular
The inverse homomorphism of a RL is regu

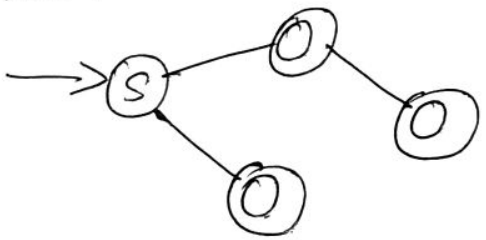
1) Closure under union:

If L and M are Regular Language, so their union is defined by:

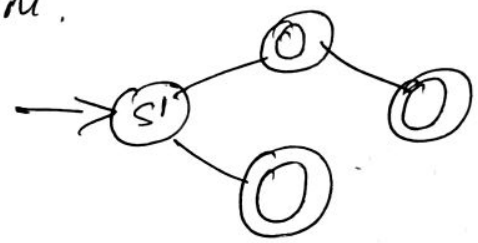
$$L \cup M = \{w : w \in L \text{ or } w \in M\}$$

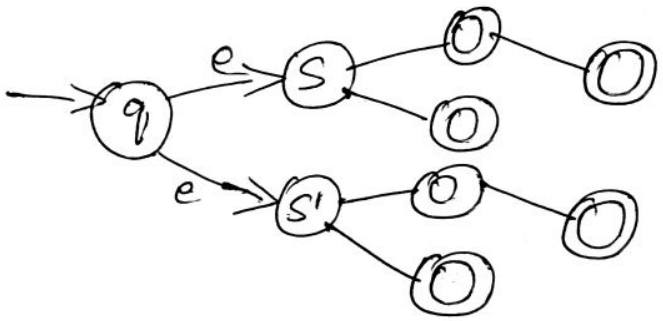
Let A_L & A_M two finite automata accepting L and M respectively.

A_L:



A_M:





Ex: If a and b are 2 integer Then a+b is also an integer so + operation is closed under addition

→ Let L & M be lang of R.E R & S respectively then R+S is a R.E whose language is $L \cup M$

2) closure under Kleen closure & concatination

⇒ R.S are regular expression whose language is $L \cdot M$

⇒ R^* is a regular expression whose language is L^*

3) Closure under Intersection:

If L & M are regular language, so there intersection is defined by

$$L \cap M = \{w : w \in L \text{ and } w \in M\}$$

using Demorgan's Law

$$L \cap M = (L^c \cup M^c)^c \qquad c - \text{complement}$$

Then their must be FA accepting $(L \cap M)$

4) Closure under Difference

Let L & M be Two language, then their difference is defined by

$$L - M = \{w : w \in L \ \& \ w \notin M\}$$

using

Demorgan's laws:

$$(L - M) = L \cap M^c$$

Thus, there must be a finite automata for $(L-M)$

4) Closure under complementation:

~~Let L and M be two languages, then their~~

~~language is defined by:~~

The complement of a language L (with respect to an alphabet Σ such that Σ^* contains L) is

$$\boxed{\Sigma^* - L} \qquad \Sigma = \{0, 1\}, \{a \text{---} z\}$$

Since Σ^* is surely regular, the complement of a regular language is always regular.

5) Closure under Reversal:

Given language L, L^R is the set of strings where reversal is in L.

$L = \{0, 01, 100\}$; $L^R = \{0, 10, 001\}$ ← Reverse

Basis: If E is a symbol a, ϵ, or ϕ then $E^R = E$

Induction: If E is

→ $F + Q$ then $E^R = F^R + Q^R$ (Reverse of E, Reverse of F, Reverse of Q)

→ FQ, then $E^R = Q^R F^R$

→ F^*, then $E^R = (F^R)^*$

Ex: Let $E = 01^* + 10^*$ the reverse of E is

$E^R = (01^* + 10^*)^R = (01^*)^R + (10^*)^R$

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

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

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

6)

Minimization of DFA

Minimization of DFA is required to obtain the minimal version of any DFA which consists of the minimum number of states possible

The minimization means reducing the number of states from the finite Automata

→ Two state can be combined when these two state are equivalent.

Two states 'A' and 'B' are said to be equivalent if:

$\delta(A, X) \longrightarrow F$ OR $\delta(A, X) \longleftrightarrow F$ and $\delta(B, X) \longleftrightarrow F$

$\delta(B, X) \longrightarrow F$

where X is any input string

Types of equivalence:

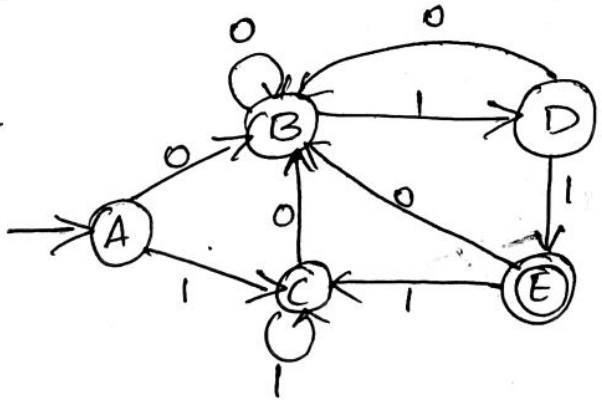
If $|X| = 0$, then A and B are said to be 0 equivalent

If $|X| = 1$, then A and B are said to be 1 equivalent

If $|X| = 2$, then A and B are said to be 2 equivalent

If $|X| = n$, then A and B are said to be n equivalent

Ex:



Step 1: State Transition table

	0	1
→A	B	C
B	B	D
C	B	C
D	B	E
*E	B	C

Step 2: → Find the 0 Equivalence (write the non final state together as one set & final state as another set)

{A, B, C, D} {E}

→ Find the 1 Equivalence (Look at 0 equivalence & check these sets are 1 equivalence to each other)

{A, B, C} {D} {E}

Ex:

A	B	A	B
i/p 0	i/p 0	i/p 1	i/p 1
B	B	C	D

A B ✓ {B & B, C & D belongs to same set so they are 1 equivalent}

A C ✓ {C & A also going to same state so C is also 1 equivalent of A & B}

C D ✗ {D cannot be checked with either A, B or C because all are equivalent to each other}

~~Steps~~ → Find the 2 Equivalence

{A, C} {B} {D} {E}

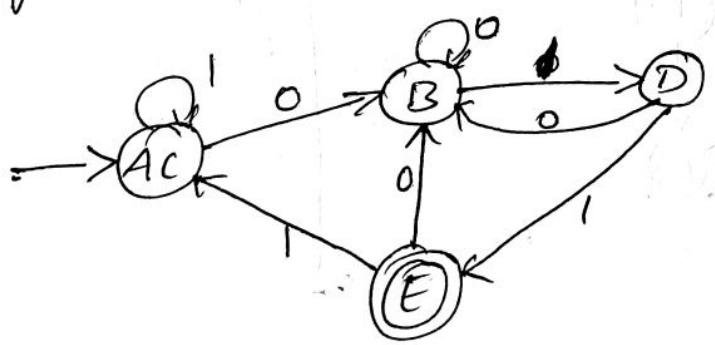
{Now we have to check A B & C}
{check A & B upon i/p 0 & 1 upon i/p 1 A goes to C, B goes to D see C & D belongs to same set or not, as C & D are of different sets they are not equivalent}

→ Find the 3 Equivalence

{A, C} {B} {D} {E}

{Now check for A, C they both going to same state upon i/p 0 & i/p 1 so they are equivalent}

Now the states are minimized & got ~~total~~ 4 states



3.6.1. Equivalence of Two States

The language generated by a DFA is unique. But, there can exist many DFA's that accept the same language. In such cases, the DFA's are said to be equivalent. During computations, it is desirable to represent the DFA with fewer states since the space is proportional to the number of states of DFA. For storage efficiency, it is required to reduce the number of states and hence it is required to minimize the DFA. This can be achieved first by finding the distinguishable and indistinguishable states. First, let us see "What are distinguishable and indistinguishable states?".

❖ **Definition**: Two states p and q of a DFA are *equivalent* (*indistinguishable*) if and only if $\delta(p, w)$ and $\delta(q, w)$ are final states or both $\delta(p, w)$ and $\delta(q, w)$ are non-final states for all $w \in \Sigma^*$ i.e. if

$$\delta(p, w) \in F \text{ and } \delta(q, w) \in F$$

then the states p and q are *indistinguishable*. If

$$\delta(p, w) \notin F \text{ and } \delta(q, w) \notin F$$

then also the states p and q are *indistinguishable*. If there is at least one string w such that one of

$$\delta(p, w) \text{ and } \delta(q, w)$$

is final state and the other is non-final state, then the states p and q are not equivalent and are called *distinguishable* states.

Note: The *distinguishable* and *indistinguishable* states can be obtained using ***table-filling algorithm*** (also called *mark* procedure).

3.6.2. Table Filling Algorithm

Now, let us see "What is table filling algorithm?" The table filling algorithm is used to find the set of states that are distinguishable and indistinguishable states. The algorithm is recursively defined as shown below:

Step 1: Identify the initial markings: For each pair (p, q) where $p \in Q$ and $q \in Q$, if $p \in F$ and $q \notin F$ or vice versa then, the pair (p, q) is distinguishable and mark the pair (p, q) [by putting say 'x' for the pair (p,q)].

Step 2: Identify the subsequent markings: For each pair (p, q) and for each $a \in \Sigma$, find $\delta(p, a) = r$ and $\delta(q, a) = s$. If the pair (r, s) is already marked as distinguishable then the pair (p, q) is also distinguishable and mark it as say 'x'. Repeat step 2 until no previously unmarked pairs are marked.

Note: If the pair (p, q) obtained using the above *table filling algorithm* (mark procedure) are indistinguishable then the two states *p* and *q* are equivalent and they can be merged into one state thus minimizing the states of DFA.

3.6.3. Minimization of DFA (Algorithm or Procedure)

Once we have found the distinguishable and indistinguishable pairs we can easily minimize the number of states of a DFA and accepting the same language as accepted by original DFA. Now, let us "Write the algorithm or the procedure to minimize a DFA using table filling algorithm".

The algorithm to minimize the DFA is shown below:

Step 1: Find the distinguishable and indistinguishable pairs: using the *table-filling* algorithm (discussed in previous section).

Step 2: Obtain the states of minimized DFA: These groups consist of indistinguishable pairs obtained in previous step and individual distinguishable states. The groups obtained are the states of minimized DFA.

Step 3: Compute the transition table: If $[p_1, p_2, \ldots, p_k]$ is a group and if $\delta([p_1, p_2, \ldots, p_k], a) = [r_1, r_2, \ldots, r_m]$ then place an edge from the group $[p_1, p_2, \ldots, p_k]$ to the group $[r_1, r_2, \ldots, r_m]$ and label the edge with the symbol *a*. Follow this procedure for each group obtained in step 2 and for each $a \in \Sigma$.

Step 4: Identify the start state: If one of the component in the group $[p_1, p_2, \ldots, p_k]$ consists of a start state of given DFA then $[p_1, p_2, \ldots, p_k]$ is the start state of minimized DFA.

Step 5: Identify the final state: If the group $[p_1, p_2, \ldots, p_k]$ contains a final state of given DFA then the group $[p_1, p_2, \ldots, p_k]$ is final state of minimized DFA.

Example 1: Obtain the distinguishable table for the automaton and then minimize the states of following DFA.

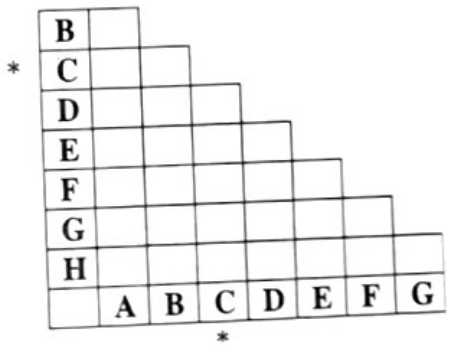
δ	a	b
→A	B	F
B	G	C
*C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

Solution: The DFA can be minimized using table filling algorithm as shown below:

The various states of given DFA are: A, B, C, D, E, F, G, H.

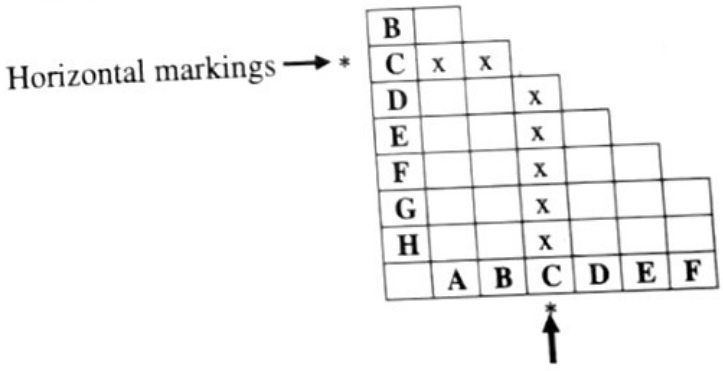
Step 1: Obtain the various pairs of states: The various pairs from the above states can be obtained by drawing the table shown below:

- Vertically, we write all the states from second state to the last state. In this case, we write the states B, C, D, E, F, G and H vertically (see table below)
- Horizontally, we write all the states from first state to last but one state. In this case, we write the states A, B, C, D, E, F and G (see table below)
- Then draw the vertical and horizontal lines as shown in the table below:



Initial Horizontal markings: Since state C is the final state, the pairs (A,C) and (B,C) has o final state and other non-final state. So, we mark the pairs (A,C) and (B,C) horizontally (see t marking x horizontally).

Initial Vertical markings: Since state C is the final state, the pairs (H,C), (G,C), (F,C), (E, (D,C) has one final state and non-final state. So, they are marked vertically (see the markin vertically) as shown in table below:



Step 2: For each pair (p, q) and for each $a \in \Sigma$ find $\delta(p, a) = r$ and $\delta(q, a) = s$. If the pair (r, s) is already marked as distinguishable then the pair (p, q) is also distinguishable and mark it as 'x'. The various markings can be obtained as shown below:

δ	a	b	
(A,B)	(B,G)	(F,C)	(F,C) is marked. So, mark (A, B)
(A,D)	(B,C)	(F,G)	(B,C) is marked. So, mark (A, D)
(A,E)	(B,H)	(F,F)	
(A,F)	(B,C)	(F,G)	(B,C) is marked. So, mark (A, F)
(A,G)	(B,G)	(F,E)	
(A,H)	(B,G)	(F,C)	(F, C) is marked. So, mark (A,H)
(B,D)	(G,C)	(C,G)	(G, C) is marked. So, mark (B,D)
(B,E)	(G,H)	(C,F)	(C, F) is marked. So, mark (B,E)
(B,F)	(G,C)	(C,G)	(G, C) is marked. So, mark (B, F)
(B,G)	(G,G)	(C,E)	(C, E) is marked. So, mark (B, G)
(B,H)	(G,G)	(C,C)	
(D,E)	(C,H)	(G,F)	(C, H) is marked. So, mark (D, E)
(D,F)	(C,C)	(G,G)	
(D,G)	(C,G)	(G,E)	(C, G) is marked. So, mark (D, G)
(D,H)	(C,G)	(G,C)	(C, G) is marked. So, mark (D,H)
(E,F)	(H,C)	(F,G)	(H, C) is marked. So, mark (E, F)
(E,G)	(H,G)	(F,E)	
(E,H)	(H,G)	(F,C)	(F,C) is marked. So, mark (E,H)
(F,G)	(C,G)	(G,E)	(C, G) is marked. So, mark (F, G)
(F,H)	(C,G)	(G,C)	(C, G) is marked. So, mark (F, H)
(G,H)	(G,G)	(E,C)	(E, C) is marked. So, mark (G, H)

The pairs shown in black background are to be marked

	A	B	C	D	E	F	G
B							
* C	x	x					
D			x				
E			x				
F			x				
G			x				
H			x				

*

The resulting marking table

	A	B	C	D	E	F	G
B	x						
* C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G		x	x	x		x	
H	x		x	x	x	x	x

*

Again consider the pairs (p,q) which are not marked in the above table and see whether the corresponding pairs (r, s) on 0 and 1 are marked as shown below:

	a	b
(p,q)	**(r,s)**	**(r,s)**
(A,E)	(B,H)	(F,F)
(A,G)	**(B,G)**	**(F,E)**
(B,H)	(G,G)	(C,C)
(D,F)	(C,C)	(G,G)
(E,G)	**(H,G)**	**(F,E)**

Mark the pairs (A,G) and (E,G) with 'x' as shown below:

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x
	A	B	C	D	E	F	G

Indistinguishable pairs:
(A,E), (B,H) and (D,F)

Distinguishable pairs:
C and G

Minimizing DFA: The DFA can be minimized as shown below:

Step 1: Find the distinguishable and indistinguishable pairs:

(A,E), (B,H) and (D,F) are in-distinguishable

C, G are distinguishable (see previous page)

Step 2: Obtain the states of minimized DFA: The states of the minimized DFA

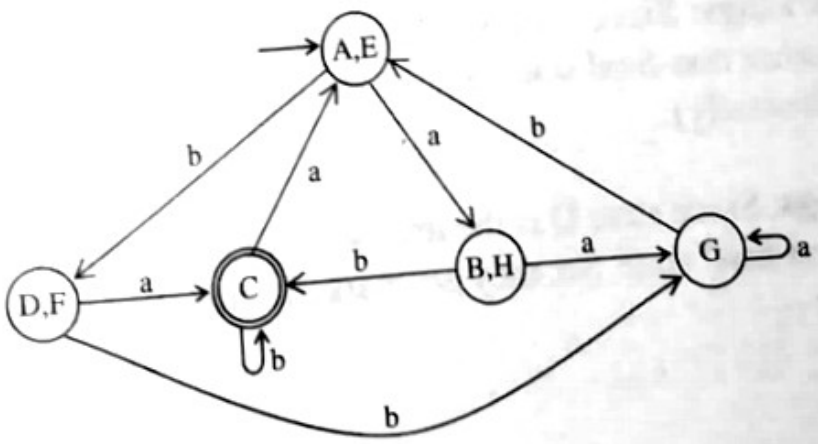
(A,E), (B,H), C, (D,F), G

Step 3: Compute the transition table: Use the transition table of given DFA and obtain the transitions for the minimized DFA as shown below:

δ	a	b
→ (A,E)	(B,H)	(D,F)
(B,H)	G	C
*C	(A,E)	C
(D,F)	C	G
G	G	(A,E)

Step 4: Identify the start state: Since the group (A,E) has the start state of DFA, the group (A, E is the start state of minimized DFA.

Step 5: Identify the final state: The group C is the final state of given DFA. So, C will be the fin state of minimized DFA. The transition diagram of the minimized DFA is shown below:



Example 2: Find the minimized DFA for the following:

δ	0	1
→ A	B	A
B	A	C
C	D	B
*D	D	A
E	D	F
F	G	E
G	F	G
H	G	D

Solution: The DFA can be minimized using table filling algorithm as shown below:

The various states of given DFA are: A, B, C, D, E, F, G, H.

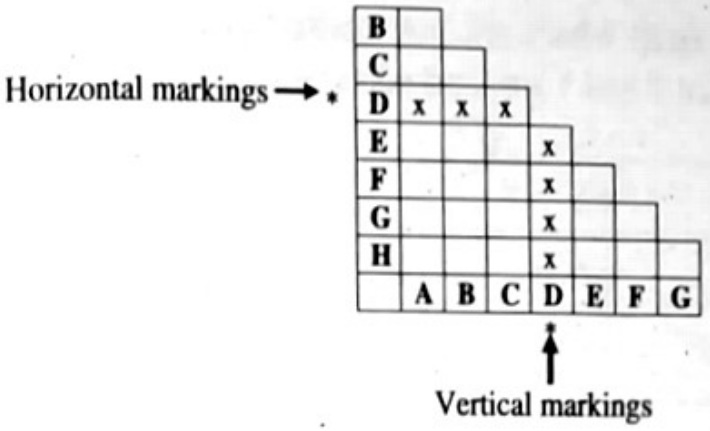
Step 1: Obtain the various pairs of states: The various pairs from the above states can be obtained by drawing the table shown below:

- Vertically, we write all the states from second state to the last state. In this case, we write the states B, C, D, E, F, G and H vertically (see table below)
- Horizontally, we write all the states from first state to last but one state. In this case, we write the states A, B, C, D, E, F and G (see table below)
- Then draw the vertical and horizontal lines as shown in the table below:

B							
C							
* D							
E							
F							
G							
H							
	A	B	C	D	E	F	G
				*			

Initial Horizontal markings: Since state D is the final state, the pairs (A,D), (B,D) and (C,D) has one final state and other non-final state. So, we mark the (A,D), (B,D) and (C,D) horizontally (see the marking x horizontally).

Initial Vertical markings: Since state D is the final state, the pairs (D,E), (D,F), (D,G) and (D,H) has one final state and non-final state. So, they are marked vertically (see the marking x vertically) as shown in table below:



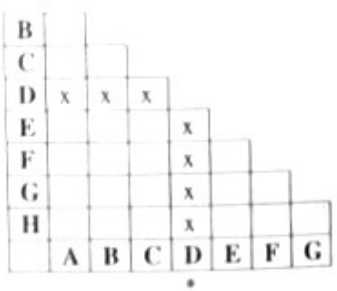
Step 2: For each unmarked pair (p, q) and for each $a \in \Sigma$ find

$\delta(p, a) = r$ and $\delta(q, a) = s$.

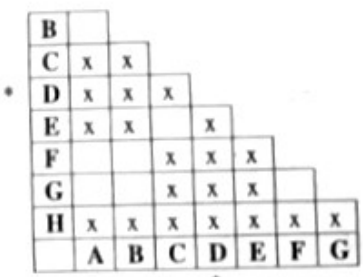
If the pair (r, s) is already marked as distinguishable then the pair (p, q) is also distinguishable and mark it as say 'x'. The various markings can be obtained as shown below:

δ	a	b	
(A,B)	(A,B)	(A,C)	
(A,C)	(B,D)	(A,B)	(B,D) is marked. So, mark (A, C)
(A,E)	(B,D)	(A,F)	(B,D) is marked. So, mark (A, E)
(A,F)	(B,G)	(A,E)	
(A,G)	(B,F)	(A,G)	
(A,H)	(B,G)	(A,D)	(A,D) is marked. So, mark (A, H)
(B,C)	(A,D)	(C,B)	(A,D) is marked. So, mark (B,C)
(B,E)	(A,D)	(C,F)	(A,D) is marked. So, mark (B,E)
(B,F)	(A,G)	(C,E)	
(B,G)	(A,F)	(C,G)	
(B,H)	(A,G)	(C,D)	(C,D) is marked. So, mark (B,H)
(C,E)	(D,D)	(B,F)	
(C,F)	(D,G)	(B,E)	(D,G) is marked. So, mark (C,F)
(C,G)	(D,F)	(B,G)	(D,F) is marked. So, mark (C,G)
(C,H)	(D,G)	(B,D)	(D,G) is marked. So, mark (C,H)
(E,F)	(D,G)	(F,E)	(D,G) is marked. So, mark (E,F)
(E,G)	(D,F)	(F,G)	(D,F) is marked. So, mark (E,G)
(E,H)	(D,G)	(F,D)	(D,G) is marked. So, mark (E,H)
(F,G)	(G,F)	(E,G)	
(F,H)	(G,G)	(E,D)	(E,D) is marked. So, mark (F,H)
(G,H)	(F,G)	(D,G)	(D,G) is marked. So, mark (G,H)

The pairs shown in black background are to be marked



The resulting marking table



Again consider the pairs (p,q) which are not marked in the above table and see whether the corresponding pairs (r, s) on 0 and 1 are marked as shown below:

	a	b	
(p,q)	(r,s)	(r,s)	
(A,B)	(B,A)	(A,C)	(A,C) is marked. So, mark (A,B)
(A,F)	(B,G)	(A,E)	(A,E) is marked. So, mark (A,F)
(A,G)	(B,F)	(A,G)	
(B,F)	(A,G)	(C,E)	
(B,G)	(A,F)	(C,G)	(C,G) is marked. So, mark (B,G)
(C,E)	(D,D)	(B,F)	
(F,G)	(G,F)	(E,G)	(E,G) is marked. So, mark (F,G)

Mark the pairs (A,B), (A,F), (B,G) and (F,G) with 'x' as shown below:

	B	x						
	C	x	x					
*	D	x	x	x				
	E	x	x		x			
	F	x		x	x	x		
	G		x	x	x	x	x	
	H	x	x	x	x	x	x	x
		A	B	C	D	E	F	G
					*			

Indistinguishable pairs:
(A,G), (B,F) and (C,E)

Distinguishable states:
D, H

Minimizing DFA: The DFA can be minimized as shown below:

Step 1: Find the distinguishable and indistinguishable pairs:

(A,G), (B,F) and (C,E) are in-distinguishable
D,H are distinguishable (see previous page)

Step 2: Obtain the states of minimized DFA: The states of the minimized DFA

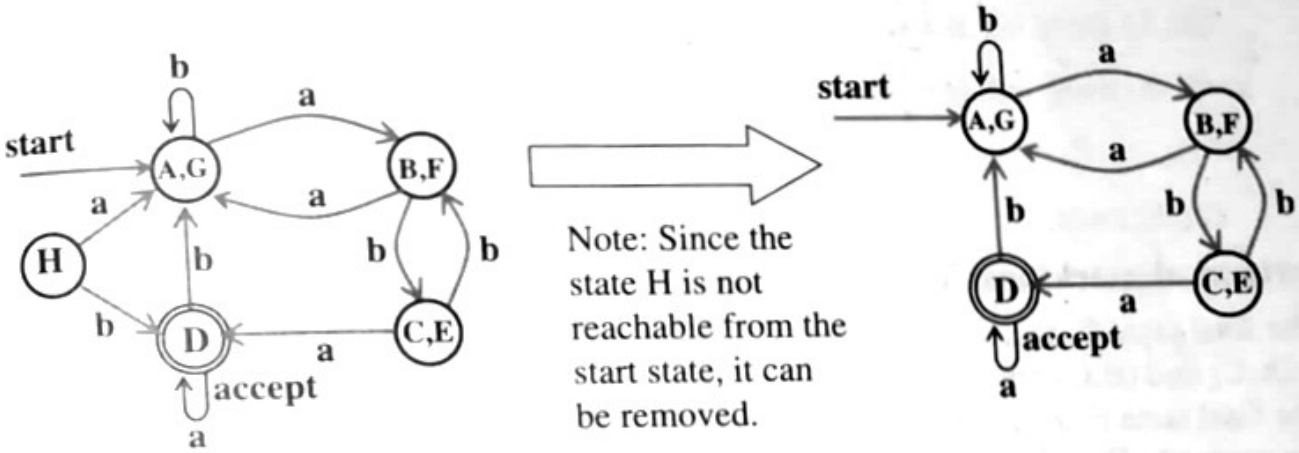
(A,G), (B,F), (C,E), D, H

Step 3: Compute the transition table: Use the transition table of given DFA and obtain the transitions for the minimized DFA as shown below:

δ	a	b
→ (A,G)	(B,F)	(A,G)
(B,F)	(A,G)	(C,E)
(C,E)	D	(B,F)
*D	D	(A,G)
H	(A,G)	D

Step 4: Identify the start state: Since the group (A,G) has the start state of DFA, the group (A, G) is the start state of minimized DFA.

Step 5: Identify the final state: The group D is the final state of given DFA. So, D will be the final state of minimized DFA. The transition diagram of the minimized DFA is shown below:



Example 3: Minimize the following DFA using table-filling algorithm where A is the start state. The states C F and I are final states.

δ	0	1
→ A	B	E
B	C	F
*C	D	H
D	E	H
E	F	I
*F	G	B
G	H	B
H	I	C
*I	A	E

Solution: The DFA can be minimized using table filling algorithm as shown below:

The various states of given DFA are: A, B, C, D, E, F, G, H, I.