CLOSURE PROPERTIES OF REGULAR LANGUAGES

1. Closure under Union

If L and M are regular languages, so is L UM.

Proof : Let L and M be the languages of regular expressions R and S, respectively.

Then R+S is a regular expression whose language is L U M

2. Closure under Concatenation and Kleene Closure

The same idea can be applied usingKleeneclosure :

RS is a regular expression whose language is LM.

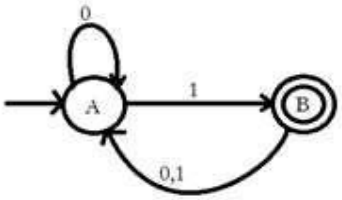
R* is a regular expression whose language is L*.

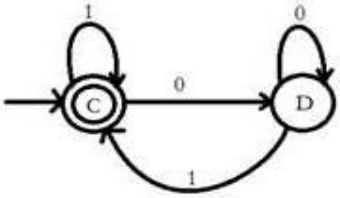
3. Closure under intersection

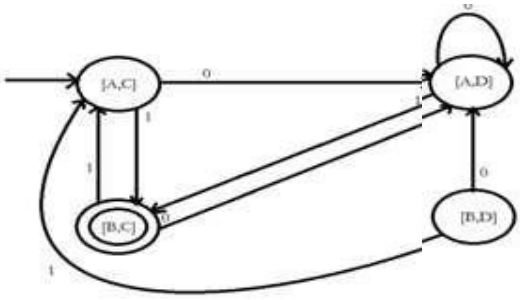
If L and M are regular languages, so is L ∩ M

Proof : Let A and B be two DFA's whose regular languages are L and M respectively.

Now, construct C, the product automation of A and B. Make the final states of C be the pairs consisting of final states of both A and B.



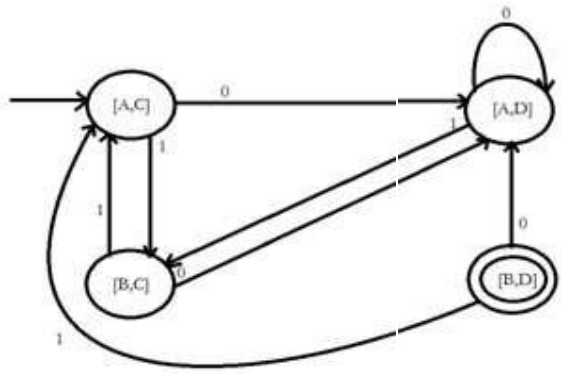




4. Closure under Difference

If L and M area regular languages, so is L-M, which means all the strings that are in L , but not in M.

Proof : Let A and B be two DFA's whose regular languages are L and M respectively. Now, construct C, the product automation of A and B. Make the final states of C be the pairs consisting of final states of A, but not of B. The DFA's A-B and C-D remain unchanged, but the final DFA varies as follows:



5. Closure under Concatenation

The complementof a language L (with respect to an alphabet Σ such that Σ* contains L) isΣ* – L Since Σ* is surely regular, the complement of a regular language is always regular

6. Closure under Reversal

Given language L, LR is the set of strings whose reversal is in L

L = {0, 01, 100}; LR = {0, 10, 001}

Basis: If E is a symbol a, ε, or ∅ , the ELR = E.

Induction: If E is

- F+G, then ER = FR + GR.
- FG, then ER = GRFR
- F, *then ER = (FR)*

Let E = 01* + 10*.

$ER=(01* +10*)R=(01*)R+(10*)RE=(01* +10*)R=(01*)R+(10* R)R$

$=(1*)R0R+(0*)R1R=(1*)R0R+(0*)$

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

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

Definition of homomorphism:

A homomorphism on an alphabet isa function that gives a string for each symbol in that alphabet.

Closure property:

If L is a regular language, and h is a homomorphism on its alphabet, then h(L) = {h(w) | w is in L} is also a regular language.

Proof: Let E be a regular expressionffor L.

Apply h to each symbol in E.

Language of resulting RE is h(L)

Example:

Let h(0) = ab; h(1) = ε.

Let L be the language of regular expression 01* + 10*.

Then h(L) is the language of regular expression abε* + ε(ab)*

abε* + ε(ab)* can be simplified.

ε* = ε, so abε* = abε.

ε is the identity under concatenation.

That is, εE = Eε= E for any RE E.

Thus, abε* + ε(ab)* = abε + ε(ab)*

= ab + (ab)*.

Finally, L(ab) is contained in L((ab))*, so a RE for h(L) is (ab)*

8. Closure under inverse homomorphism

a. Start with a DFA A for L.

b. Construct a DFA B for h-1(L) with: - The same set of states.

- The same start state.
- The same final states.
- Input alphabet = the symbols to which homomorphism h applies

CONVERSION FROM NFA WITH EPSILON TO DFA

Non-deterministic finite automata(NFA) is a finite automata where for some cases when a specific input is given to the current state, the machine goes to multiple states or more than 1 states. It can contain ε move. It can be represented as M = { Q, ∑, δ, q0, F}.

Where

1. Q: finite set of states
2. ∑: finite set of the input symbol
3. q0: initial state
4. F: **final** state
5. δ: Transition function

NFA with ϵ move: If any FA contains ε transaction or move, the finite automata is called NFA with ϵ

ε-closure: ε-closure for a given state A means a set of states which can be reached from the state A with only ε(null) move including the state A itself.

Steps for converting NFA with ε to

Step 1: We will take the ε-closure for the starting state of NFA as a starting state of DFA.

Step 2: Find the states for each input symbol that can be traversed from the present. That means the union of transition value and their closures for each state of NFA present in the current state of DFA.

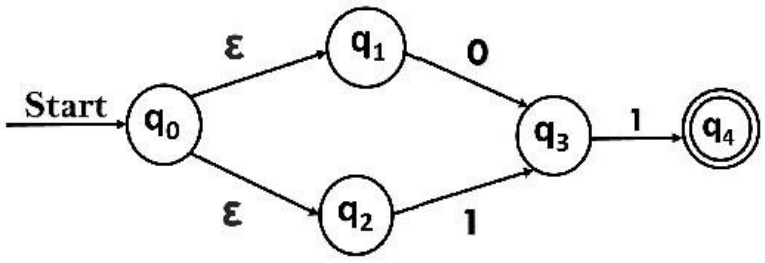
Step 3: If we found a new state, take it as current state and repeat step 2.

Step 4: Repeat Step 2 and Step 3 until there is no new state present in the transition table of

Step 5: Mark the states of DFA as a final state which contains the final state of NFA.

Example 1:

Convert the NFA with ε into its equivalent DFA.



Solution:

Let us obtain ε-closure of each state.

1. ε-closure {q0} = {q0, q1, q2}
2. ε-closure {q1} = {q1}
3. ε-closure {q2} = {q2}
4. ε-closure {q3} = {q3}
5. ε-closure {q4} = {q4}

Now, let ε-closure {q0} = {q0, q1, q2} be state A.

Hence

δ'(A, 0) = ε-closure {δ((q0, q1, q2), 0) }

= ε-closure {δ(q0, 0) ∪ δ(q1, 0) ∪ δ(q2, 0) }

= ε-closure {q3}

= {q3} **call it as state B**.

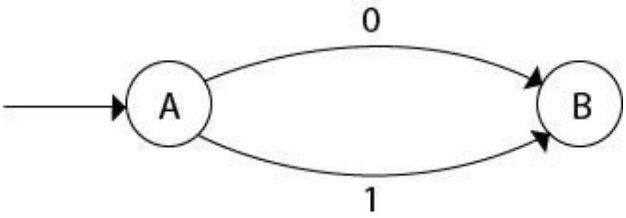
δ'(A, 1) = ε-closure {δ((q0, q1, q2), 1) }

= ε-closure {δ((q0, 1) ∪ δ(q1, 1) ∪ δ(q2, 1) }

= ε-closure {q3}

= {q3} = B.

The partial DFA will be



Now,

$\delta'(B, 0) = \varepsilon\text{-closure } \{\delta(q3, 0)\}$

$= \phi$

$\delta'(B, 1) = \varepsilon\text{-closure } \{\delta(q3, 1)\}$

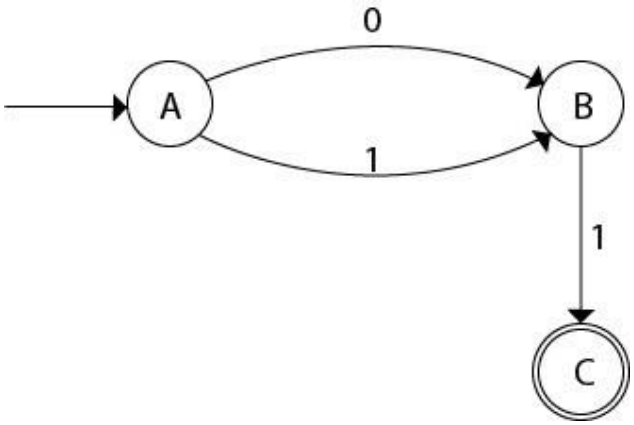
$= \varepsilon\text{-closure } \{q4\}$

$= \{q4\}$ **i.e. state C**

For state C:

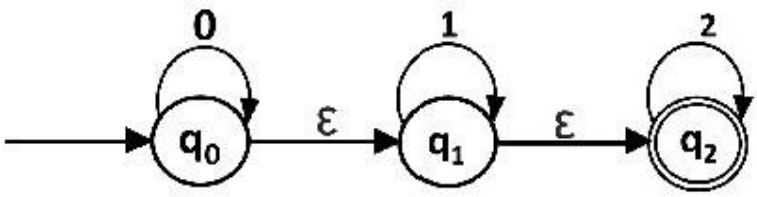
1. $\delta'(C, 0) = \varepsilon\text{-closure } \{\delta(q4, 0)\}$
2. $= \phi$
3. $\delta'(C, 1) = \varepsilon\text{-closure } \{\delta(q4, 1)\}$
4. $= \phi$

The DFA will be,



Example 2:

Convert the given NFA into its equivalent DFA.



Solution: Let us obtain the ε-closure of each state.

1. ε-closure(q0) = {q0, q1, q2}
2. ε-closure(q1) = {q1, q2}
3. ε-closure(q2) = {q2}

Now we will obtain δ' transition. Let ε-closure(q0) = {q0, q1, q2} call it as **state A**.

δ'(A, 0) = ε-closure{δ((q0, q1, q2), 0)}

= ε-closure{δ(q0, 0) ∪ δ(q1, 0) ∪ δ(q2, 0)}

= ε-closure{q0}

= {q0, q1, q2}

δ'(A, 1) = ε-closure{δ((q0, q1, q2), 1)}

= ε-closure{δ(q0, 1) ∪ δ(q1, 1) ∪ δ(q2, 1)}

= ε-closure{q1}

= {q1, q2} **call it as state B**

δ'(A, 2) = ε-closure{δ((q0, q1, q2), 2)}

= ε-closure{δ(q0, 2) ∪ δ(q1, 2) ∪ δ(q2, 2)}

= ε-closure{q2}

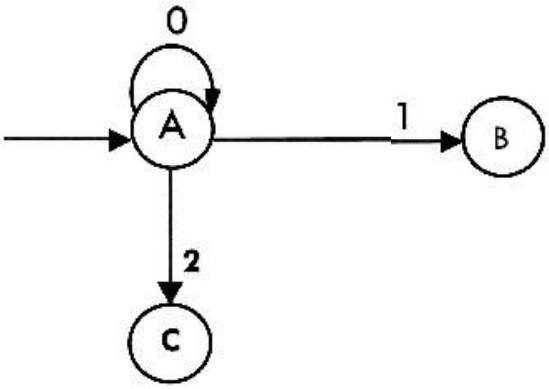
= {q2} **call it state C**

Thus we have

obtained 1. δ'(A, 0) = A

2. δ'(A, 1) = B
3. δ'(A, 2) = C

The partial DFA will be:



Now we will find the transitions on states B and C for each input.

Hence

$\delta'(B, 0) = \varepsilon\text{-closure}\{\delta((q1, q2), 0)\}$

$= \varepsilon\text{-closure}\{\delta(q1, 0) \cup \delta(q2, 0)\}$

$= \varepsilon\text{-closure}\{\phi\}$

$= \phi$

$\delta'(B, 1) = \varepsilon\text{-closure}\{\delta((q1, q2), 1)\}$

$= \varepsilon\text{-closure}\{\delta(q1, 1) \cup \delta(q2, 1)\}$

$= \varepsilon\text{-closure}\{q1\}$

$= \{q1, q2\}$ **i.e. state B itself**

$\delta'(B, 2) = \varepsilon\text{-closure}\{\delta((q1, q2), 2)\}$

$= \varepsilon\text{-closure}\{\delta(q1, 2) \cup \delta(q2, 2)\}$

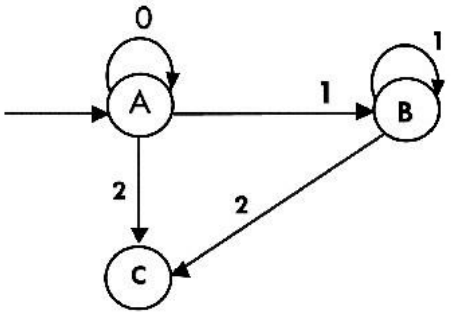
$= \varepsilon\text{-closure}\{q2\}$

$= \{q2\}$ **i.e. state C itself**

Thus we have obtained

1. $\delta'(B, 0) = \phi$
2. $\delta'(B, 1) = B$
3. $\delta'(B, 2) = C$

The partial transition diagram will be



Now we will obtain transitions for C:

$\delta'(C, 0) = \varepsilon\text{-closure}\{\delta(q2, 0)\}$

$= \varepsilon\text{-closure}\{\phi\}$

$= \phi$

$\delta'(C, 1) = \varepsilon\text{-closure}\{\delta(q2, 1)\}$

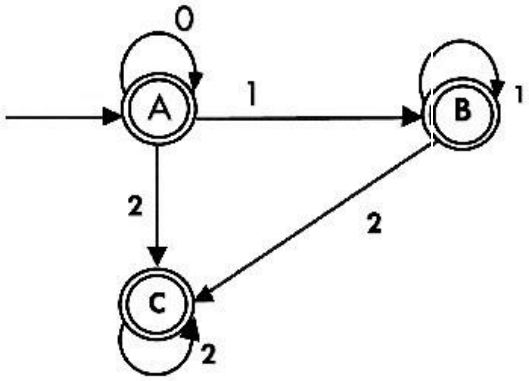
$= \varepsilon\text{-closure}\{\phi\}$

$= \phi$

$\delta'(C, 2) = \varepsilon\text{-closure}\{\delta(q2, 2)\}$

$= \{q2\}$

Hence the DFA is



As A = {q0, q1, q2} in which final state q2 lies hence A is final state. B = {q1, q2} in which the state q2 lies hence B is also final state. C = {q2}, the state q2 lies hence C is also a final state.

CONVERSION FROM NFA TO DFA

In NFA, when a specific input is given to the current state, the machine goes to multiple states. It can have zero, one or more than one move on a given input symbol. On the other hand, in DFA, when a specific input is given to the current state, the machine goes to only one state. DFA has only one move on a given input symbol.

Let, M = (Q, ∑, δ, q0, F) is an NFA which accepts the language L(M). There should be equivalent DFA denoted by M' = (Q', ∑', q0', δ', F') such that L(M) = L(M').

Steps for converting NFA to DFA:

Step 1: Initially Q' = ϕ

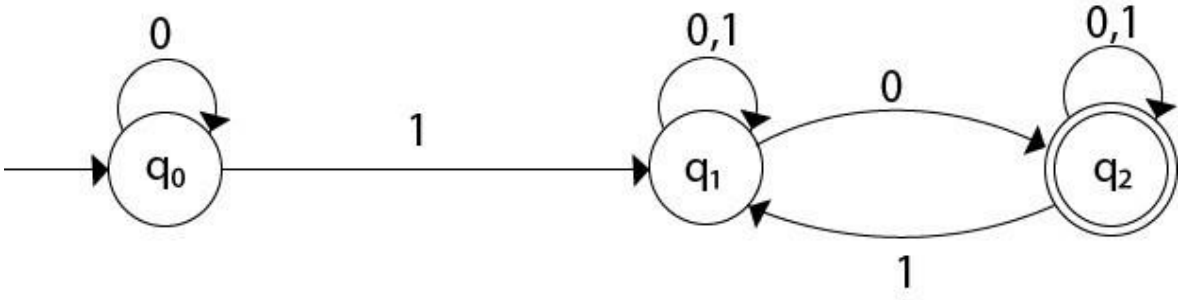
Step 2: Add q0 of NFA to Q'. Then find the transitions from this start state.

Step 3: In Q', find the possible set of states for each input symbol. If this set of states is not in Q', then add it to Q'.

Step 4: In DFA, the final state will be all the states which contain F(final states of NFA)

Example 1:

Convert the given NFA to DFA.



Solution: For the given transition diagram we will first construct the transition table.

State	0	1

→q0		q0	q1
q1		{q1, q2}	q1
*q2		q2	{q1, q2}

Now we will obtain δ' transition for state q0.

1. δ'([q0], 0) = [q0]
2. δ'([q0], 1) = [q1]

The δ' transition for state q1 is obtained as:

1. δ'([q1], 0) = [q1, q2] (**new** state generated)
2. δ'([q1], 1) = [q1]

The δ' transition for state q2 is obtained as:

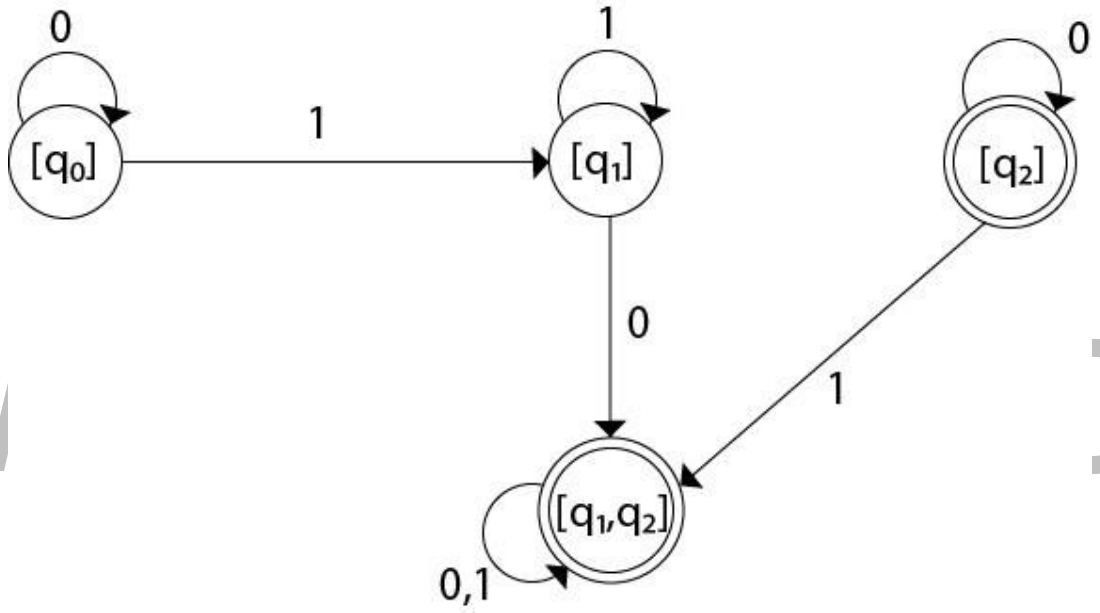
1. δ'([q2], 0) = [q2]
2. δ'([q2], 1) = [q1, q2]

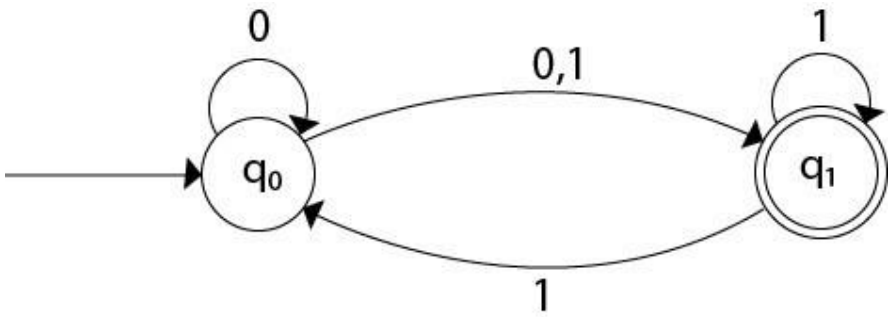
Now we will obtain δ' transition on [q1, q2].

1. δ'([q1, q2], 0) = δ(q1, 0) ∪ δ(q2, 0)
2. = {q1, q2} ∪ {q2}
3. = [q1, q2]
4. δ'([q1, q2], 1) = δ(q1, 1) ∪ δ(q2, 1)
5. = {q1} ∪ {q1, q2}
6. = {q1, q2}
7. = [q1, q2]

The state [q1, q2] is the final state as well because it contains a final state q2. The transition table for the constructed DFA will be:

State	**0**	**1**





Solution: For the given transition diagram we will first construct the transition table.

State	0	1
→q0	{q0, q1}	{q1}
*q1	Φ	{q0, q1}

Now we will obtain δ' transition for state q0.

1. δ'([q0], 0) = {q0, q1}
2. = [q0, q1] (**new** stategenerated)
3. δ'([q0], 1) = {q1} =[q1]

The δ' transition for state q1 is obtained as:

1. δ'([q1], 0) = ϕ
2. δ'([q1], 1) = [q0, q1]

Now we will obtain δ' transition on [q0, q1].

1. δ'([q0, q1], 0) = δ(q0, 0) ∪ δ(q1, 0)
2. = {q0, q1} ∪ ϕ
3. = {q0, q1}
4. = [q0, q1]

Similarly,

1. δ'([q0, q1], 1) = δ(q0, 1) ∪ δ(q1, 1)

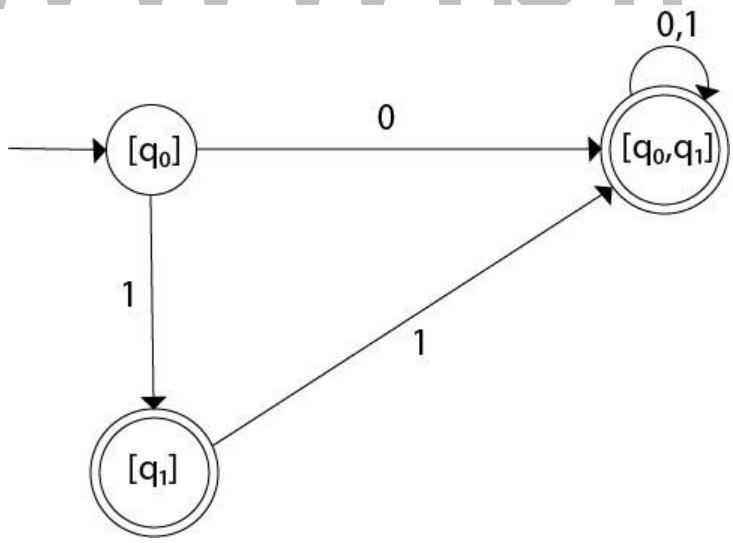
2. = {q1} ∪ {q0, q1}
3. = {q0, q1}
4. = [q0, q1]

As in the given NFA, q1 is a final state, then in DFA wherever, q1 exists that state becomes a final state. Hence in the DFA, final states are [q1] and [q0, q1]. Therefore set of final states F = {[q1], [q0, q1]}.

The transition table for the constructed DFA will be:

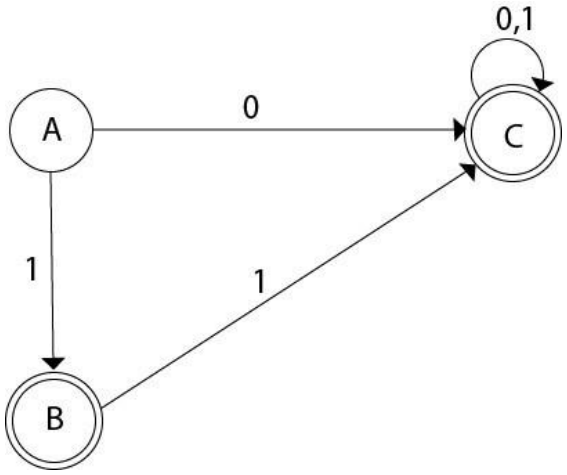
State	0	1
→[q0]	[q0, q1]	[q1]
*[q1]	ϕ	[q0, q1]
*[q0, q1]	[q0, q1]	[q0, q1]

The Transition diagram will be:



Even we can change the name of the states of DFA.

Suppose



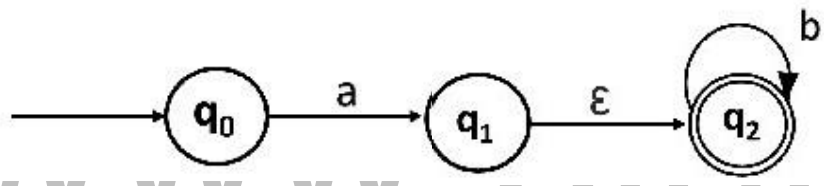
ELIMINATING EPSILON TRANSITIONS

NFA with ε can be converted to NFA without ε, and this NFA without ε can be converted to DFA. To do this, we will use a method, which can remove all the ε transition from given NFA. The method will be:

1. Find out all the ε transition ε from each state from Q. That will be called as ε-closure{q1} where qi ∈ Q.
2. Then δ' transitions can be obtained. The δ' transitions mean a ε-closure on δ moves.
3. Repeat Step-2 for each input symbol and each state of given NFA.
4. Using the resultant states, the transition table for equivalent NFA without ε can be built.

Example:

Convert the following NFA with ε to NFA without ε.



Solutions: We will first obtain ε-closures of q0, q1 and q2 as follows:

1. ε-closure(q0) = {q0}
2. ε-closure(q1) = {q1, q2}
3. ε-closure(q2) = {q2}

Now the δ' transition on each input symbol is obtained as:

1. δ'(q0, a) = ε-closure(δ(δ^(q0, ε),a))
2. = ε-closure(δ(ε-closure(q0),a))
3. = ε-closure(δ(q0, a))
4. = ε-closure(q1)
5. = {q1, q2}
6.
7. δ'(q0, b) = ε-closure(δ(δ^(q0, ε),b))
8. = ε-closure(δ(ε-closure(q0),b))

9. = ε-closure(δ(q0, b))
10. = Φ

Now the δ' transition on q1 is obtained as:

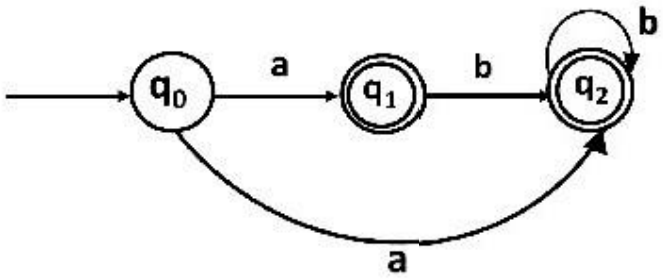
1. δ'(q1, a) = ε-closure(δ(δ^(q1,ε),a))
2. = ε-closure(δ(ε-closure(q1),a))
3. = ε-closure(δ(q1, q2), a)
4. = ε-closure(δ(q1, a) ∪ δ(q2, a))
5. = ε-closure(Φ ∪ Φ)
6. = Φ
7.
8. δ'(q1, b) = ε-closure(δ(δ^(q1, ε),b))
9. = ε-closure(δ(ε-closure(q1),b))
10. = ε-closure(δ(q1, q2), b)
11. = ε-closure(δ(q1, b) ∪ δ(q2, b))
12. = ε-closure(Φ ∪ q2)
13. = {q2}

The δ' transition on q2 is obtained as:

1. δ'(q2, a) = ε-closure(δ(δ^(q2, ε),a))
2. = ε-closure(δ(ε-closure(q2),a))
3. = ε-closure(δ(q2, a))
4. = ε-closure(Φ)
5. = Φ
6.
7. δ'(q2, b) = ε-closure(δ(δ^(q2, ε),b))
8. = ε-closure(δ(ε-closure(q2),b))
9. = ε-closure(δ(q2, b))
10. = ε-closure(q2)
11. = {q2}

Now we will summarize all the computed δ' transitions:

1. δ'(q0, a) = {q0, q1}
2. δ'(q0, b) = Φ



CONVERSION OF REGULAR EXPRESSION TO FINITE A TOMATA

To convert the RE to FA, we are going to use a method called the subset method. This method is used to obtain FA from the given regular expression. This method is given below:

Step 1: Design a transition diagram for given regular expression, using NFA with ε moves.

Step 2: Convert this NFA with ε toN FA without ε.

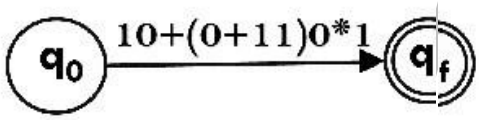
Step 3: Convert the obtained NFA to equivalent DFA.

Example 1:

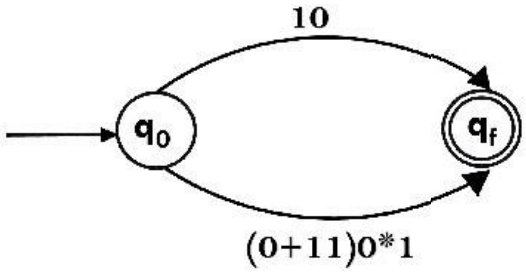
Design a FA from given regular expression 10 + (0 + 11)0* 1.

Solution: First we will construct the transition diagram for a given regular expression.

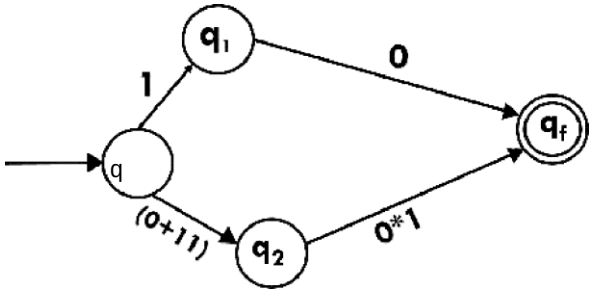
Step 1:



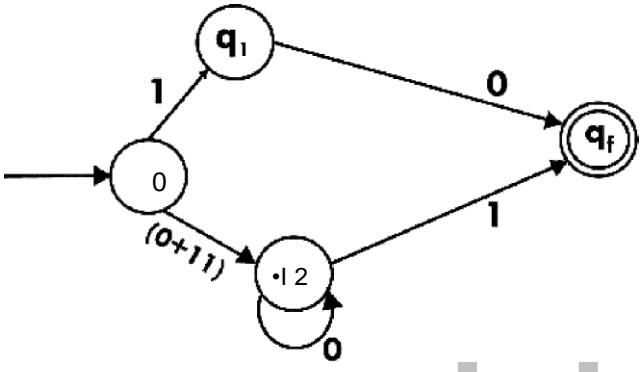
Step 2:



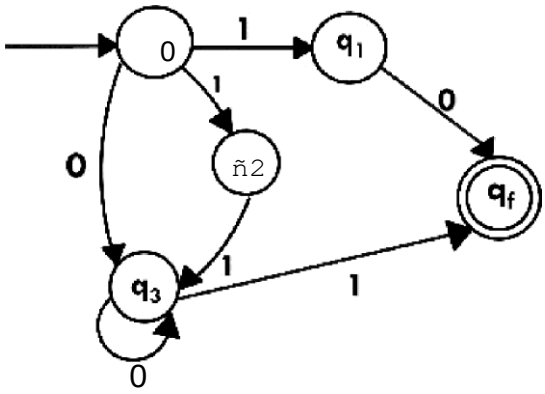
Step 3:



Step 4:



Step 5:



Now we have got NFA without ε. Now we will convert it into required DFA for that, we will first write a transition table for this NFA.

State	0	1

→q0	q3	{q1, q2}
q1	Qf	Φ
q2	Φ	q3
q3	q3	Qf
*qf	Φ	Φ

The equivalent DFA will be:

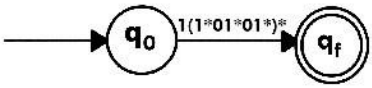
State	0	1
→[q0]	[q3]	[q1, q2]
[q1]	[qf]	Φ
[q2]	Φ	[q3]
[q3]	[q3]	[qf]
[q1, q2]	[qf]	[qf]
*[qf]	Φ	Φ

Example 2:

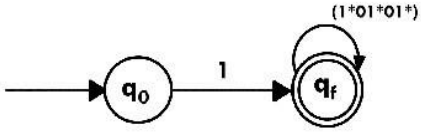
Design a NFA from given regular expression 1 (1* 01* 01*)*.

Solution: The NFA for the given regular expression is as follows:

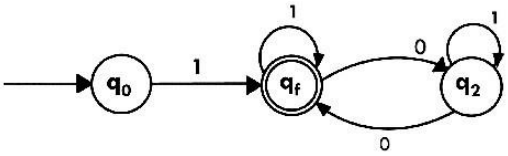
Step 1:



Step 2:



Step 3:



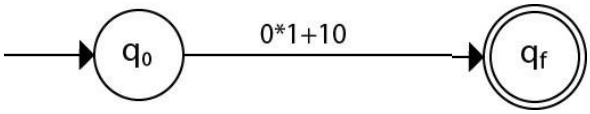
Example 3:

Construct the FA for regular expression 0*1 + 10.

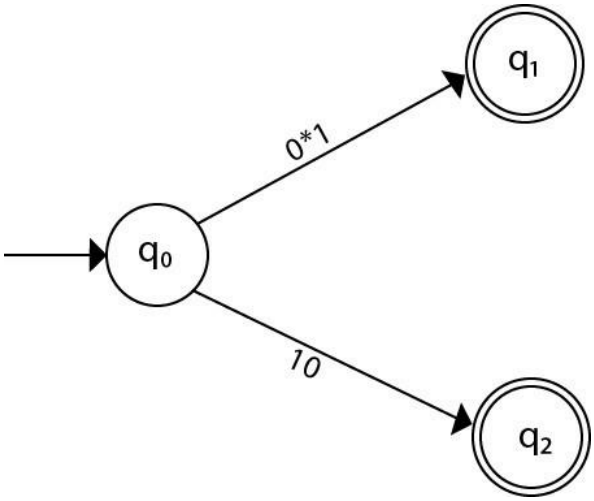
Solution:

We will first construct FA for R = 0*1 + 10 as follows:

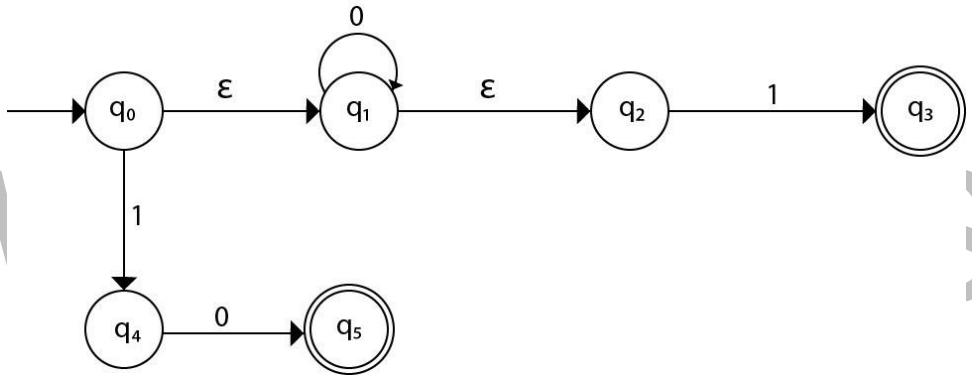
Step 1:



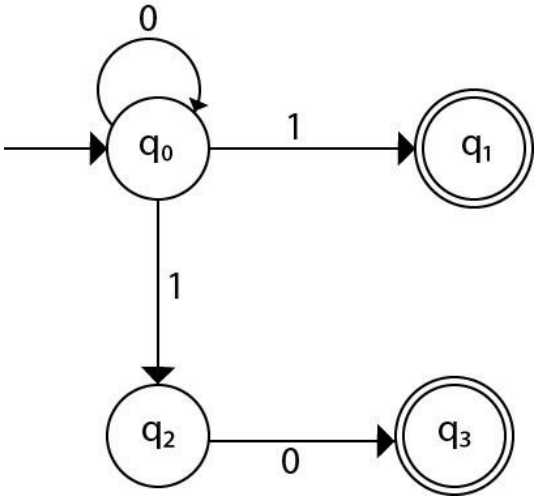
Step 2:



Step 3:



Step 4:



PROVING LANGUAGES NOT TO BE REGULAR

Theorem

Let L be a regular language. Then there exists a constant **'c'** such that for every string **w** in **L** −

|w| ≥ c

We can break **w** into three strings, **w = xyz**, such that −

- |y| > 0
- |xy| ≤ c
- For all k ≥ 0, the string xy^kz is also in

L. Applications of Pumping Lemma

Pumping Lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

- If **L** is regular, it satisfies Pumping Lemma.
- If **L** does not satisfy Pumping Lemma, it is non-regular.

Method to prove that a language L is not regular

- At first, we have to assume that **L** is regular.
- So, the pumping lemma should hold for **L**.
- Use the pumping lemma to obtain a contradiction −
  - Select **w** such that **|w| ≥ c**
  - Select **y** such that **|y| ≥ 1**
  - Select **x** such that **|xy| ≤ c**
  - Assign the remaining string to **z.**
  - Select **k** such that the resulting string is not in

L. Hence L is not regular.

Problem

Prove that $\mathbf{L = \{a^i b^i \mid i \geq 0\}}$ is not regular.

Solution

- At first, we assume that **L** is regular and n is the number of states.
- Let w = $a^n b^n$. Thus |w| = 2n ≥ n.
- By pumping lemma, let w = xyz, where |xy| ≤ n.
- Let x = a^p, y = a^q, and z = $a^r b^n$, where p + q + r = n, p ≠ 0, q ≠ 0, r ≠ 0. Thus |y| ≠ 0.
- Let k = 2. Then $xy^2z = a^p a^{2q} a^r b^n$.
- Number of as = (p + 2q + r) = (p + q + r) + q = n + q
- Hence, $xy^2z = a^{n+q} b^n$. Since q ≠ 0, xy^2z is not of the form $a^n b^n$.
- Thus, xy^2z is not in L. Hence L is not regular.

REGULAR EXPRESSION

- The language accepted by finite automata can be easily described by simple expressions called Regular Expressions. It is the most effective way to represent any language.
- The languages accepted by some regular expression are referred to as Regular languages.
- A regular expression can also be described as a sequence of pattern that defines a string.
- Regular expressions are used to match character combinations in strings. String searching algorithm used this pattern to find the operations on a string.

For instance:

In a regular expression, x* means zero or more occurrence of x. It can generate {e, x, xx, xxx, xxxx, ...}

In a regular expression, x^+ means one or more occurrence of x. It can generate {x, xx, xxx, xxxx, ...}

Operations on Regular Language

The various operations on regular language are:

Union: If L and M are two regular languages then their union L U M is also a union.

1. 1. L U M = {s | s is in L or s is in M}

Intersection: If L and M are two regular languages then their intersection is also an intersection.

1. 1. L ⋂ M = {st | s is in L and t is in M}

Kleen closure: If L is a regular language then its Kleen closure L1* will also be a regular language.

1. 1. L* = Zero or more occurrence of language L.

Example 1:

Write the regular expression for the language accepting all combinations of a's, over the set ∑ = {a}

Solution:

All combinations of a's means a may be zero, single, double and so on. If a is appearing zero times, that means a null string. That is we expect the set of {ε, a, aa, aaa, ...}. So we give a regular expression for this as:

1. R = a*

That is Kleen closure of a.

Example 2:

Write the regular expression for the language accepting all combinations of a's except the null string, over the set ∑ = {a}

Solution:

The regular expression has to be built for the language

1. L = {a, aa, aaa, ...}

This set indicates that there is no null string. So we can denote regular expression as:

$R = a^+$

Example 3:

Write the regular expression for the language accepting all the string containing any number of a's and b's.

Solution:

The regular expression will be:

1. r.e. = (a + b)*

This will give the set as L = {ε, a, aa, b, bb, ab, ba, aba, bab,..... }, any combination of a and b.

The (a + b)* shows any combination with a and b even a null string.

Examples of Regular Expression

Example 1:

Write the regular expression for the language accepting all the string which are starting with 1 and ending with 0, over ∑ = {0, 1}.

Solution:

In a regular expression, the first symbol should be 1, and the last symbol should be 0. The r.e. is as follows:

1. R = 1 (0+1)* 0

Example 2:

Write the regular expression for the language starting and ending with a and having any having any combination of b's in between.

Solution:

The regular expression will be:

1. R = a b* b

Example 3:

Write the regular expression for the language starting with a but not having consecutive b's.

Solution: The regular expression has to be built for the language:

1. L = {a, aba, aab, aba, aaa, abab, ... }

The regular expression for the above language is:

1. R = {a + ab}*

Example 4:

Write the regular expression for the language accepting all the string in which any number of a's is followed by any number of b's is followed by any number of c's.

Solution: As we know, any number of a's means a* any number of b's means b*, any number of c's means c*. Since as given in problem statement, b's appear after a's and c's appear after b's. So the regular expression could be:

1. R = a* b* c*

Example 5:

Write the regular expression for the language over ∑ = {0} having even length of the string.

Solution:

The regular expression has to be built for the language:

1. L = {ε, 00, 0000, 000000, ... }

The regular expression for the above language is:

1. R = (00)*

Example 6:

Write the regular expression for the language having a string which should have atleast one 0 and alteast one 1.

Solution:

The regular expression will be:

1. R = [(0 + 1)* 0 (0 + 1)* 1 (0 + 1)*] + [(0 + 1)* 1 (0 + 1)* 0 (0 + 1)*]

Example 7:

Describe the language denoted by following regular expression

1. r.e. = (b* (aaa)* b*)*

Solution:

The language can be predicted from the regular expression by finding the meaning of it. We will first split the regular expression as:

r.e. = (any combination of b's) (aaa)* (any combination of b's)

L = {The language consists of the string in which a's appear triples, there is no restriction on the number of b's}

Example 8:

Write the regular expression for the language L over ∑ = {0, 1} such that all the string do not contain the substring 01.

Solution:

The Language is as follows:

1. L = {ε, 0, 1, 00, 11, 10, 100,..... }

The regular expression for the above language is as follows:

1. R = (1* 0*)

Example 9:

Write the regular expression for the language containing the string over {0, 1} in which there are at least two occurrences of 1's between any two occurrences of 1's between any two occurrences of 0's.

Solution: At least two 1's between two occurrences of 0's can be denoted by (0111*0)*.

Similarly, if there is no occurrence of 0's, then any number of 1's are also allowed. Hence the r.e. for required language is:

1. R = (1 + (0111*0))*

Example 10:

Write the regular expression for the language containing the string in which every 0 is immediately followed by 11.

Solution:

The regular expectation will be:

1. R = (011 + 1)*