MODULE-2

Regular Expressions (RE): what is a RE? Kleene's theorem, Applications of REs, Manipulating and Simplifying REs. Regular Grammars: Definition, Regular Grammars and Regular languages. Regular Languages (RL) and Non-regular Languages: How many RLs, To show that a language is regular, Closure properties of RLs, to show some languages are not RLs.

Module 2

CONTENTS

Chapter No: 2	Title	Page No.
2.1	**REGULAR EXPRESSION**	1 - 46
2.1.1	What is Regular Expression	1
2.1.2	Kleen's Theorem	9
2.1.2.1	Building FSM from Regular Expression	10
2.1.2.2	Building Regular Expression from FSM – State Elimination Technique	20
2.1.2.3	Building Regular Expression from FSM – Kleen's Theorem	38
2.1.3	Application of Regular Expression	40
2.1.4	Manipulating and Simplifying Regular Expressions.	40
2.2	**REGULAR GRAMMARS**	47- 66
2.2.1	Definition of Regular Grammars and Regular Languages	47
2.2.2	Regular and Non-Regular Languages, How many Regular languages?	54
2.2.3	To show that a language is regular,	54
2.2.4	Closure properties of RLs	55
2.2.5	To show some languages are not RLs.(**P**umping Theorem for RLs)	59

Regular Expressions

Introduction

Instead of focusing on the power of a computing device, let's look at the task that we need to perform. Let's consider problems in which our goal is to match finite or repeating patterns.

For example regular expressions are used as pattern description language in

- Lexical analysis.-- compiler
- Filtering email for spam.
- Sorting email into appropriate mailboxes based on sender and/or content words and phrases.
- Searching a complex directory structure by specifying patterns that are known to occur in the file we want.

A regular expression is ***a pattern description*** language, which is used to describe particular patterns of interest. A regular expression provides a concise and flexible means for "**matching**" strings of text, such as particular characters, words, or patterns of characters.

Example: **[ ]** : A character class which matches any character within the brackets

[^ \t\n] matches any character except space, tab and newline character.

Regular expression:

A language accepted by a finite- state machine is called as ***regular language***. A regular language can be described using ***regular expressions***, in the form of algebraic notations consisting of the symbols such as alphabets or symbols in **Σ** and a set of special symbols to which we will attach particular meanings when they occur in a regular expression. These symbols are Ø, U, ε, (,), *, and .

Define a Regular expression.

A ***regular expression*** is a string that can be formed according to the following rules:

1. **Ø** is a regular expression.
2. **ε** *is a* regular expression.
3. Every element in ∑ is a regular expression.
4. Given two regular expressions ***α*** *and* ***β****, αβ is a regular expression.*

5. Given two regular expressions ***α*** *and* ***β****, α U β is a regular expression.*
6. Given a regular expression ***α****, α* is a regular expression.*
7. Given a regular expression ***α****,* α^+ *is a regular expression.*
8. Given a regular expression ***α****, **(α)** is a regular expression.*

Example:

Let ∑ = {a, b}, The following strings are regular expressions:

Ø , ε, a, b, $(aUb)^*$ **etc…….**

NOTE:

1. L (**Ø**) = Ø, the language that contains no strings.
2. L(**ε**) = {ε}, the language that contains just the empty string.
3. For any **c** ϵ ∑, L (c) = {c}, the language that contains the single one character string c
4. For any regular expressions **α** and **β**. L (αβ) = L(**α**)L(**β**). That is concatenation of two regular expressions.
5. If either L(α) or L(β) is equal to Ø, then the concatenation will also be equal to **Ø**
6. For any regular expressions **α** and **β**. L (αUβ) = L(α) **U** L(β). That is union of two regular expressions.
7. For any regular expressions α, L (α^*) = **(L (α))*** where * is the Kleen start operator.
8. L (**Ø***) = { ε }

What is L((**a**U **b**)* **b**) = ?

L((aU b)* b) = L((a U b)*) L(b)

= (L(a) U L(b))* L(b)

= ({a} U {b})* {b}

= ***{a, b}* {b}.*** That is the set of all strings over the alphabet {a, b} that ends in b.

$L(((a \cup b)(a \cup b))a(a \cup b)^*)$ $= L(((a \cup b)(a \cup b)))L(a)\ L((a \cup b)^*)$

$= L((a \cup b)(a \cup b))\ \{a\}\ (L((a \cup b)))^*$

$= L((a \cup b))L((a \cup b))\ \{a\}\ \{a,b\}^*$

$= \{a,b\}\ \{a,b\}\ \{a\}\ \{a,b\}^*$

So the meaning of the regular expression $((a \cup b)(a \cup b))a(a \cup b)^*$ is:

$\{xay : x \text{ and } y \text{ are strings of a's and b's and } |x| = 2\}$.

Regular expression	Meaning
a^*	String consisting of any number of a's. (zero or more a's)
a^+	String consisting of at least one a. (one or more a's)
(a , b)	String consisting of either a or b
$(a, b)^*$	String consisting of any nuber of a's and b's including ε
$(a, b)^*$ ab	Strings of a's and b's ending with ab.
ab$(a, b)^*$	Strings of a's and b's starting with ab.
$(a , b)^*$ ab $(a,b)^*$	Strings of a's and b's with substring **ab**.

Write the regular expressions for the following languages:

a. **Strings of a's and b's having length 2**:

Regular expression = (a + b) (a+ b). **OR** (a, b) (a, b) **OR** (a U b) (a U b)

b. **Strings of a's and b's of length ≤ 10**:

Regular expression = $(\varepsilon + a + b)^{10}$.

c. **Strings of a's and b's of even length.**

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

d. **Strings of a's and b's of odd length**

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

e. **Strings of a's of even length**

Regular expression = $(aa)^*$

f. **Strings of a's of odd length**

Regular expression = $a(aa)^*$

Strings of a's and b's with alternate a's and b's.

Alternate a's and b's can be obtained by concatenating the string (ab) zero or more times. ie $(ab)^*$ and adding an optional **b** to the front ie: (ε +b) and adding an optional **a** at the end, ie: (ε +a)

The regular expression = $\mathbf{(\varepsilon + b)\ (ab)^*(\varepsilon + a)}$

Obtain regular expression to accept the language containing at least one a and one b over Σ = { a, b, c}. OR
Obtain regular expression to accept the language containing at least one 0 and one 1 over Σ = { 0, 1, 2}.

String should contain at least one a and one b, so the regular expression corresponding to this is given by = ab + ba

There is no restriction on c's. Insert any number of a's, b's and c;s ie: $(a+b+c)^*$ in between the above regular expression.

So the *regular expression* = $\mathbf{(a+b+c)^* a (a+b+c)^* b(a+b+c)^* + (a+b+c)^* b(a+b+c)^* a(a+b+c)^*}$

Obtain regular expression to accept the language containing at least 3 consecutive zeros.

Regular expression for string containing 3 consecutive 0's = 000

The above regular expression can be preceded or followed by any number of 0's and 1's, ie: $(0+1)^*$

Regular expression = $\mathbf{(0+1)^*000(0+1)^*}$

Obtain regular expression to accept the language containing strings of a's and b's ending with b and has no substring **aa.**

Regular expression for strings of a's and b's ending with b and has no substring aa is nothing but the string containing any combinations of either b or ab without ε.

Regular expression = $\mathbf{(b + ab)^* (b +ab)^*}$

Obtain regular expression to accept the language containing strings of a's and b's such that L = { $a^{2n} b^{2m}$ | n, m ≥ 0 }

a^{2n} means even number of a's, regular expression = $(aa)^*$

b^{2m} means even number of b's, regular expression = $(bb)^*$.

The *regular expression* for the given language = $\mathbf{(aa)^* (bb)^*}$

Obtain regular expression to accept the language containing strings of a's and b's such that L = { $a^{2n+1} b^{2m}$ | n, m ≥ 0 }.

a^{2n+1} means odd number of a's, regular expression = $a(aa)^*$

b^{2m} means even number of b's, regular expression = $(bb)^*$

The *regular expression* for the given language = $\mathbf{a(aa)^* (bb)^*}$

Obtain regular expression to accept the language containing strings of a's and b's such that L = { $a^{2n+1}\ b^{2m+1}$ | n, m ≥ 0 }.

a^{2n+1} means odd number of a's, regular expression = $a(aa)^*$

b^{2m+1} means odd number of b's, regular expression = $b(bb)^*$

The *regular expression* for the given language = **$a(aa)^*b(bb)^*$**

Obtain regular expression to accept the language containing strings of 0's and 1's with exactly one 1 and an even number of 0's.

Regular expression for exactly one 1 = 1

Even number of 0's = $(00)^*$

So here 1 can be preceded or followed by even number of 0's or 1 can be preceded and followed by odd number of 0's.

The regular expression for the given language = **$(00)^*\ 1\ (00)^* + 0(00)^*\ 1\ 0(00)^*$**

Obtain regular expression to accept the language containing strings of 0's and 1's having no two consecutive 0's. OR

Obtain regular expression to accept the language containing strings of 0's and 1's with no pair of consecutive 0's.

Whenever a 0 occurs it should be followed by 1. But there is no restriction on number of 1's. So it is a string consisting of any combinations of 1's and 01's, ie regular expression = $(1+01)^*$

Suppose string ends with 0, the above regular expression can be modified by inserting $(0 + \varepsilon)$ at the end.

Regular expression for the given language = **$(1+01)^*\ (0 + \varepsilon)$**

Obtain regular expression to accept the language containing strings of 0's and 1's having no two consecutive 1's. OR

Obtain regular expression to accept the language containing strings of 0's and 1's with no pair of consecutive 1's.

Whenever a 1 occurs it should be followed by 0. But there is no restriction on number of 0's. So it is a string consisting of any combinations of 0's and 10's, ie regular expression = $(0+10)^*$

Suppose string ends with 1, the above regular expression can be modified by inserting $(1 + \varepsilon)$ at the end.

Regular expression for the given language = **$(0+10)^*\ (1 + \varepsilon)$**

Obtain regular expression to accept the following languages over Σ = { a, b}

i. Strings of a's and b's with substring aab.

Regular expression = $\mathbf{(a+b)^* aab(a+b)^*}$

ii. Strings of a's and b's such that 4th symbol from right end is **b** and the 5th symbol from right end is **a**.

Here the 4th symbol from right end is b and the 5th symbol from right end is a the corresponding regu;lar expression = ab(a+b)(a+b)(a+b).

But the above regular expression can be preceded with any number of a's and b's.

Therefore the regular expression for the given language = $\mathbf{(a+b)^*ab(a+b)(a+b)(a+b).}$

iii. Strings of a's and b's such that 10th symbol from right end is **b**.

The regular expression for the given language = $\mathbf{(a+b)^*b(a+b)^9.}$

iv. Strings of a's and b's whose lengths are multiple of 3.

OR

L = { |w| mod 3 = 0, where w is in Σ = { a, b}

Length of string w is multiple of 3, the regular expression = $\mathbf{((a+b)\ (a+b)\ (a+b))^*}$

v. Strings of a's and b's whose lengths are multiple of 5.

OR

L = { |w| mod 5 = 0, where w is in Σ = { a, b}

Length of string w is multiple of 5,

The regular expression = $\mathbf{((a+b)\ (a+b)\ (a+b)\ (a+b)(a+b))^*}$

vi. Strings of a's and b's not more than 3 a's:

Not more than 3 a's, regular expression= (ε+a) (ε+a) (ε+a).

But there is no restriction on b's, so we can include b^* in between the above regular expression.

The regular expression for the given language = $\mathbf{b^*(\varepsilon+a)\ b^*(\varepsilon+a)\ b^*\ (\varepsilon+a)\ b^*}$

vii. Obtain the regular expression to accept the words with two or more letters but beginning and ending with the same letter. Σ = { a, b}

Regular expression beginning and ending with same letter is = a a + b b. In between include any number of a's and b's.

Therefore the regular expression = **a** $(a+b)^*$ **a** + **b** $(a+b)^*$ **b**

viii. Strings of a's and b's of length is either even or multiple of 3.

Multiple of regular expression = $[(a+b)\ (a+b)\ (a+b)]^*$

Length is of even, regular expression = $[(a+b)\ (a+b)]^*$

So the regular expression for the given language = $((a+b)\ (a+b)\ (a+b)]^* + [(a+b)\ (a+b))^*$

ix. Obtain the regular expression to accept the language L = { a^nb^m | m+n is **even }**

Here n represents number of a's and m represents number of b's.

m+n is even results in two possible cases;

case i. when even number of a's followed by even number of b's.

regular expression : $(aa)^*(bb)^*$

case ii. Odd number of a's followed by odd number of b's.

regular expression = $a(aa)^*\ b(bb)^*$.

So the regular expression for the given language = **$(aa)^*(bb)^*$ + $a(aa)^*$ $b(bb)^*$**

x. Obtain the regular expression to accept the language L = { a^nb^m | n ≥ 4 and m ≤ **3 }.**

Here n ≥ 4 means at least 4 a's, the regular expression for this = $aaaa(a)^*$

m ≤ 3 means at most 3 b's, regular expression for this = (ε+b) (ε+b) (ε+b).

So the regular expression for the given language = **$aaaa(a)^*$ (ε+b) (ε+b) (ε+b).**

xi. Obtain the regular expression to accept the language L = { a^nb^m c^p | n ≥ 4 and m ≤ 3 p ≤ **2}.**

Here n ≥ 4 means at least 4 a's, the regular expression for this = $aaaa(a)^*$

m ≤ 3 means at most 3 b's, regular expression for this = (ε+b) (ε+b) (ε+b).

p ≤ 2 means at most 2 c's, regular expression for this = (ε+c) (ε+c)

So the regular expression for the given language = **aaaa(a)* (ε+b) (ε+b) (ε+b) (ε+c) (ε+c).**

xii. All strings of a's and b's that do not end with **ab.**

Strings of length 2 and that do not end with ab are ba, aa and bb.

So the regular expression = **(a+b)*(aa + ba +bb)**

xiii. All strings of a's, b's and c's with exactly one **a**.

The regular expression = **(b+c)* a (b+c)***

xiv. All strings of a's and b's with at least one occurrence of each symbol in Σ = {a, b}.

At least one occurrence of a's and b's means **ab** + **ba**, in between we have n number of a's and b's.

So the regular expression = **(a+b)* a (a+b)* b(a+b)* +(a+b)* b(a+b)* a(a+b)***

Obtain the regular expression for the language L = { $a^n b^m$ | m ≥ 1, n ≥ 1, nm ≥ 3 }

Solution:

Case i. Since **nm ≥ 3,** if **n** = 1 then **m** should be **≥ 3.** The equivalent regular expression is given by: RE = a bbb(b)*

Case ii. Since **nm ≥ 3,** if m = 1 then **n** should be **≥ 3.** The equivalent regular expression is given by: RE = aaa(a)* b

Case iii. Since **nm ≥ 3,** if m ≥ 2 and n ≥ 2 then the equivalent regular expression is given by:

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

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

Regular expression = **abbb(b)* + aaa(a)*b + aa(a)*bb(b)***

The regular expression language provides three operators (precedence order from highest to lowest)

1. Kleene star
2. Concatenation, and
3. Union

NOTE:

(α U ε) : optional α and expression can be satisfied either by matching α or the empty string.

(a U b)* : Describes the set of all strings composed of the characters a and b.

a* U b* ≠ (a U b)* : Every string in the language on the left contains only a's or b's whereas right side it contains combination of a's and b's.

(ab)* ≠ a*b*: The language on the left contains the string **abab**….. while the language on the right does not. The language on the right contains the string ***aaabbbb***, while the language on the left does not.

The regular expression a* is simply a string. It is different from Language L(a*) = {w: w is composed of zero or more a's}.

Kleene's Theorem

- The regular expression language is a useful way to define patterns..
- Any language that can be defined by a regular expression can be accepted by some finite state machine.
- Any language that can be accepted by a finite state machine can be defined by some regular expression.

*****Building an FSM from a Regular Expression

Theorem: Any language that can be defined with a regular expression can be accepted by some FSM and so is regular.

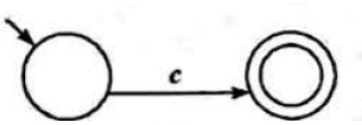
OR

Prove that if ***R*** is a regular expression then there exists some ***ε-NFA*** that accepts **L(R)**

Proof: The proof is given by constructing FSM:

For a given regular expression α, we can construct an FSM **M** such that L (α) = L (M).

If **α** is any **c** ϵ ∑, we construct for it the simple FSM as:



If α is **Ø**, we construct for it the simple FSM as:

If **α** is **ε** we construct for it the simple FSM as:

Let **β** and **γ** be regular expressions that define languages over the alphabet ∑

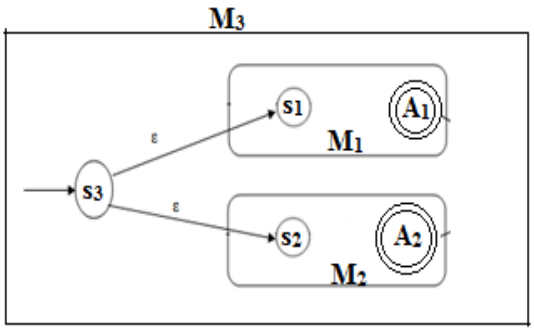
If L (β) is regular, then it is accepted by some FSM $M_1 = (K_1, \sum, \delta_1, s_1, A_1)$.

If L (γ) is regular, then it is accepted by some FSM $M_2 = (K_2, \sum, \delta_2, s_2, A_2)$.

If regular expression **α** = β U γ and if both **L(β)** and **L(γ)** are regular, then we construct $M_3 = (K_3, \sum, \delta_3, s_3, A_3)$, such that $L(M_3) = L(\alpha) = L(\beta) \cup L(\gamma)$.

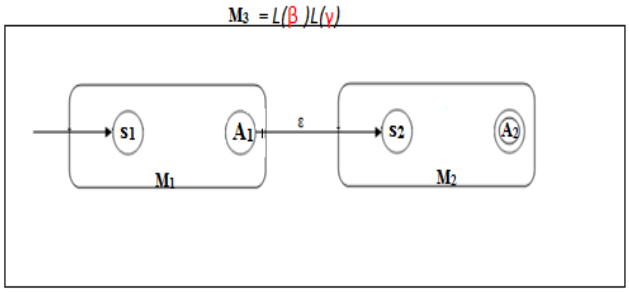
Construct a new machine M_3, by creating a new start state s_3, and connect it to the start states of M_1 and M_2 via ε-transitions. M_3 accepts if either M_1 or M_2 accepts.

So $M_3 = (\{ s_3\} \cup K_1 \cup K_2, \sum, \delta_3, s_3, A_1 \cup A_2)$ where $\delta_3 = \delta_1 \cup \delta_2\{((s_3, \varepsilon), s_1), (s_3, \varepsilon), s_2)\}$



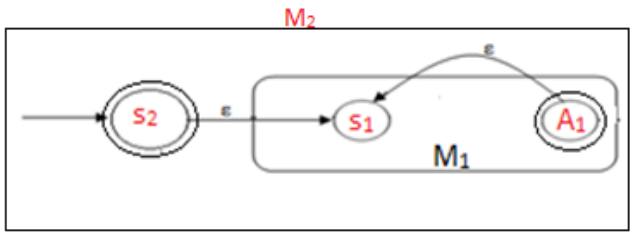
If regular expression $\alpha = \beta\gamma$ and if both $L(\beta)$ and $L(\gamma)$ are regular, then we construct $M_3 = (K_3, \sum, \delta_3, s_3, A_3)$, such that $L(M_3) = L(\alpha) = L(\beta)L(\gamma)$.

Construct a new machine M_3, by connecting every accepting state of M1 to the start state of M2 via an **ε**-transition. M_3 will start in the start state of M_1 and will accept if M_2 does. So $M_3 = (K_1 \cup K_2, \sum, \delta_3, s_1, A_2)$ where $\delta_3 = \delta_1 \cup \delta_2\{((q, \varepsilon), s_2) : q \in A_1)\}$



If regular expression $\alpha = \beta^*$ and $L(\beta)$ is regular, then we construct $M_2 = (K_2, \sum, \delta_2, s_2, A_2)$, such that $L(M_2) = L(\alpha) = L(\beta)^*$

M_2 is constructed by creating a new start state s_2 and make it **accepting state**, thus assuming that M_2 accepts ε. We link the new s_2 to s_1 via an ε –transitions. Finally, we create ε -transitions from each of M_1's accepting states back to s_1. So $M_2 = (\{s_2\} \cup K_1, \sum, \delta_2, s_2, \{s_2\} \cup A_1)$ where $\delta_2 = \delta_1 \cup \{((s_2, \varepsilon), s_1)\} \cup \{((q, \varepsilon), s_1) : q \in A_1\}$



NOTE:

Finite state Machines constructed from Regular expression are typically highly ***non-eterministic*** because of their use of ε-transitions. These FSM's have a large number of unnecessary states. As a practical matter, it is not a problem, since, given an arbitrary NDFSM *M*, we have an algorithm that can construct an equivalent DFSM **M**'. We also have an algorithm that can minimize **M'**

Construct a FSM for the regular expression $(b \cup ab)^*$

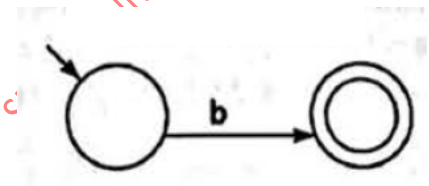
OR

Convert the regular expression $(b + ab)^*$ to an ε- NFA

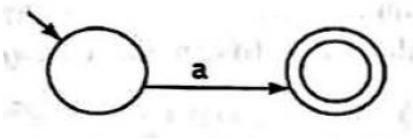
OR

Convert the regular expression $(b, ab)^*$ to a FSM.

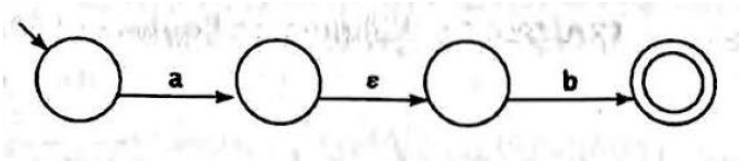
FSM for **b :**



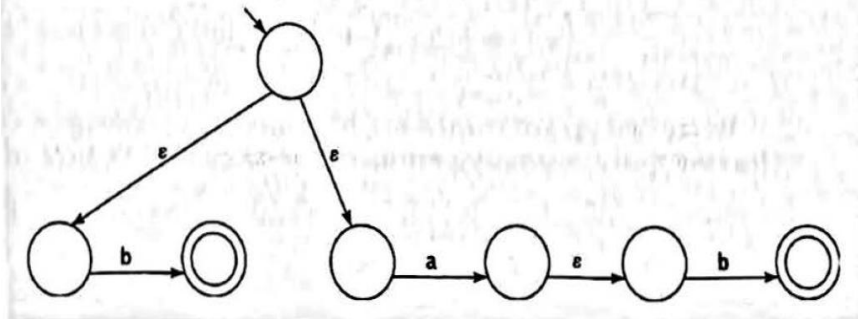
FSM for **a :**



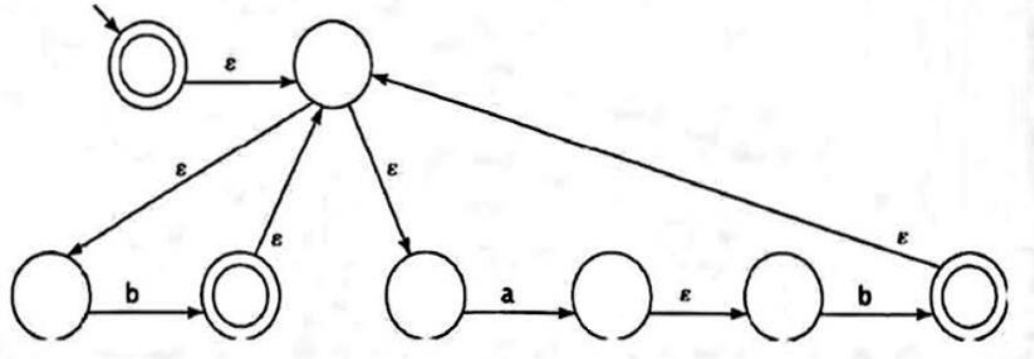
FSM for **ab :**



FSM for **(b U ab) :**



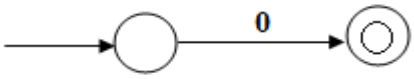
FSM for **(b U ab)***



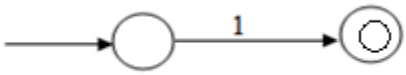
Convert the regular expression $(0 + 1)^* 1 (0 + 1)$ to an ε- NFA or a FSM.

Convert the regular expression $(0 \cup 1)^* 1 (0 \cup 1)$ to an ε- NFA or a FSM

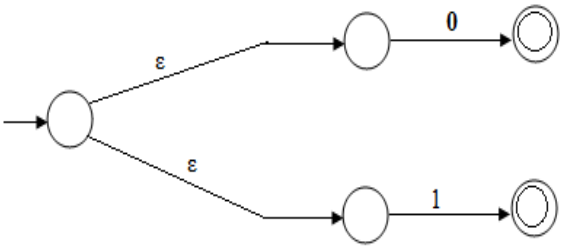
FSM for 0:



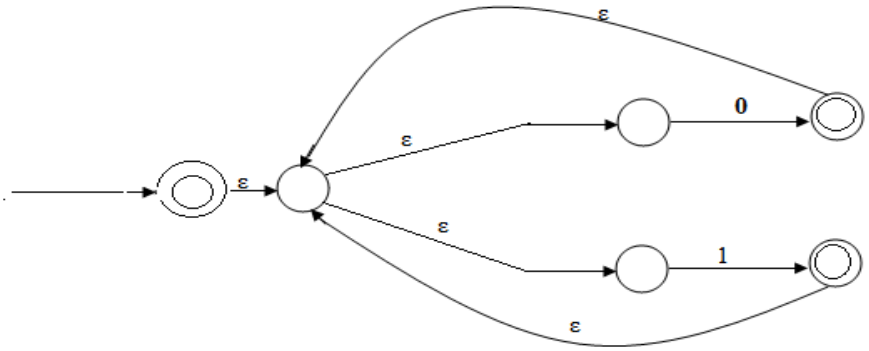
FSM for 1:



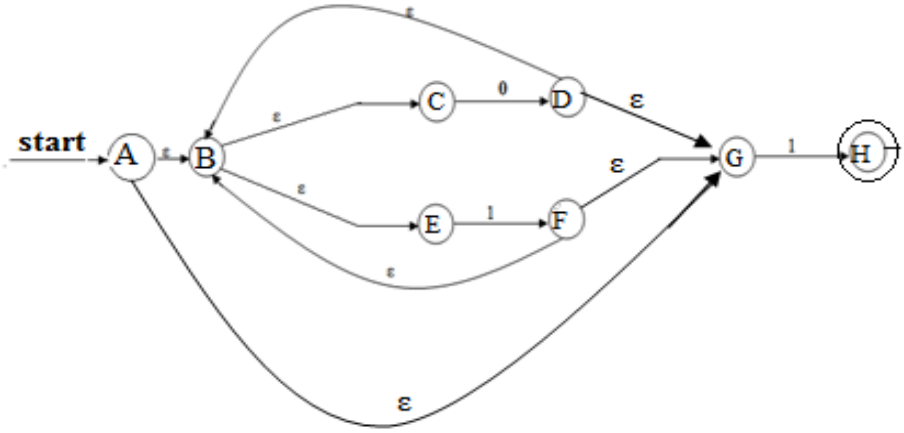
FSM for (0+1):



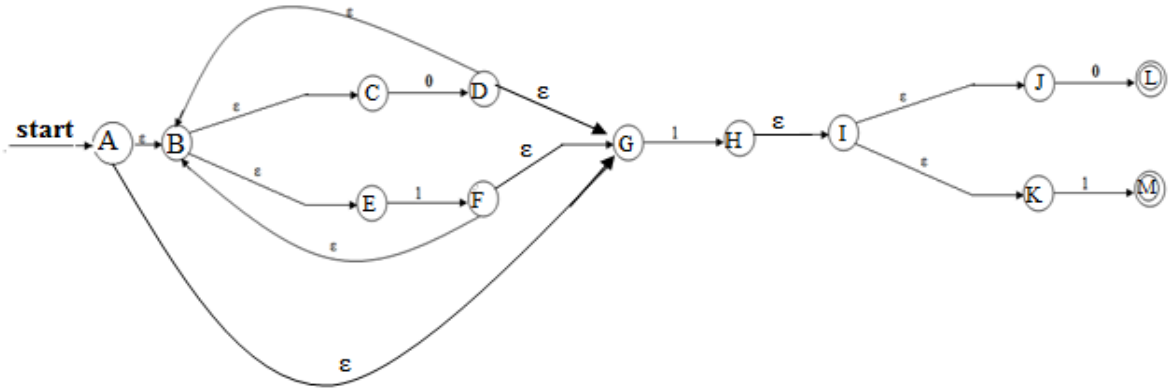
FSM for $(0+1)^*$:



FSM for $(0+1)^* 1$:

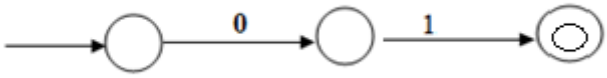


FSM for $(0+1)^*$ 1 (0+1) :

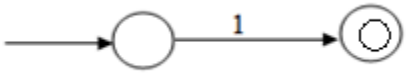


Convert the regular expression $(01+1)^*$ to an **ε- NFA or a FSM**

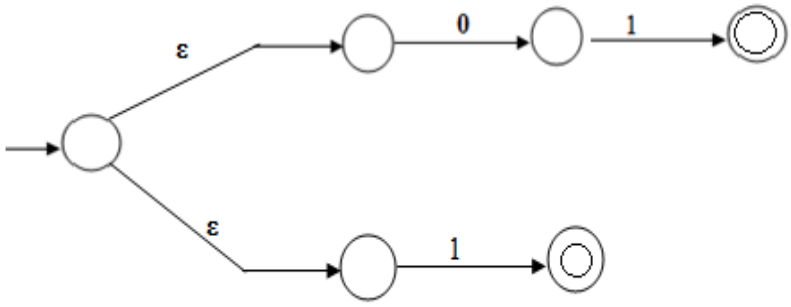
FSM for 01:



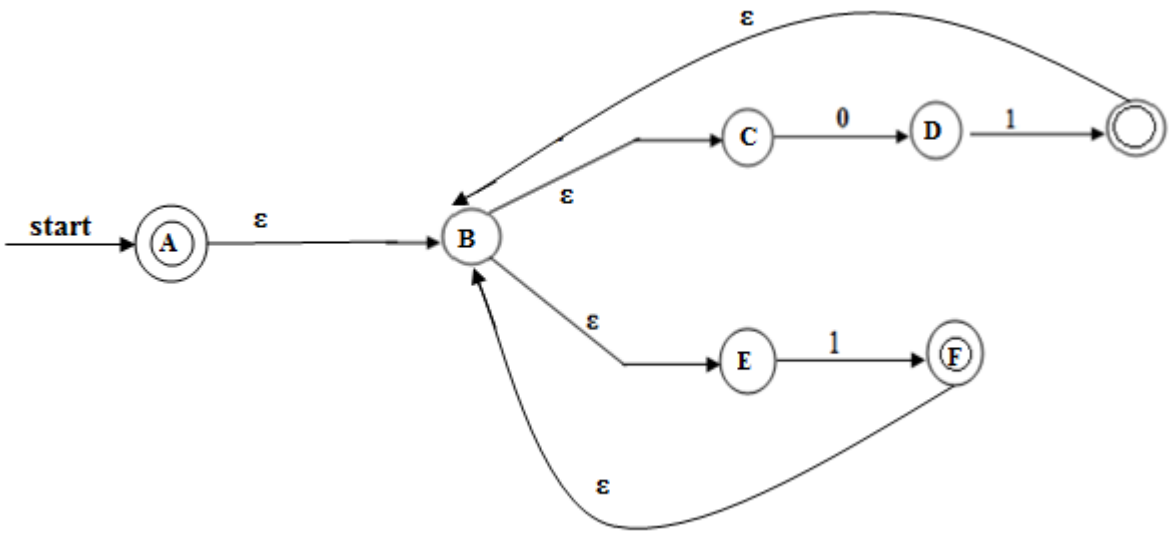
FSM for 1:



FSM for (01+1):

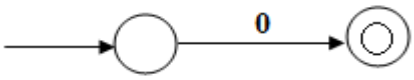


FSM for (01+ 1)*:

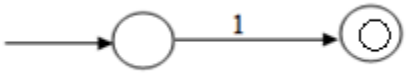


Convert the regular expression (0 U 1)*01 to an **ε- NFA** or **a FSM**

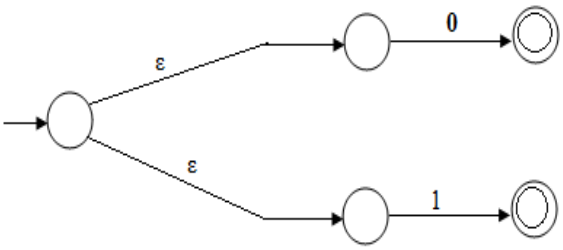
FSM for 0:



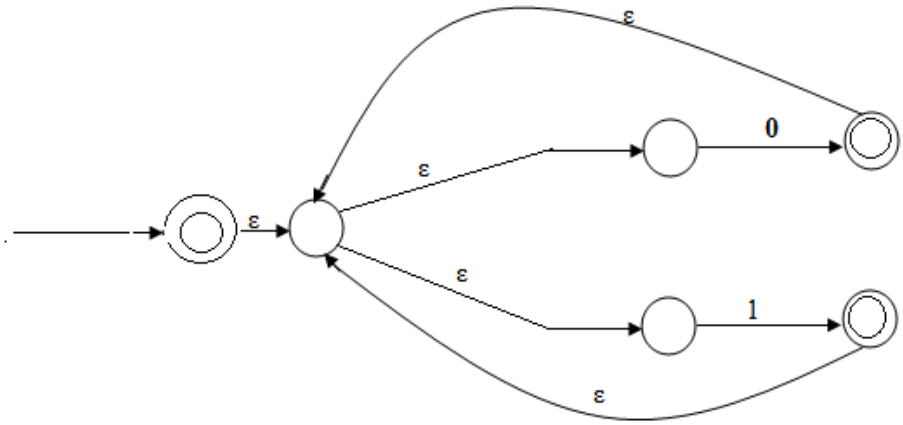
FSM for 1:



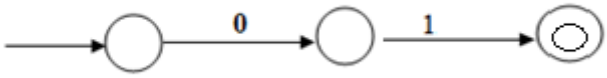
FSM for (0 U 1) or (0+1):



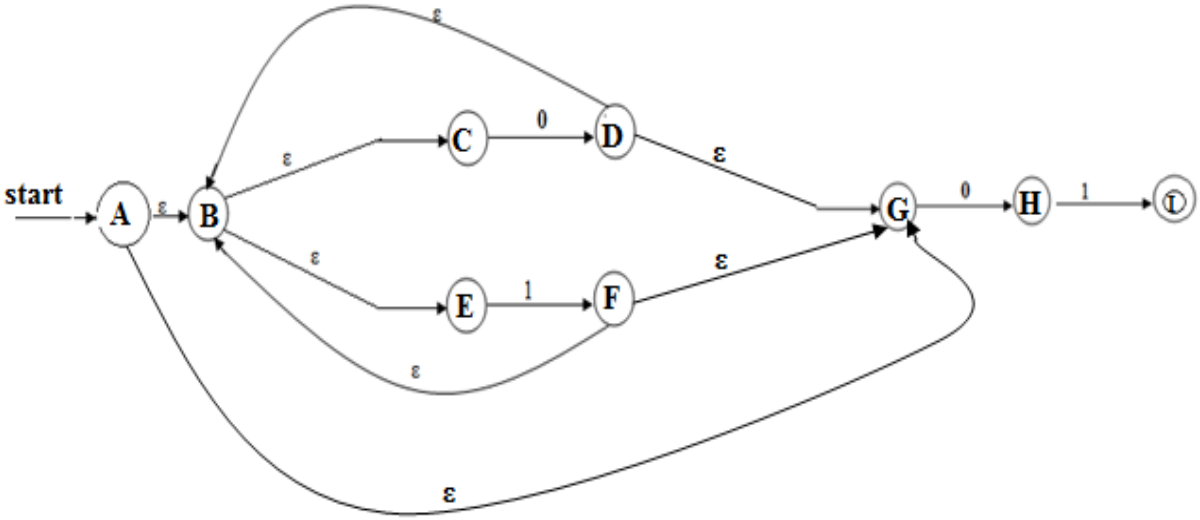
FSM for (0 U 1)* or (0+1)*:



FSM for 01:

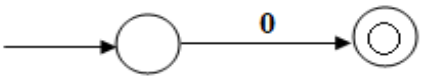


FSM for (0 U 1)*01 or (0+1)*01:

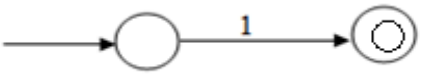


Convert the regular expression $0^* + 1^* + 2^*$ to an ε- NFA or a FSM

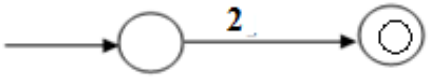
FSM for 0:



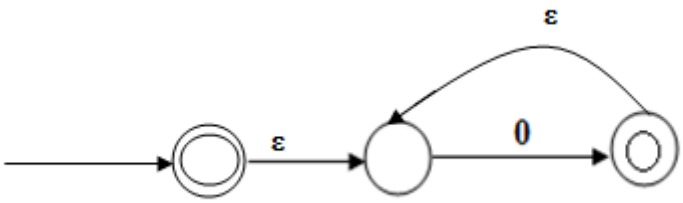
FSM for 1:



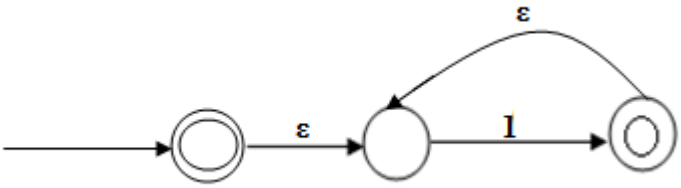
FSM for 2:



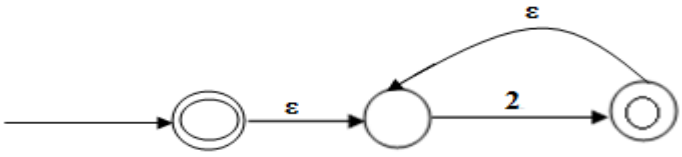
FSM for 0^*:



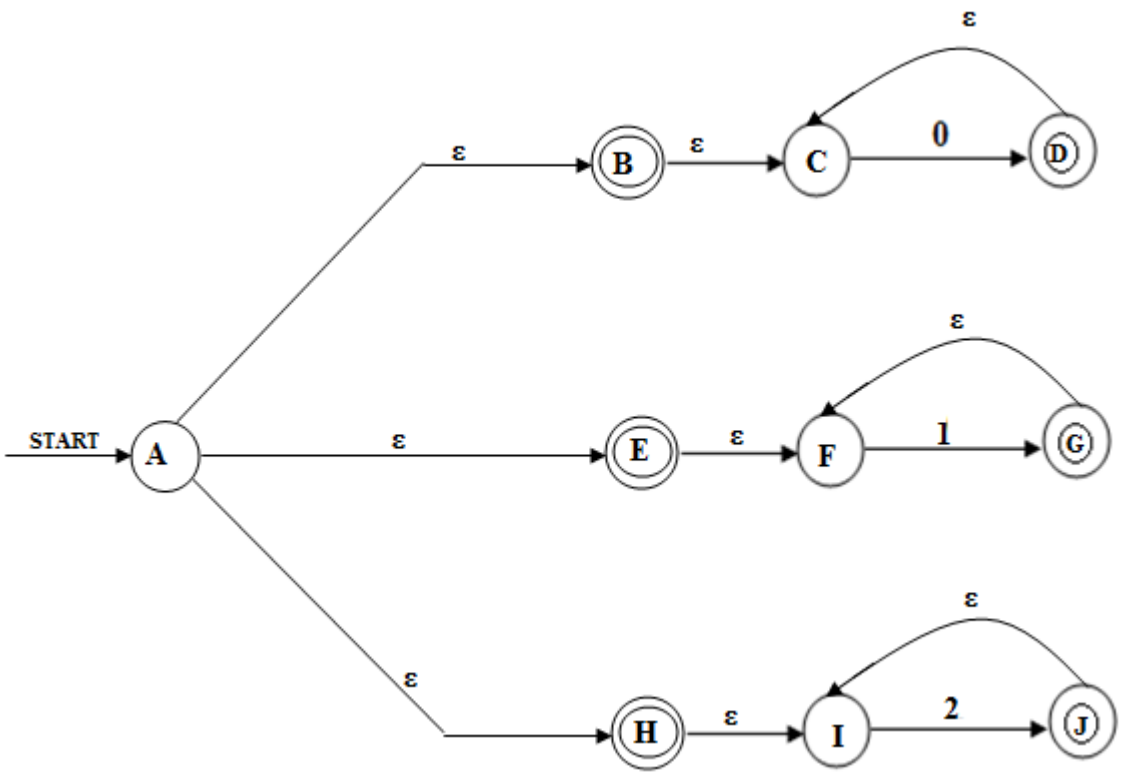
FSM for 1^*:



FSM for 2^*:



FSM for 0 * + 1* + 2*:



Building a Regular Expression from an FSM (State Elimination Technique)

How to build a regular expression for a FSM. Instead of limiting the labels on the transitions of an FSM to a single character or ε, we will allow entire regular expressions as labels.

- For a given input FSM M, we will construct a machine M' such that M and M' are equivalent and M' has only *two states*, start state and a single accepting state.
- M' will also have just one transition, which will go from its start state to its accepting state. The label on that transition will be a regular expression that describes all the strings that could have driven the original machine *M from its start state to* some accepting state

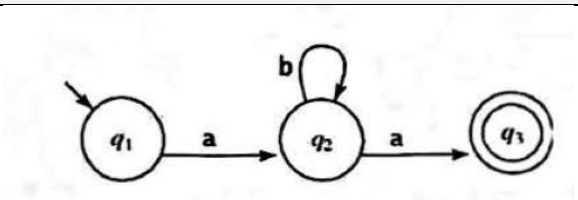
Algorithm to create a regular expression from FSM: (State elimination)

1. Remove any states from given FSM M that are unreachable from the start state
2. If FSM M has no accepting states then halt and return the simple regular expression Ø.
3. If the start state of FSM M is part of a loop (i.e: it has any transitions coming into it), then create a new start state **s** and connects to M 's start state via an ε-transition. This new start state s will have no transitions into it.
4. If a FSM M has ***more than one*** accepting state or if there is ***just one*** but there are any transitions out of it, create a new accepting state and connect each of **M's** accepting states to it via an ε-transition. Remove the old accepting states from the set of accepting states. Note that the new accepting state will have no transitions out from it.
5. At this point, if **M** has only one state, then that state is both the ***start state*** and the ***accepting state*** and **M** has no transitions. So L (M} = {**ε**}. Halt and return the simple regular expression as ***ε.***
6. Until only the start state and the accepting state remain do:
   - **6.1**. Select some state ***rip*** *of M.* Any state except the start state or the accepting state may be chosen.
   - **6.2** Remove ***rip*** *from M.*
   - **6.3** Modify the transitions among the remaining states so that *M accepts the* same strings The labels on the rewritten transitions may be any regular expression.
7. Return the regular expression that labels the one remaining transition from the start state to the accepting state

Consider the following FSM ***M***: Show a regular expression for L(M).

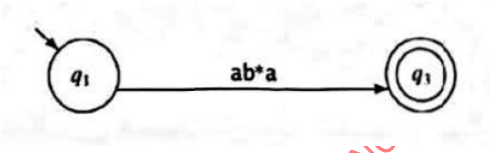
OR

Obtain the regular expression for the following finite automata using state elimination method



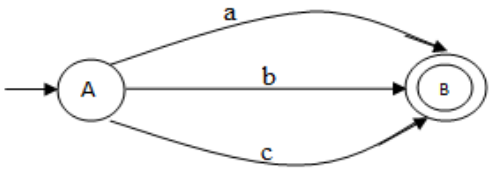
We can build an equivalent machine ***M'*** *by eliminating state* q_2 *and replacing it by a* transition from q_1 to q_3 labeled with the regular expression **ab*a**.

So *M' is:*

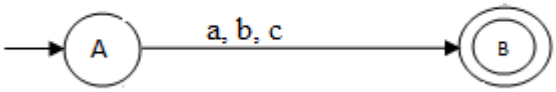


Regular Expression = **ab*a**

Obtain the regular expression for the following finite automata using state elimination method

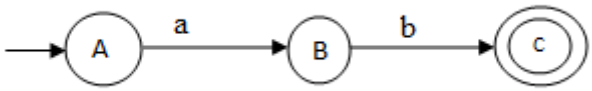


There is no incoming edge into the initial state as well as no outgoing edge from final state. So there is only two states, initial and final.



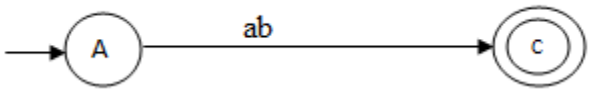
Regular expression = **(a+b+c)** or **(a U b U c)**

Obtain the regular expression for the following finite automata using state elimination method



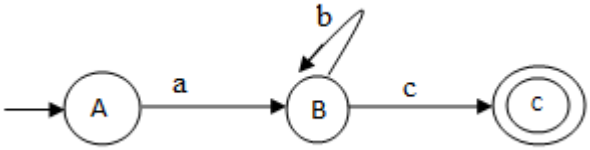
There is no incoming edge into the initial state as well as no outgoing edge from final state.

After eliminating the state B:



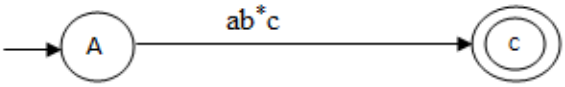
Regular expression = **ab**

Obtain the regular expression for the following finite automata using state elimination method



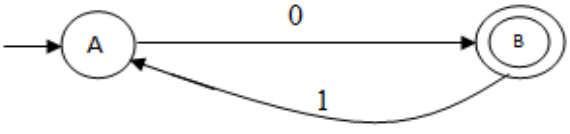
There is no incoming edge into the initial state as well as no outgoing edge from final state.

After eliminating the state B:

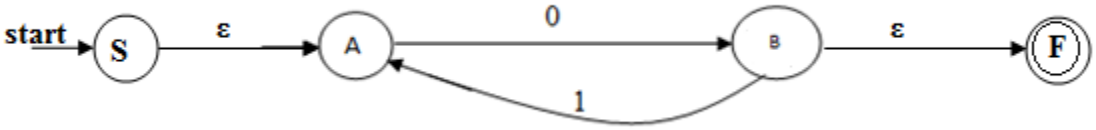


Regular expression = **ab*c**

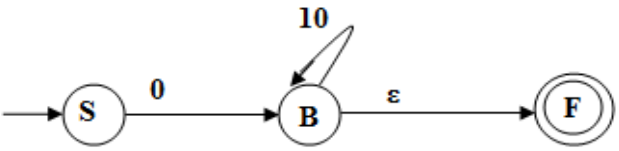
Obtain the regular expression for the following finite automata using state elimination method



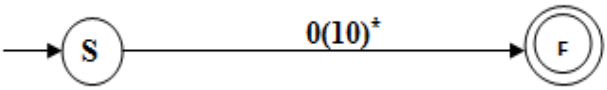
Since initial state has incoming edge, and final sate has outgoing edge, we have to create a new iniatial and final state by connecting new initial state to old initial state through **ε** and old final state to new final state through ε. Make old final state has non-final state.



After removing state A:

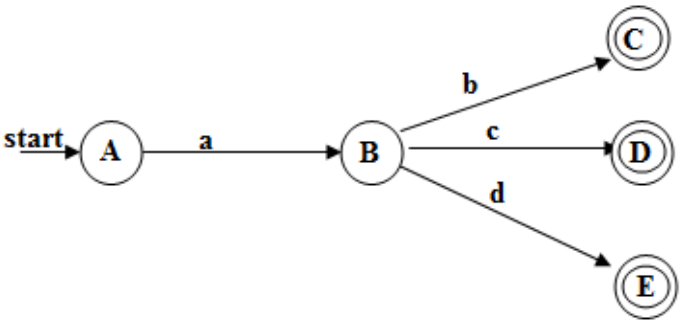


After removing state B:

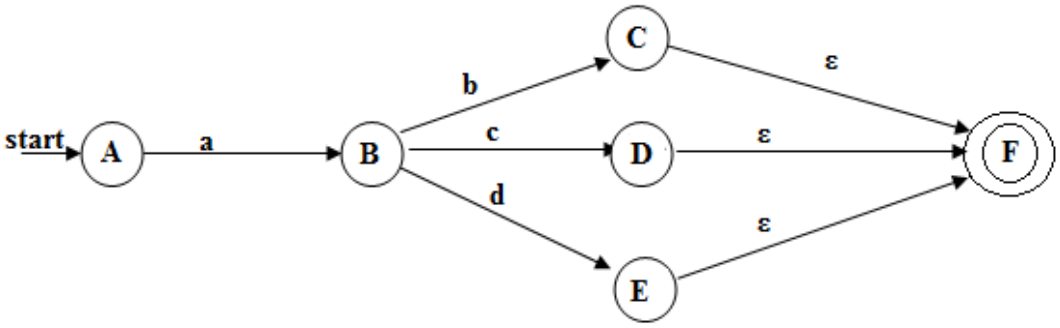


Regular expression: **0(10)***

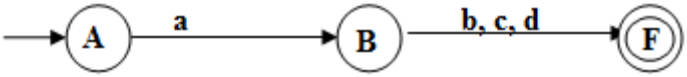
Obtain the regular expression for the following finite automata using state elimination method



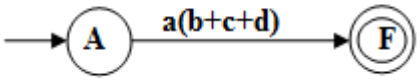
Since there are multiple final states, we have to create a new final state.



After removing states C, D and E:

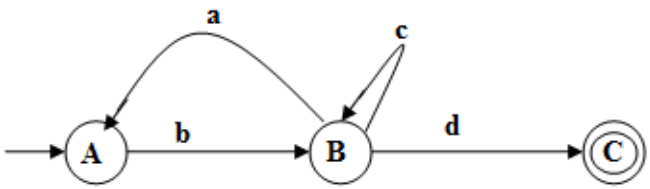


After removing state B:

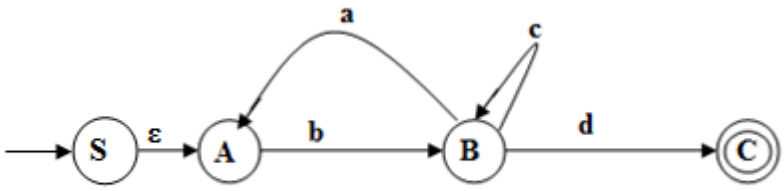


Regular Expression: **a(b+c+d)**

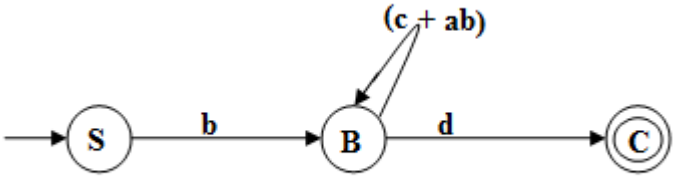
Obtain the regular expression for the following finite automata using state elimination method



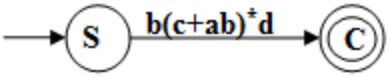
After inserting new start state:



After removing state A:

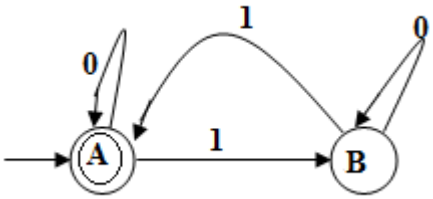


After removing state B:

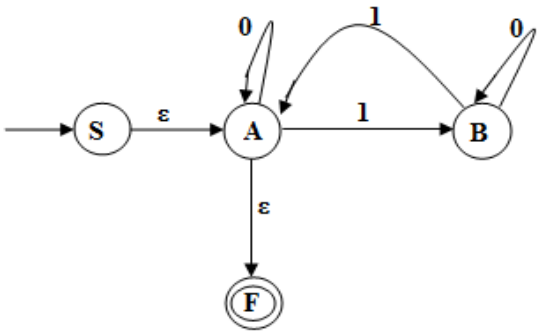


Regular expression: **b(c +ab)*d**

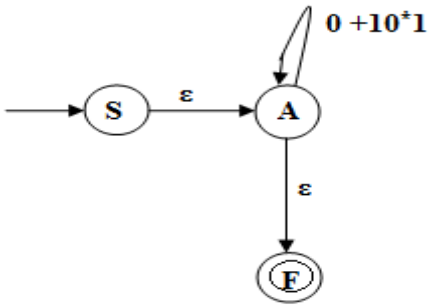
Obtain the regular expression for the following finite automata using state elimination method



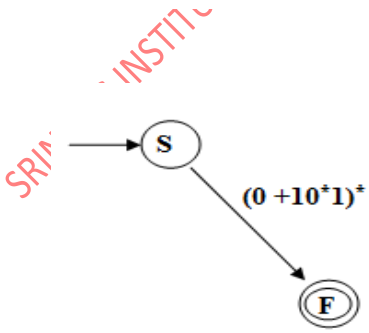
By creating new start and final states:



After removing state B:

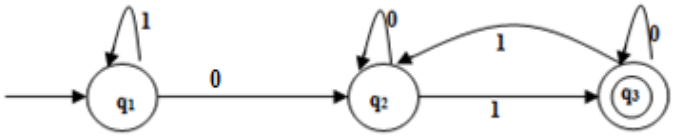


After removing state A:

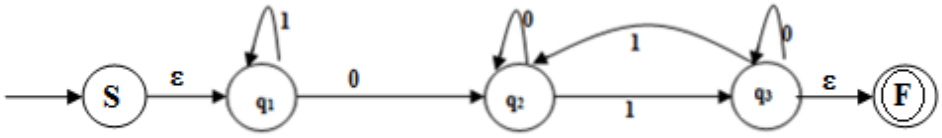


Regular expression: $\mathbf{(0+10^{*}1)^{*}}$

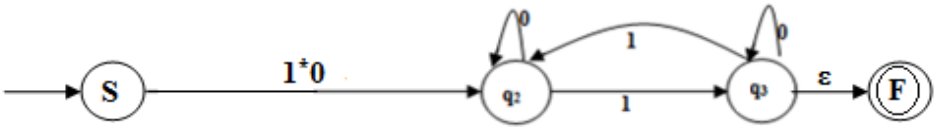
Obtain the regular expression for the following finite automata using state elimination method



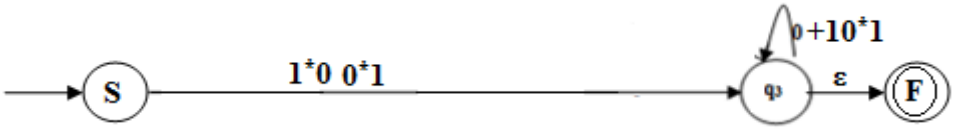
By creating new start and final states:



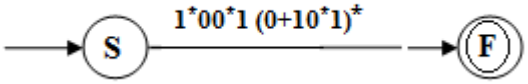
After removing state q_1:



After removing state q_2:

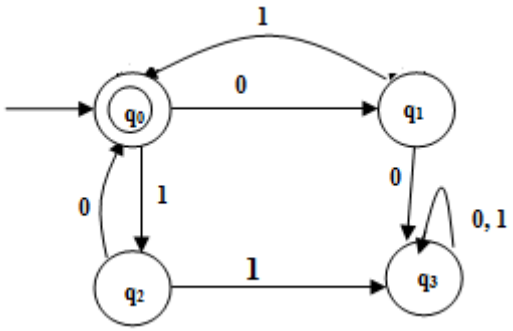


After removing state q_3:

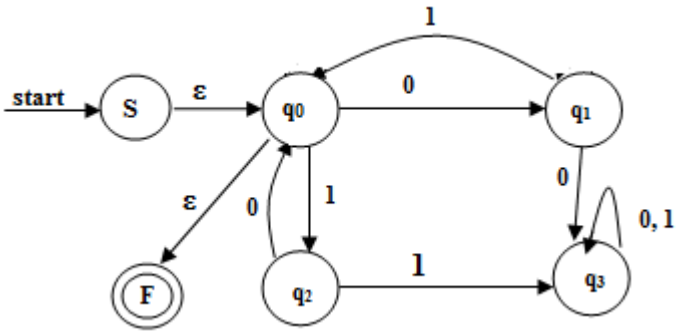


Regular expression: $\mathbf{1^*00^*1(0+10^*1)^*}$

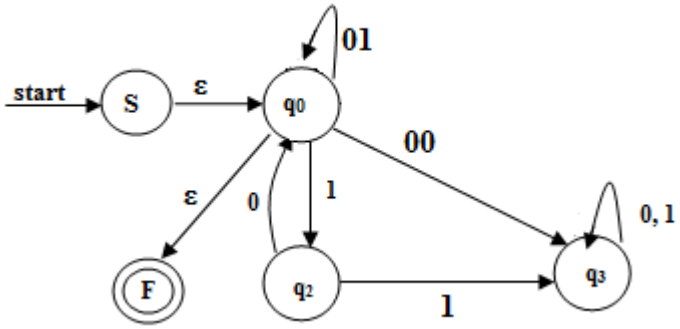
Obtain the regular expression for the following finite automata using state elimination method



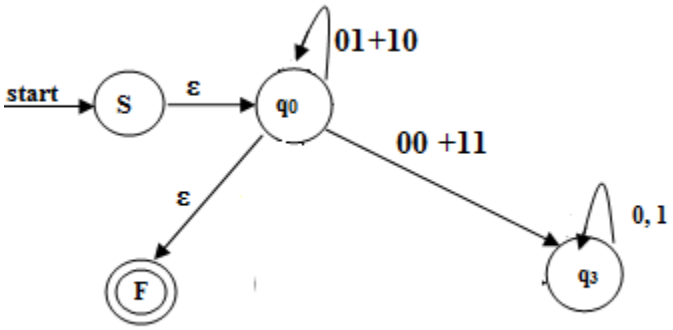
By creating new start state and final state:



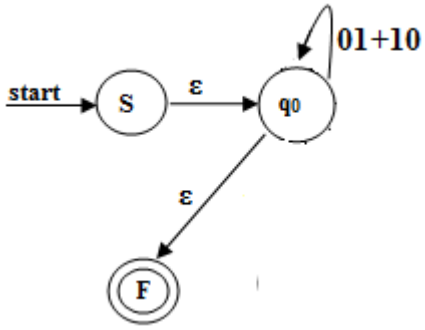
After removing q_1 state:



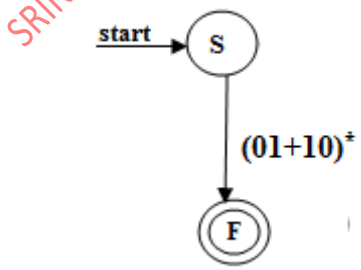
After removing q_2 state:



After removing q_3 state:



After removing q_0 state:

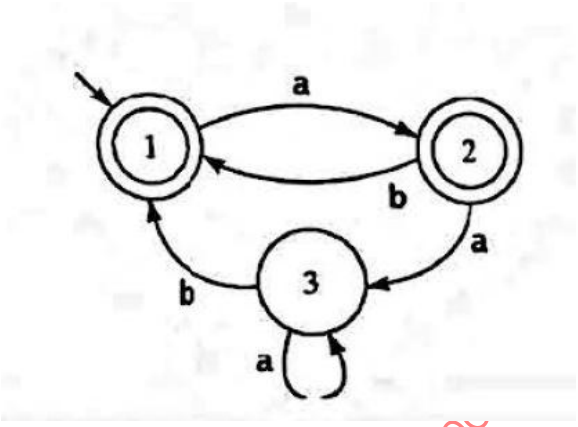


Regular expression: **(01+10)***

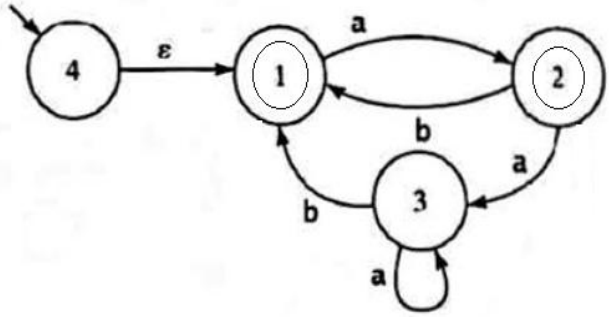
Consider the following FSM M: Show a regular expression for L(M).

OR

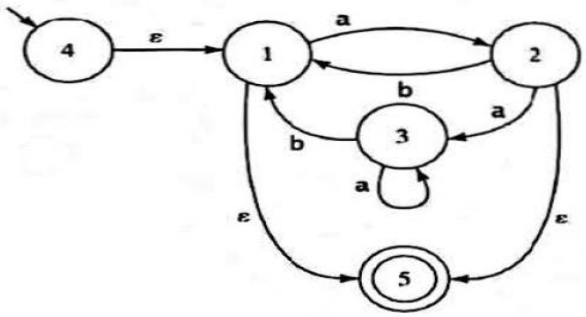
Obtain the regular expression for the following finite automata using state elimination method.



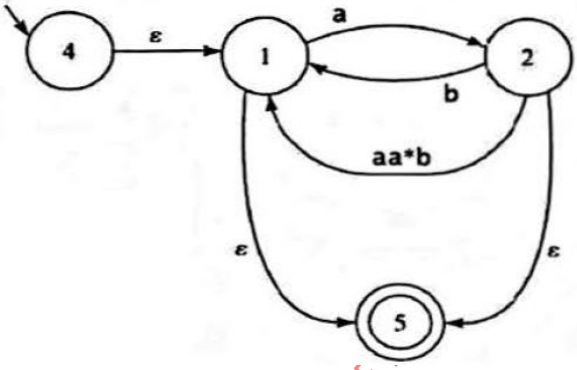
Since start state 1 has incoming transitions, we create a new start state and link that state to state 1 through **ε.**



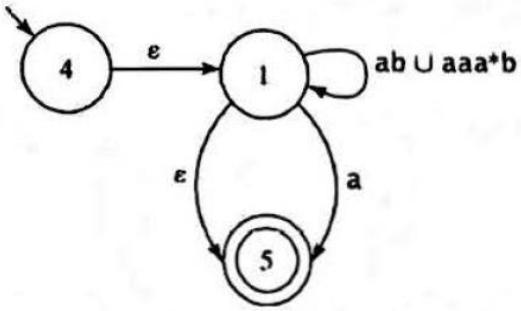
Since accepting state 1 and 2 has outgoing transitions, we create a new accepting state and link that state to state 1 and state 2 through **ε.** Remove the old accepting states from the set of accepting states. **(ie:** consider 1 and 2 has non final states)



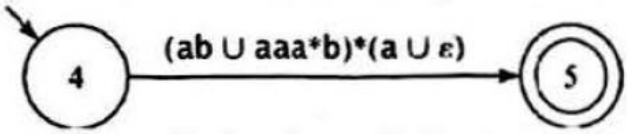
Consider the state 3 has rip and remove that state:



Consider the state 2 has rip and remove that state:



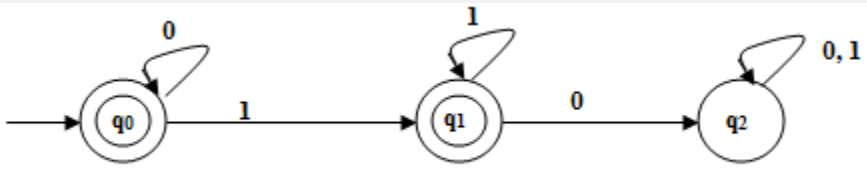
Consider the state 1 has **rip** and remove that state:



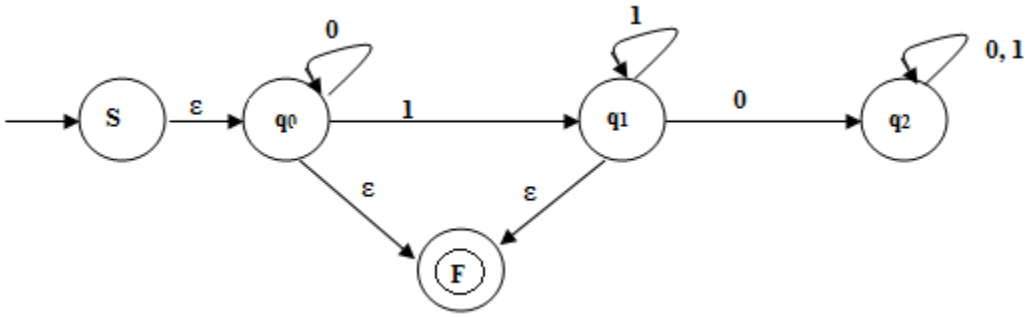
Finally we have only start and final states with one transition from start state 1 to final state 2, The labels on transition path indicates the regular edpression.

Regular Expression = **(ab U aaa* b)* (a U ε)**

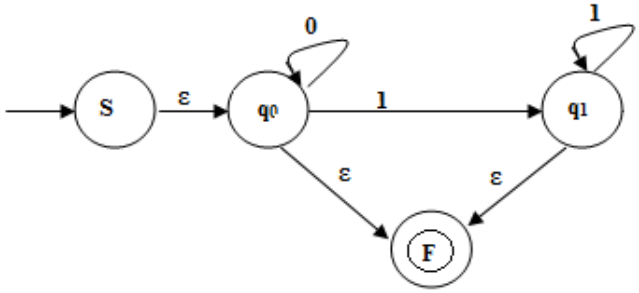
Consider the following FSM M: Show a regular expression for L(M).



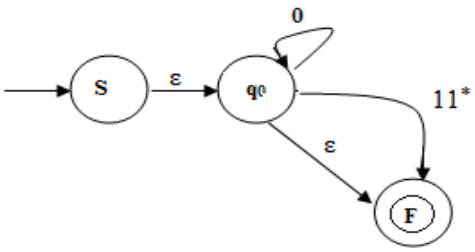
After creating new start and final states:



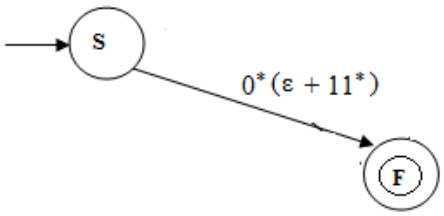
After removing q_2 state:



After removing q_1 state:

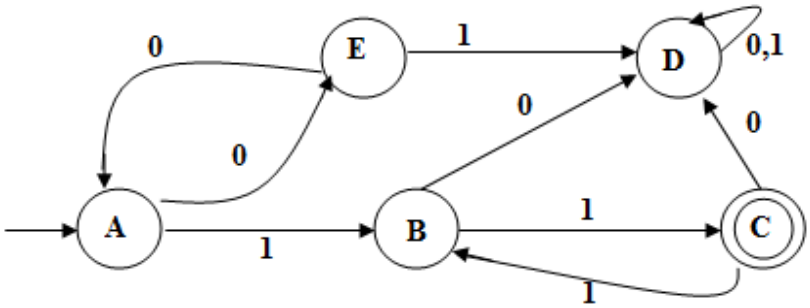


After removing q_0 state:

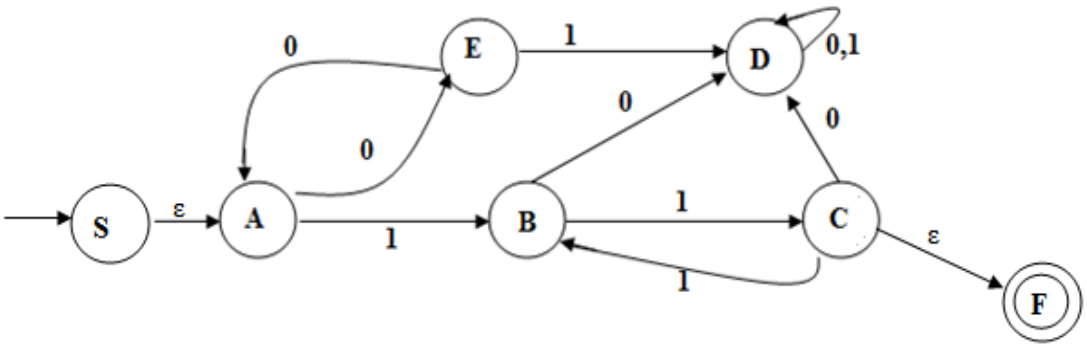


Regular expression: $0^* (\varepsilon + 1^+) = \mathbf{0^* 1^*}$

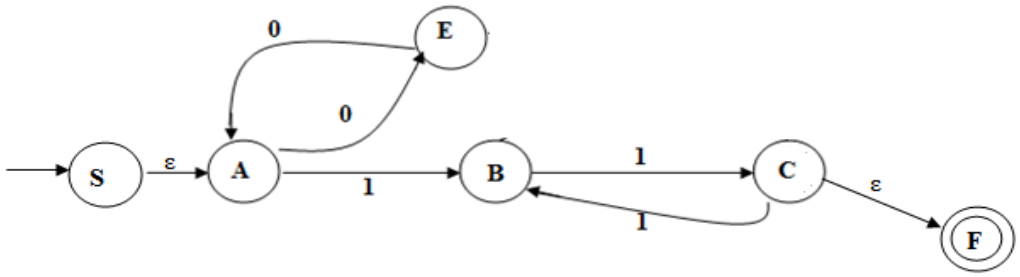
Consider the following FSM M: Show a regular expression for L(M). OR Construct regular expression for the following FSM using state elimination method



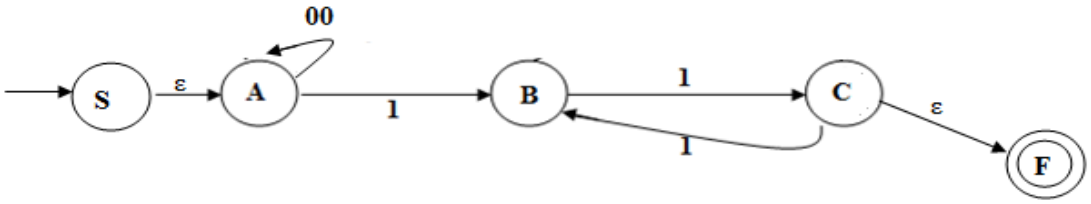
By creating new state and final states.



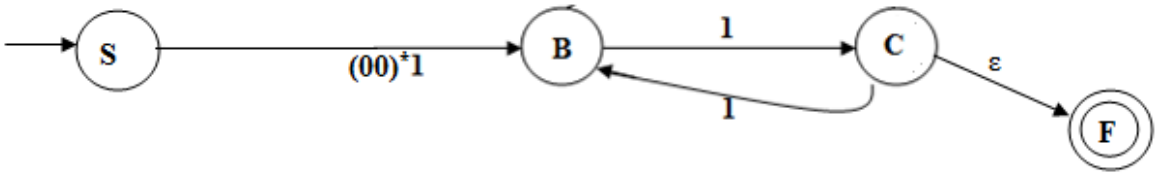
After removing D state:



After removing E state:

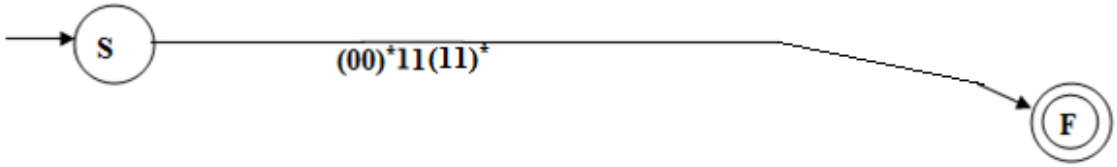


After removing A state:



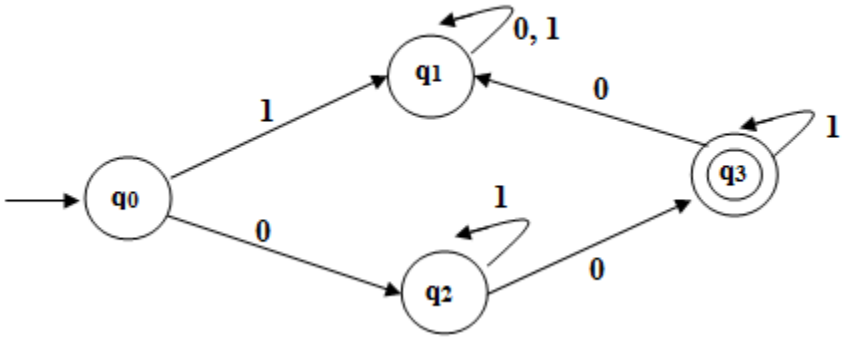
After removing B state:

After removing C state:

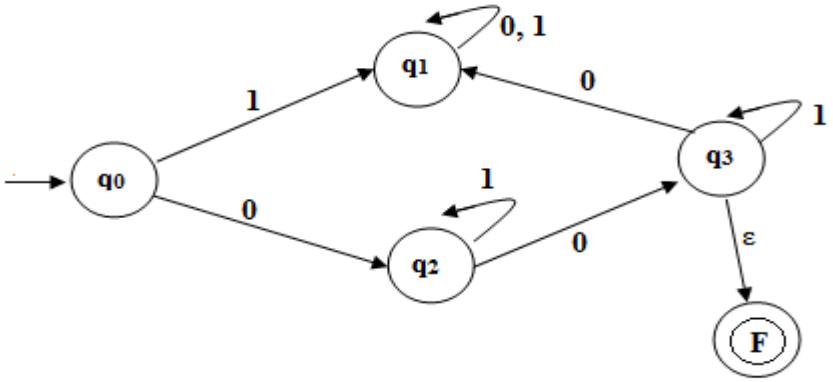


Regular expression = $(00)^*11(11)^*$

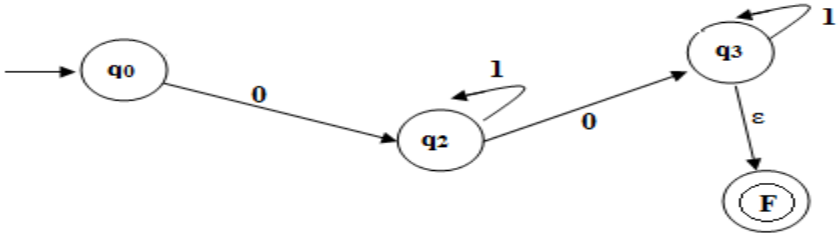
Consider the following FSM M: Show a regular expression for L(M). OR Construct regular expression for the following FSM using state elimination method



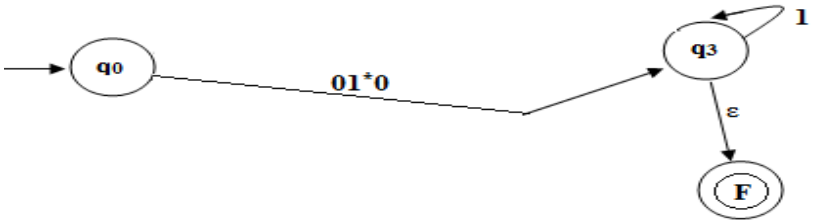
By creating final state.



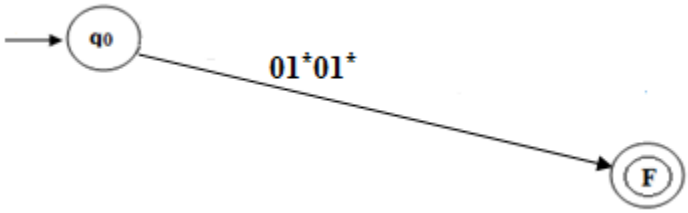
After removing $q_1$state:



After removing $q_2$state:



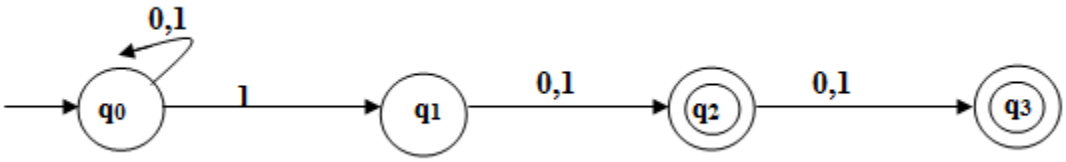
After removing $q_3$state:



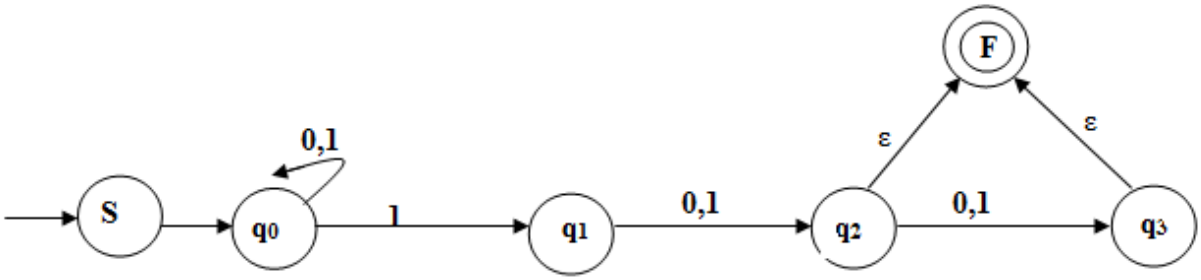
Regular expression= $\mathbf{01^*01^*}$

Consider the following FSM M: Show a regular expression for L(M). OR

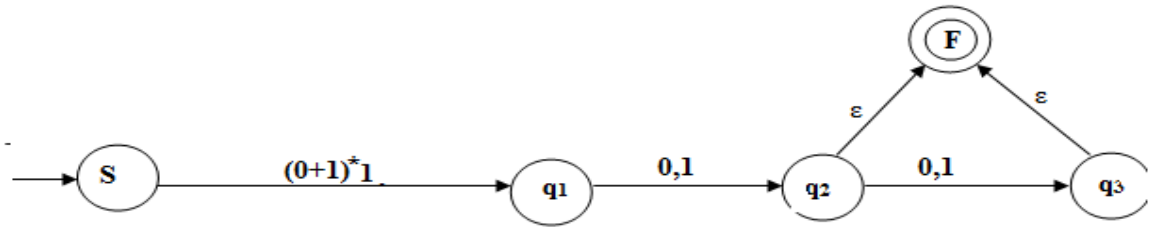
Construct regular expression for the following FSM using state elimination method



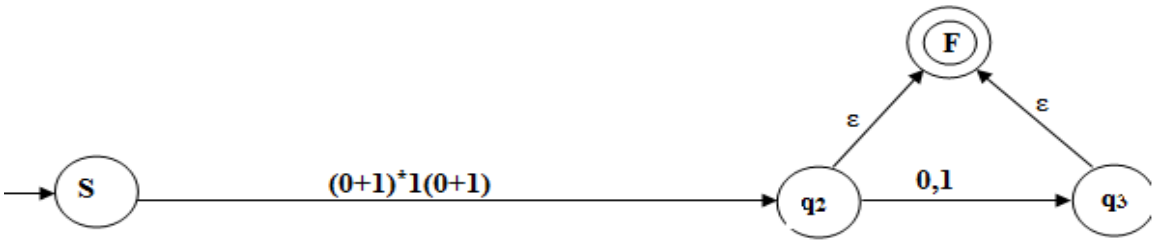
By creating new start and final states:



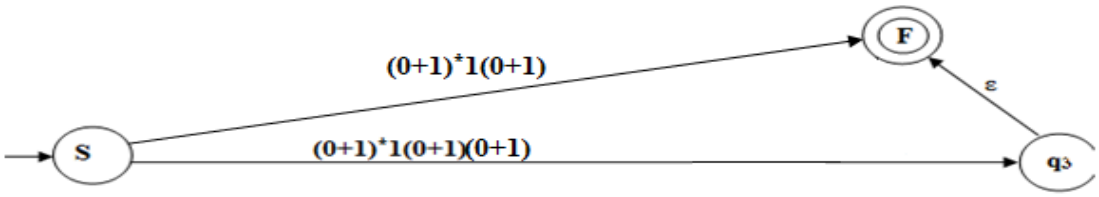
After removing q_0 state:



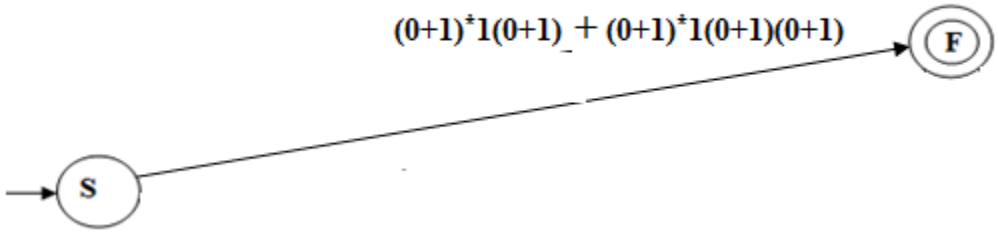
After removing q_1 state:



After removing q_2 state:



After removing q_3 state:



Regular expression: **$(0+1)^*1(0+1) + (0+1)^*1(0+1)(0+1)$**

Kleen's Thereom

Theorem: Every regular language (ie: every language that can be accepted by some FSM) can be defined with a regular expression.

This proof is by construction of FSM (construct a new FSM by using the following steps)

1. Remove any states from given FSM *M that are unreachable from the start state.*
2. If the start state of *M is part of a loop (i.e: it has any transitions coming into it), then* create a new start state *s and connects to M 's start state via an ε-transition. This* new start state *s will have no transitions into it.*
3. If there is *more than o*ne accepting state of *M or if there is just one but there are* any transitions out of it, create a new accepting state and connect each of M's accepting states to it via an ε-transition. Remove the old accepting states from the set of accepting states. Note that the new accepting state will have no transitions out from it.
4. If there is more than one transition between states p and q, collapse them into a single transition.
5. If there is a pair of states *p, q* and there is no transition between them and *p* is not the accepting state and *q* is not the start state, then create a transition from *p* to *q* labeled Ø.
6. At this point, if *M has only one state, then that state is both the start state and the* accepting state and *M has no transitions. So L (M} = {ε}. Halt and return the* simple regular expression as *ε.*
7. If *M has no accepting states then halt and return the simple regular expression Ø.*
8. Until only the start state and the accepting state remain do:

   i. Select some state *rip* of *M.* Any state except the start state or the accepting state may be chosen.

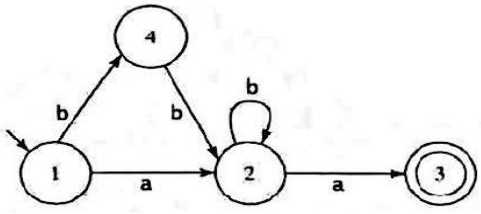
   ii. For every transition from some state *p* to some state , if both ***p*** and ***q*** are not ***rip*** then, using the current labels given by the expressions ***R***, compute the new label ***R '*** for the transition from ***p*** to ***q*** using the formula:

   $$R'(p, q) = R(p, q) \cup R(p, rip)R(rip, rip)^* R(rip, q)$$

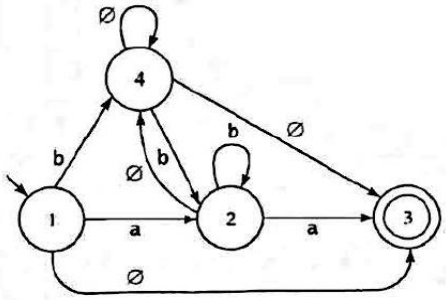
9. Remove *rip* and all transitions into and out of it.

10. Return the regular expression that labels the one remaining transition from the start state to the accepting state.

Construct the regular expression for the following FSM using Kleen's Theorem



By Adding all the required transitions, the FSM becomes:



Ripping States Out One at a Time:

Let *rip* be state 2. Then:

R'(1, 3) = *R(1,* 3) U R(1, *rip)R(rip, rip)*R(rip,* 3).

= R(1, 3) **U** R(1, 2) R(2, 2)***** R(2, 3)

$R(1, 3)^3$ = Ø U a b^* a = ab^*a

$R'(p, q)^k = R(p, q)^{k-1} + R(p, k)^{k-1} (R(k, k)^{k-1})^* R(k,q)^{k-1}$

$R(1, 3)^4 = R(1, 3)^3 + R(1, 4)^3 (R(4, 4)^3)^* R(4,3)^3$

$R(1, 4)^3$ = b

$R(4, 4)^3$ = Ø

$R(4, 3)^3 = bb^*a$ we know that $(Ø)^* = ε$

Regular expression = $\mathbf{ab^*a + bbb^*a}$

Applications of Regular Expressions

- Text editors: which are some programs used for processing the text. Example: UNIX text editor uses the RE for substituting the strings.
- Lexical Analysers Regular expressions are extensively used in the design of Lexical analyzer phase, for example to recognize identifier as : (letter) (letter+digit)*
- Regular expressions are used to search patterns in text.

Manipulating and simplifying regular expressions:

P, Q and R are regular expressions then the identity rules are:

Identity rule	**Example**
$\varepsilon R = R \varepsilon = R$	**$1 \varepsilon = \varepsilon 1 = 1$**
$\emptyset R = R\emptyset = \emptyset$	$1\emptyset = \emptyset 1 = \emptyset$
$\varepsilon^* = \varepsilon$	
$(\emptyset)^* = \varepsilon$	
$\emptyset + R = R+\emptyset = R$	$\emptyset + 1 = 1$
$R + R = R$	$1 \cup 1 = 1$
$RR^* = R^*R = R^+$	$00^* = 0^+$
$(R^*)^* = R^*$	$(1^*)^* = 1^*$
$R^* R^* = R^*$	
$\varepsilon + RR^* = R^*$	**$\varepsilon + 1^+ = 1^*$**
$(P+Q)R = PR + QR$	
$(P+Q)^* = (P^*Q^*) =$	
$R^*(\varepsilon + R) = (\varepsilon + R) R^* =$	
$(\varepsilon + R)^* = R^*$	
$\varepsilon + R^* = R^*$	
$(PQ)^* P = P(QP)^*$	
$R^*R + R = R^*R = R^+$	

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

Let us consider LHS : $(1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^* (0 + 10^*1)$

By taking a common factor as $(1 + 00^*1)$

$(1 + 00^*1) (\varepsilon + (0 + 10^*1)^* (0 + 10^*1))$

We know that $(\varepsilon + R^*R) = R^*$ where $R = (0 + 10^*1)$

The above expression reduces to:

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

Again taking 1 as common factor in above expression:

$(\varepsilon + 00^*)1 (0 + 10^*1)^*$

By applying $(\varepsilon + 00^*) = 0^*$

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

Hence the two regular expressions are equivalent.

Prove that $\varepsilon + 1^*(011)^* (1^*(011)^*)^* = (1 + 011)^*$

Let LHS is

$\varepsilon + 1^*(011)^* (1^*(011)^*)^*$

By considering $1^*(011)^*$ as R and applying $(\varepsilon + R^*R) = R^*$

LHS = $(1^*(011)^*)^*$

Put P = 1 and Q = 011^* and by applying

$(P^* Q^*)^* = (P + Q)^*$

$= (1 + 011)^*$

= RHS

LHS = RHS

Hence the proof

State True or False : Every subset of regular language is regular

Let us consider one regular language as $L_1 = \{ 0^* \}$

The subset of L_1 say $L_2 = \{ 0^{2n} \mid n \geq 1 \}$ is not a regular language. (Its not possible to construct FSM)

Hence the given statement is ***False.***

Check the equivalence of the regular expressions: $(a^*bbb)^*a^*$ and $a^*(bbba^*)^*$

$(ab)^*a = a(ba)^*$

By identity rule $(PQ)^* P = P(QP)^*$

Let us map this rule with the given expression:

$P = a^*$ $Q = bbb$

Hence the given expressions are equivalent.

$(ab)^*a = a(ba)^*$: True, by using the above identity rule.(P = a and Q= b)

Exercises

Write a regular expressions to describe each of the following languages

1 **{**w € {a. b}* : every a in **w** is immediately preceded and followed by b**}**.

Regular expression = (b + bab)*

2 **{** w € {a. b}* : *w* does not end in ba}.

Regular expression = ε + a + (a + b)* (b + aa)

3 **{** w € {0,1}* : *w* has 001 as a substring}

Regular expression = (0 + 1)* **001** (0 + 1)*

4 **{** w € {0, 1}* : ***w*** does not have **001** as a substring}

Regular expression = (1 + 01)* 0*

5 **{**w € {a. b}* : w has both **aa** and bb as substrings}

Regular expression = (a + b)* **aa** (a + b)* **bb** (a + b)* + (a + b)* **bb** (a + b)* **aa** (a + b)*

6 **{**w € {a. b}* : w has both **aa** and aba as substrings}

Regular expression = (a + b)* **aa** (a + b)***aba** (a + b)* + (a + b)* **aba** (a + b)* **aa** (a + b)* + (a + b)* **aaba** (a + b)* + (a + b)***abaa** (a + b)*

7 **{**w € {0 ,1}* : none of the prefixes of w ends in **0**}

Regular expression = **1***

8 $\{w \in \{a, b\}^* : \#_a(w) \equiv_3 0\}$.

Regular expression = **(**b***a** b***a** b***a)***b*

9 $\{w \in \{a, b\}^* : \#_a(w) \leq 3\}$ (Number of **a**'s in ***w*** is at most 3)

Regular expression = b* (a + ε) b* (a + ε) b* (a + ε) b*

10 **{**w € {a. b}* : ***w*** contains exactly two occurrences of the substring **aa**}

Regular expression = (b + ab)***aaa*** (b + ba)* **+** (b + ab)***aab*** (b + ab)****aa***(b + ba)*

Simplify each of the following regular expressions

a (a U b)* (a U ε) b* = **(a U b)***

b (**Ø*** + b) b*.

We know that **Ø* = ε**

(ε +b) b* = b* + bb* = **b***

c (a U b)*a* U b = **(a U b)***

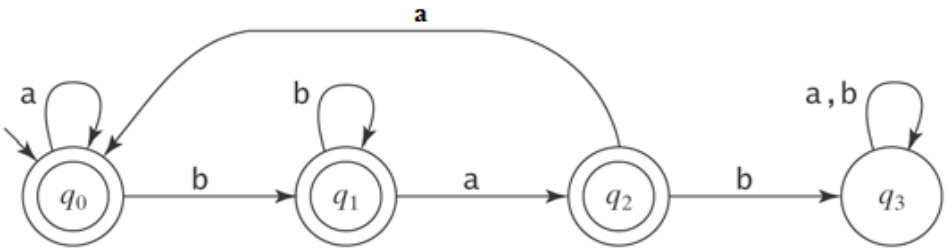
d ((a U b)*)* = **(a U b)***

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

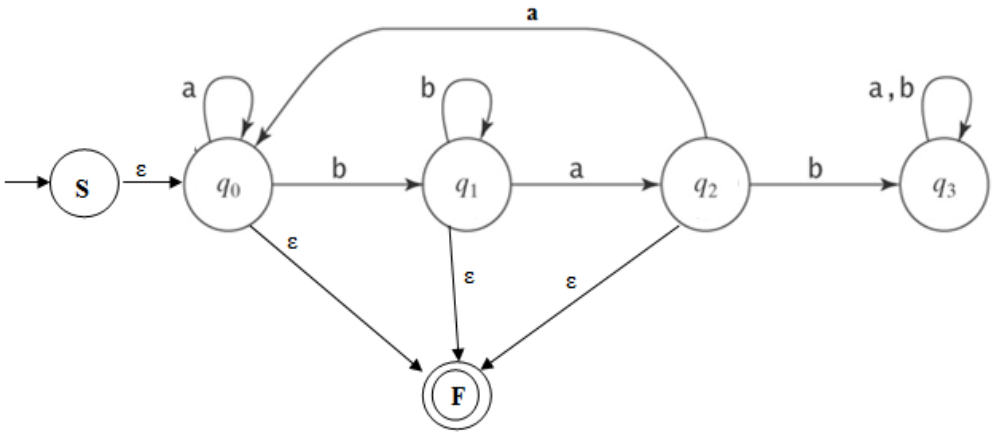
Let $L = \{a^n b^n : 0 \leq n \leq 4\}$.

Regular expression = (ε + ab + aabb + aaabbb + aaaabbbb)

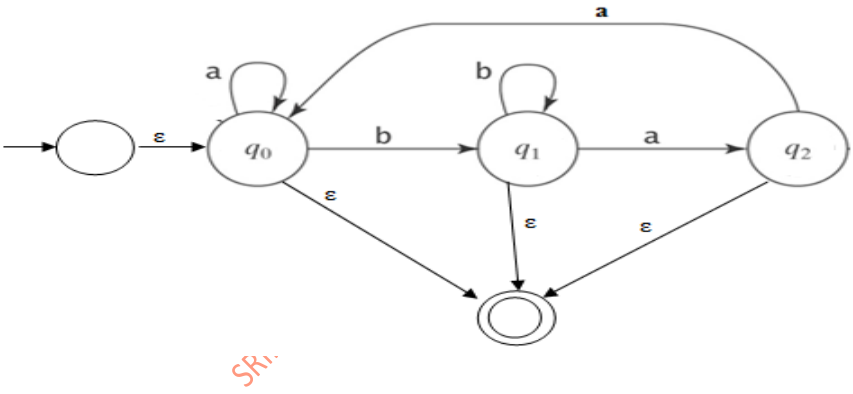
Write the regular expression for the following FSM ***M*** using state elimination method.



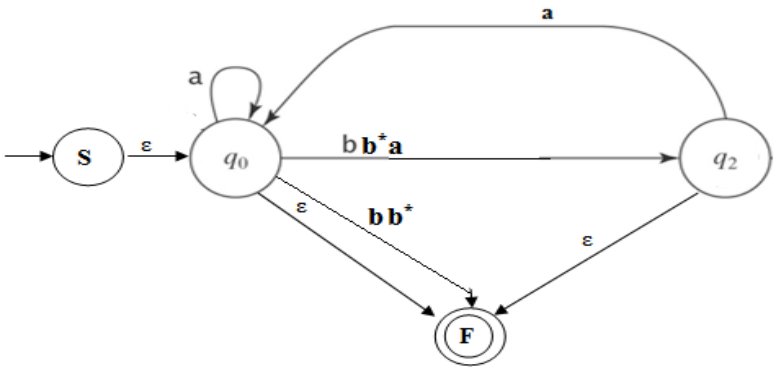
By creating a new start and final state.



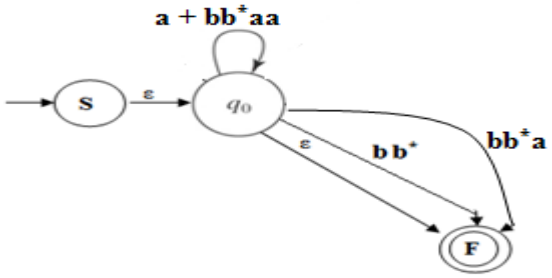
After removing state q_3:



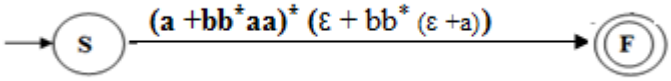
After removing state q_1:



After removing state q_2:

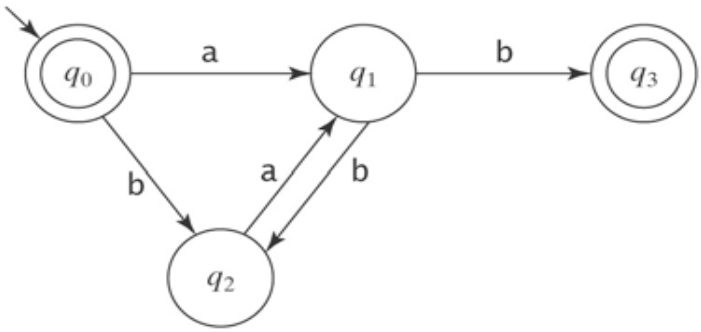


After removing state q_0:

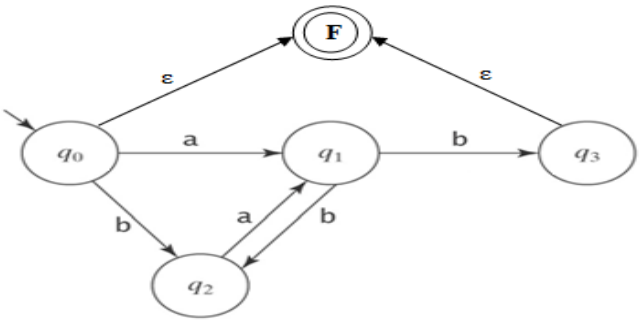


Regular expression = (a + bb*aa)* **(**ε + bb*(ε +a)**)**

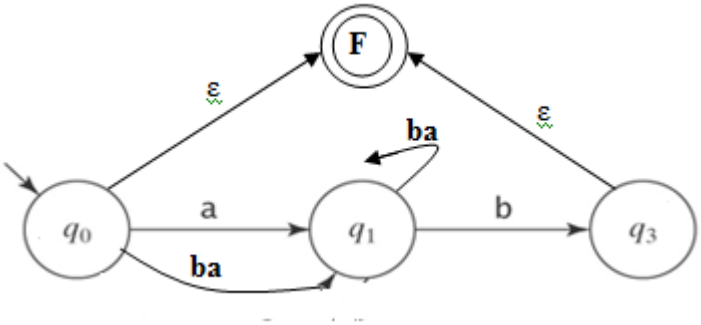
Write the regular expression for the following FSM ***M*** using state elimination method



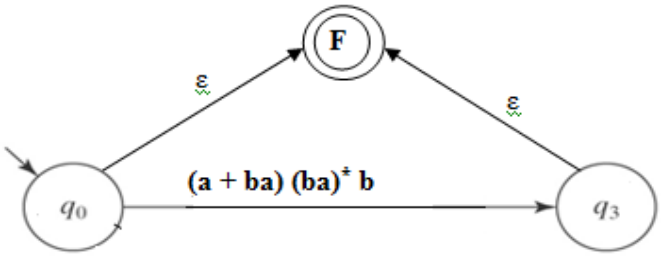
By creating a new Final state:



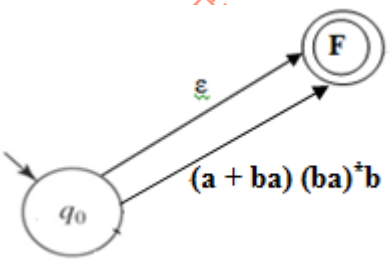
After removing q_2 state:



After removing q_1 state:



After removing q_3 state:



Regular expression = **ε + ((a + ba) (ba)* b)**

Regular Grammars

So far we have considered ***two equivalent*** ways to describe exactly the class of regular languages:

- Finite state machines.
- Regular expressions.

We now introduce a third:

• Regular grammars (sometimes also called right linear grammars).

Define regular Grammar

A regular grammar G is a quadruple (**V**, ∑, **R**,**S**)

where*:*

- **V** is the rule alphabet, which contains Non-terminal symbols and Terminal symbols.
- ∑ is the set of terminal symbols (Subset of V)
- **R** is finite set of rules of the form X → Y
- **S** is the start symbol, which is a non-terminal symbol.

Example**:** G = ({ A, C, a, b}, {a, b}, R, **A**) where rule R is: A → ε, A → b, A → aC and C →a

Non-Terminal and Terminal Symbols:

Non-terminal Symbols: symbols that are used in the grammar but that do not appear in strings in the language. In above example ***A, C*** are non-terminal (***Variable***) symbols.

Terminal Symbols: symbols that can appear in strings generated by G. In above example ***a, b*** are terminal symbols.

Rules R of any Regular Grammar:

Rule **R** is of the form **X** → **Y** must satisfy the following 2 conditions:

1. Left-hand side contains only one symbol that must be a non- terminal.
2. RHS contains **ε or** a single character (terminal) **or** a single character (terminal) followed by a single non-terminal

Example: A → ε or A → b or A → aA are legal Rules

BA → ε , A → aSa are not legal rules.

Language generated by a Grammar:

The language generated by a grammar G =(V, ∑, R, S), denoted L(G), is the set of all strings ***w*** in $\sum^*$ such that it is possible to start with **S**, apply some finite set of rules in **R**, and derive ***w.***

To generate any string by using regular grammar G; to start with **S**, apply derivation step, by replacing non-terminal symbol in each derivation step until the required string is generated.

Regular Grammars and Regular Languages**:**

Theorem : The Regular grammar defines exactly the regular languages or The class of languages that can be defined with regular grammars is exactly the regular languages.

To ***prove*** this theorem one must prove that for given regular grammar it is possible to construct equivalent FSM or from FSM it is possible to get the regular grammar.

Method for conversion of FSM to Regular Grammar G:

Conversion of FSM to regular Grammar **G** = ({ $A_0, A_1, \ldots\ldots A_n$}, ∑, R, A_0) is as follows:

Where **R** is the set of production rules can be defined by following rules:

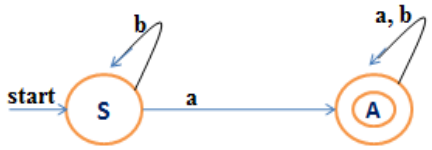
1. $A_i \rightarrow aA_j$ is a production rule if $\delta(A_i, a) = A_j$ where A_j can be any final or non-final state but not the final state #
2. $A_i \rightarrow a$ is a rule if $\delta(A_i, a) = A_j$ where A_j is final state #
3. $A_i \rightarrow \varepsilon$ is a rule if A_i is any final state other than final state #

Note: Final state # of FSM is not the part of regular grammar rule.

Let us consider one regular language L = { strings of **a**'s and **b**'s with at least one **a**}

The ***regular expression*** which defines the regular language L is = $(a + b)^*$ **a** $(a + b)^*$

The ***FSM*** which accepts the regular language L:



By applying the above methods (Above FSM doesnot contain final state as #)

The regular Grammar which defines the regular language L is:

Regular grammar **G** = **(** { S, A, a, b}, { a, b}, R, S **)** where S is the start Non-terminal symbol of grammar G and R is the rules defined as:

S→ aA

S → bS

A → aA

A → bA

A → ε

Method for conversion of Regular Grammar G to FSM:

1. Create in FSM *M* a separate state for each non terminal in ***V***.
2. Make the state corresponding to ***S*** the start state.
3. If there are any rules in ***R*** of the form $X \rightarrow w$, for some **w** € ∑ then create an additional state labelled **#.**
4. For each rule of the form $X \rightarrow wY$, add a transition from *X* to *Y* labelled **w**.
5. For each rule of the form $X \rightarrow w$, add a transition from *X* to # labelled ***w***.
6. For each rule of the form $X \rightarrow \varepsilon$, mark state *X* as accepting.
7. Mark state # as accepting.

Consider the following regular grammar *G*:

S → aT

T → bT

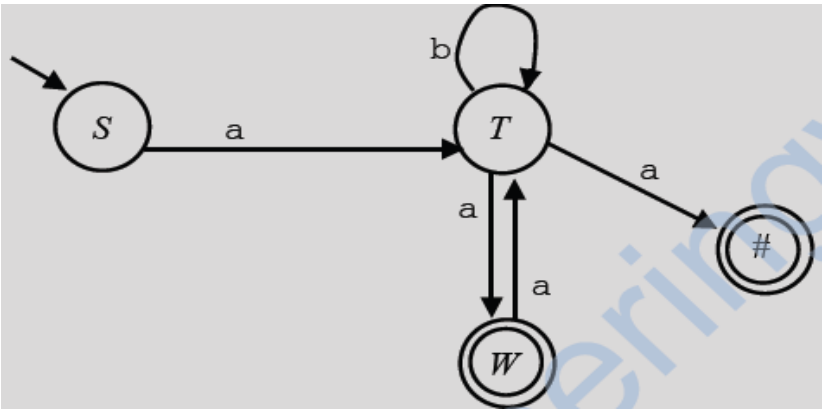
T → a

T → aW

W → ε

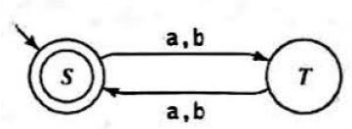
W → aT

The equivalent FSM for the given regular grammar G:



Write the regular grammar for the language L = {w € {a, b }*: |w| is even}.

The following DFSM ***M*** accepts L:



By converting FSM to regular grammar G:

G = ({S, T, a, b}, {a, b}, R, S) where R is the rule defined as follows:

S → aT

S → bT

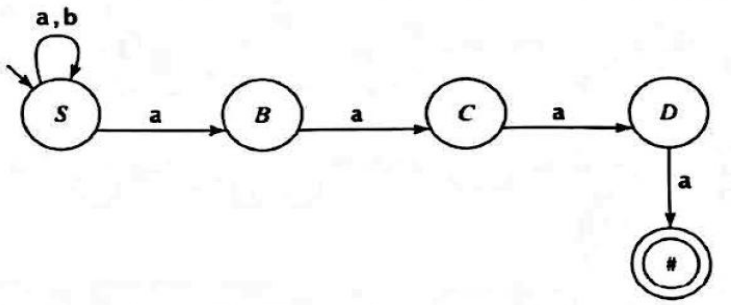
T → aT

T → bT

S→ ε

Write the regular grammar for the language L = {w € {a, b }* **:** ***w*** ends with pattern ***aaaa***}.

The following FSM (NDFSM) **M** accepts L:



By converting FSM to regular grammar G:

G = ({S, B, C, D, a, b}, {a, b}, R, S) where R is the rule defined as follows:

S → aS

S → bS

S → aB

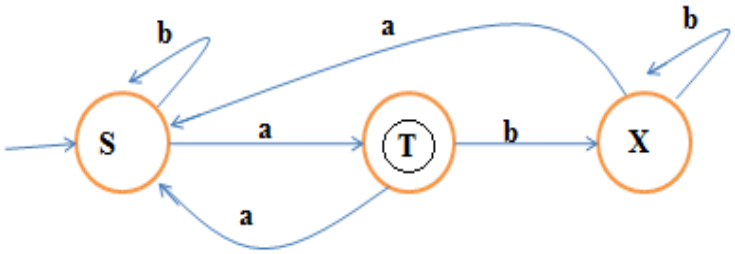
B → aC

C→ aD

D→ a

Write the regular grammar for L = {w € {a, b}*: ***w*** contains an ***odd*** number of ***a***'s and ***w*** ends in ***a***}. Also generate the string ***baaba*** by using this regular grammar.

DFSM which accepts L is:



Regular grammar G = ({S, T, X, a, b}, {a, b}, R, S) where R is the rule defined as follows:

S → bS

S → aT

T → aS

T→ bX

T→ ε

X→ aS

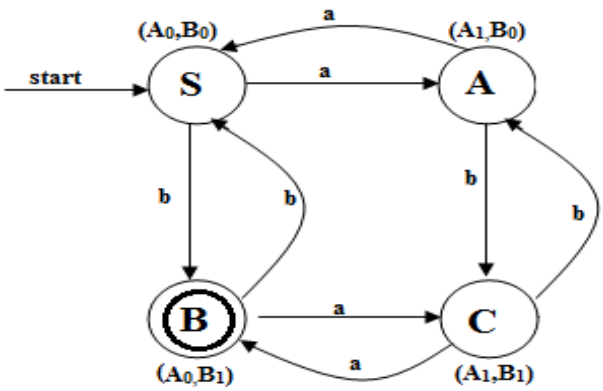
X→ bX

To generate the string ***baaba***, start with S apply derivation process, by replacing non terminal symbol in each step, until the required string is generated.

$$S \Rightarrow bS$$
$$\Rightarrow baT$$
$$\Rightarrow baaS$$
$$\Rightarrow baabS$$
$$\Rightarrow baabaT$$
$$\Rightarrow baaba$$

Show a regular grammar for the language: L ={ w € {a, b }*: *w* contains an even number of **a**'s and an odd number of **b**'s }

DFSM which accepts L is:



Regular grammar G = ({S, A, B, C, a, b}, {a, b}, R, S) where R is the rule defined as follows:

S → aA

S → bB

A → aS

A→ bC

B→ aC

B→ bS

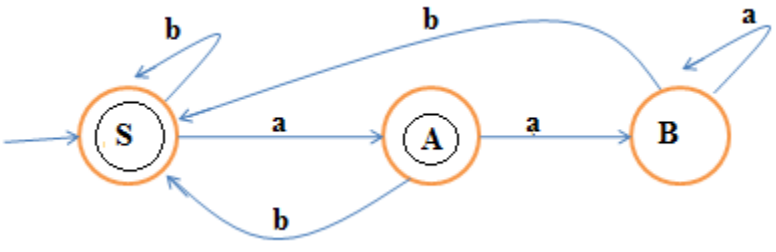
B→ ε

C→ aB

C→ bA

Show a regular grammar for the language: L ={ w € {a, b }*: *w* does not end in ***aa*** }

DFSM which accepts L:



Regular grammar G = ({S, A, B, a, b}, {a, b}, R, S) where R is the rule defined as follows:

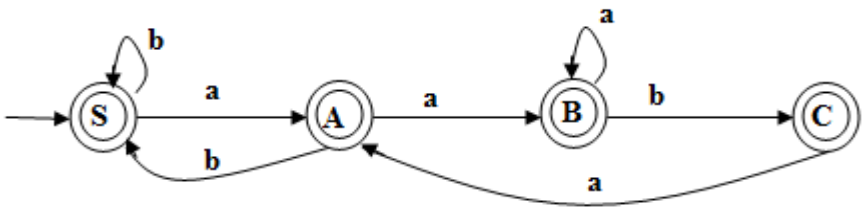
S → aA | bS | **ε**

A → aB | bS | **ε**

B→ aB | bS

Show a regular grammar for the language: L ={ w € {a, b }*: *w* does not contain the substring ***aabb*** }

DFSM which accepts L:



Regular grammar G = ({S, A, B, C, a, b}, {a, b}, R, S) where R is the rule defined as follows:

S → aA | bS | **ε**

A → aB | bS | **ε**

B→ aB | bC | **ε**

C→ aA | **ε**

Regular and Non regular Languages.

The language L = { a^*b^*} is a regular language because it can be modeled by a FSM.

The language L = { $a^n b^n$| n≥ 0 } is not a regular language because it is not possible, given some finite number of states, to count an arbitrary number of **a**'s and then compare that count to the number of **b**'s.

Regular languages are normally denoted by some regular expression.

How Many Regular Languages Are There?

There is a countably infinite number of regular languages. There cannot be more regular languages than there are DFSMs. There are at most a countably infinite number of regular languages. There is no one to one relationship between RLs and DFSM's, since there is an infinite number of machines that accept any given language.

Example: {a}, { aa} , { aaa}, { aaaa}. { aaaaa}, { aaaaaa } ….. .. .

Showing That a Language is Regular.

Theorem: Every finite language is regular.

Proof:

- If L is the empty language (no strings): L = { } Then the regular expression corresponding to L is = **Ø** , so L is regular.
- If any finite language L composed of the strings s_1, s_2, ... s_n for some positive integer n, then it is defined by the regular expression: s_1 U s_2 U……..U s_n ; So L is regular.
- Intersection of two infinite languages is finite.

Example: L_1 = { a^nb^n |n ≥ 0 } and L_2 = { b^na^n |n ≥ 0 }. Here both L_1 and L_2 are non-regular. But $L_1 \cap L_2$ = { **ε** } which is finite, is a regular language.

Prove that the following language L is regular or not?

i. L = { $a^i b^j$ |i, j ≥ } 0 and i + j = 5

L is Regular. A simple FSM with five states just counts the total number of characters.

ii. L = { $a^i b^j$ |i, j ≥ } 0 and i - j = 5

L is Not Regular. L consists of all strings of the form a*b* where the number of a's is five more than the number of b's. So we need infinite number of states to model this.

iii. L = { $a^i b^j$ | 0 ≤ i < j< 2000 }

L is Regular, because language is finite.

iv. {*w* ∈ {Y, N}* : *w* contains at least two Y's and at most two N's}.

L is Regular. *L* can be accepted by an FSM that keeps track of the count (up to 2) of Y's and N's.

Closure Properties of Regular Languages.

If certain languages are regular and language **L** is formed from them by certain operations such as union, concatenation, difference, star closure etc. then L is also regular. These properties are called closure properties. Closure property is a useful tool for building many complex automata.

The various closure properties of regular languages are:

1. Regular languages are closed under Union, concatenation and star (closure) operation.
2. The complement of a regular language is regular.
3. The intersection of two regular languages is regular.
4. The difference of two regular languages is regular.
5. The reversal of a regular language is regular.
6. A homomorphism of regular languages is regular.
7. The inverse homomorphism of regular language is regular.

1. Regular languages are closed under Union, concatenation and star (closure) operation.:

If L_1 and L_2 are regular languages, then prove that $L_1 U L_2$, $L_1.L_2$ and L_1^* is also regular language.

Proof: Since L_1 and L_2 are regular languages, they have regular expressions, say R_1 and R_2 such that $L_1 = L(R_1)$ and $L_2 = L(R_2)$.

By definition of regular expressions, we have

$R_1 U R_2$ is a regular expression denoting the language $L_1 U L_2$. (ie: $L(R_1) U L(R_2)$)

$R_1.R_2$ is a regular expression denoting the language $L_1.L_2$ (ie: $L(R_1)$ $L(R_2)$)

R_1^* is a regular expression denoting the language L_1^* (ie: $L(R_1)^*$)

So $L_1 U L_2$, $L_1.L_2$ and L_1^* is also regular language.

2. Closure under complementation:

The complement of a regular language is regular.

If L is regular language over alphabet Σ, then $\overline{L}$ is also regular language.

Proof:

Let $L = L(M_1)$ for some DFSM $\mathbf{M_1} = (Q, \Sigma, \delta, s, A)$. Then $\overline{L} = L(M_2)$, where M_2 is the DFSM = (K, Σ, δ, s, K - A). That is M_2 is exactly like M_1, but the accepting states of M_1 have become non-accepting states of M_2, and vice versa.

So the language which is rejected by $\mathbf{M_1}$ is accepted by M_2 and vice versa. Thus we have a machine $\mathbf{M_2}$ which accepts all those strings '***w***' in $\overline{L}$ that are rejected by machine M_1. So the complement of regular language L is $\overline{L}$ is also regular.

3. Closure under intersection**:**

The intersection of two regular languages is regular.

If L and M are regular languages then show that L∩M is also regular.

Proof::

Let L and M are regular languages. We know that complement of a regular language is regular.

So complement of L, ie: $\overline{L}$ and complement of M, ie: $\overline{M}$ is also regular language.

Also union of two regular languages is a regular language.

So union of $\overline{L}$ and $\overline{M}$, ie: $\overline{L} \cup \overline{M}$ is also regular.

But we know that complement of $\overline{L} \cup \overline{M}$ ie: = $\overline{\overline{L} \cup \overline{M}}$ is also regular.

According to De-morgan's law, $L \cap M = \overline{\overline{L} \cup \overline{M}}$ which is a regular language.

Therefore L∩M is also regular.

4. The difference of two regular languages is regular.

If L and M are regular languages, then so is L – M.

Proof: L and M are regular languages,

We know that complement of a regular language is again a regular language. So the complement of M , ie: $\overline{M}$ is regular.

Also the intersection of two regular languages is regular, ie: $L \cap \overline{M}$ is also regular.

According De-morgan's law $L - M = L \cap \overline{M}$ is regular language.

Therefore **L – M** is a regular language.

5. The reversal of a regular language is regular.

If L is regular language, so is L^R

Proof: Assume that L is defined by a regular expression E.

We have to show that there is another regular expression E^R such that $L(E^R) = (L(E))^R$

By definition of a regular expression, ε, φ, or **a** for some symbol a, is a regular expression.

If, $E = \varepsilon$, $E^R = \varepsilon$, $E = \varphi$, $E^R = \varphi$ and $E = a$, $E^R = a$ then E^R is same as E.

Also by definition of regular expression,

1. $E = E_1 + E_2$ then $E^R = E_1^R + E_2^R$ **ie:** the reversal of the union of two languages is obtained by computing the reversal of the two languages and taking the union of those languages.
2. $E = E_1E_2$. Then $E^R = E_2^R E_1^R$.
3. $E = E_1^*$. Then $E^R = (E_1^R)^*$ ie: string is in L(E) if and only if its reversal is in $L(E_1^R)^*$

Homomorphism (Letter substitution)

The term homomorphism means substitution of strings by some other symbols.

Example: string ***aabb*** can be written as 0011 under Homomorphism by replacing ***a*** with **0** and ***b*** with 1.

Let $\sum$ is the set of input alphabets and Γ be the set of substitution symbols, then $\sum^* \rightarrow \Gamma^*$ is homomorphism.

Let $w = a_1, a_2, a_3, \ldots\ldots\ldots\ldots\ldots\ldots a_n$

Then $h(w) = h(a_1)\, h(a_2)\, h(a_3)\ldots\ldots\ldots\ldots h(a_n)$

If L is a language belongs to set $\sum$, then the homomorphic image of L can be defined as:

$h(L) = \{ \mathbf{h(w) : w \in L} \}$

6. A homomorphism of a regular language is regular.

Proof: by counter example

Let L is a regular language, we have to prove that h(L) is also regular.

$\sum = \{ a, b\}$ and string w = abab

h(a) = 00

h(b) = 11

By definition $h(w) = h(a_1)\, h(a_2)\, h(a_3)\ldots\ldots\ldots\ldots h(a_n)$

h(w) = h(a) h(b) h(a) h(b) = 00110011

The homomorphism to a language is applied by applying homomorphism on each string of language. Let L = ab^*b (regular language) ie: **L** = { ab, abb, abbb, abbbb, ……}

h(L) = {0011, 001111, 00111111, 0011111111,…….}

h(L) = $\mathbf{00(11)^+}$; h(L) can be represented by **RE,** so it is a regular language.

7. The inverse homomorphism of regular language is regular.

Proof: Let $\sum^* \rightarrow \Gamma^*$ is homomorphism, where $\sum$ is the set of input alphabets and Γ be the set of substitution symbols, then h(L) be homomorphic language.

The inverse of homomorphic language can be represented as $h^{-1}(L)$

Let $h^{-1}(L) = \{ w : w \in L \}$

If L is regular then we know that h(L) is also regular. Since L is regular there exists some FSM M = (K, ∑, δ, s, A) which accepts L. Then h(L) must also accepted by M.

For complement of L ie: $\overline{L}$ is also regular which is accepted by FSM M' where Final states of M becomes non final states and all the non final states of M become the final states. Clearly $\overline{L}$ is accepted' by M. The inverse homomorphic language of $\overline{L}$ is h^{-1} (L) is also accepted by M'. So , h^{-1} (L) is also regular.

Showing That a Language, is Not Regular.

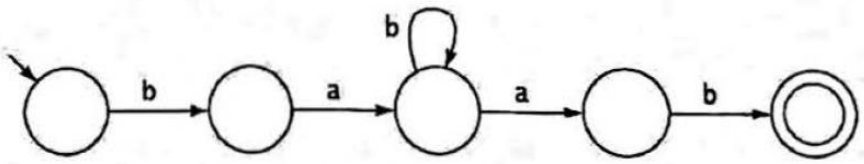
We can show that a language is regular by exhibiting ***a regular expression*** or an ***FSM*** or a finite list of the ***equivalence classes of ~ L*** or a ***regular grammar*** or by using the ***closure properties*** that we have proved hold for the regular languages. But how shall we show that a language is not regular. In other words, how can we show that none of those descriptions exists for it'?

We need a technique that does not rely on our cleverness.

What we can do is to make use of the following observation about the regular languages:

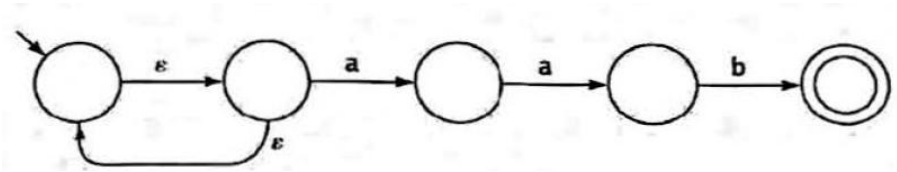
Every regular language ***L*** can be accepted by an FSM ***M*** with a finite number of states. If ***L*** is infinite, then there must be at least one loop in M. All sufficiently long strings in ***L*** must be characterized by one or more repeating patterns corresponding to the substrings that drive ***M*** through its loops.

Let us consider the following ***5*** state ***FSM***:



- It can accept infinite number of strings.
- The longest string that can be accepted by the FSM without going through any loops has length 4.
- The total number of states in FSM is 5 **ie:** |**k**| = 5.
- Take any longest string ***w***, such that length of longest string in L ≥ number of states in FSM
- If we take string w = ***babbab***, then |w| = 6 which is greater than the number of states in FSM. Here the second ***b*** in ***w*** drove the FSM through its loop, we call it as ***y***
- Suppose if we remove this y (pump y out), resulting in a new string ***babab*** which is also accepted by FSM.
- Also we can pump in **(**adding one or more**)** as many copies of ***b*** as we like, generating such strings as ba***bb***bab, ba***bbb***bab and so forth. FSM also accepts all of them.
- In the original string w = ***babbab,*** the third ***b*** also drove FSM through its loop. We could also pump it in or out and we get a similar result.

Let us consider one more FSM with 5 states:



This FSM accepts only one string, ***aab***. The only string that can drive *FSM* through its loop is **ε**

No matter how many times *FSM* goes through the loop, it cannot accept any longer strings. Therefore the length of pumping string y must be greater 0. It should not be empty.

- This property of FSMs and the languages that they can accept is the basis for a powerful tool for showing that a language is not regular.
- If a language contains even one long string that cannot be pumped in the fashion that we have just described, then it is not accepted by any FSM and so is not regular.

We formalize this idea, in Pumping Theorem.

The Pumping Theorem for Regular languages (Pumping Lemma for Regular Languages)

**********State and prove pumping theorem for regular languages

Theorem:

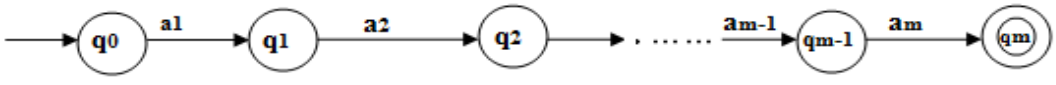
Let **L** be a regular language. Then there exists a constant '**k**' (number of states in FSM which depends on L) such that for every string '***w***' in L such that $|\mathbf{w}| \geq \mathbf{k}$, we can break ***w*** into three strings, w = xyz, such that:

$|y| > 0$ ie: $y \neq \varepsilon$

$|xy| \leq k$

For all $q \geq 0$, the string xy^qz is also in L

Proof: Suppose L = L(M) for some DFSM 'M' and L is regular language. Suppose 'M' has '***k***' number of states. Consider any string w = $a_1a_2a_3$.....a_m of length '***m***' where $\mathbf{m} \geq \mathbf{k}$ and each $\mathbf{a_i}$ is an input symbol. Since we have '***m***' input symbols, naturally we should have '***m+1***' states, in sequence q_0, q_1, q_2.....$\mathbf{q_m}$ where $\boldsymbol{q_0}$ is the start state and $\boldsymbol{q_m}$ is the final state.



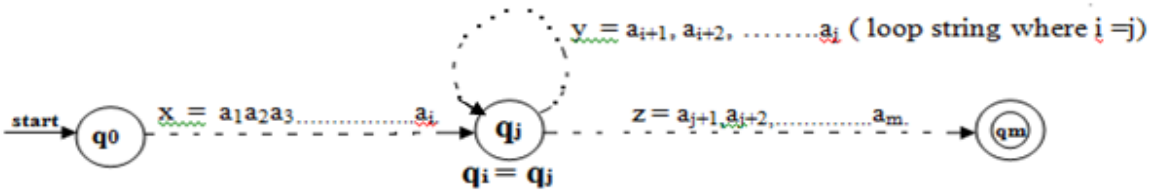
Since $|\mathbf{w}| \geq \mathbf{k}$, by the pigeonhole principle it is not possible to have ***distinct transitions***, since there are only **'k'** different states. So one of the state can have a loop. Thus we can find two different integers **i** and **j** with $0 \leq \mathbf{i} < \mathbf{j} \leq k$, such that $\mathbf{q_i} = \mathbf{q_j}$. Now we can break the string **w** = xyz as follows:

x = $a_1a_2a_3$.....a_i.

y = a_{i+1}, a_{i+2}, ...a_j (loop string where i = j)

z = a_{j+1}, a_{j+2},.....a_m.

The relationships among the strings and states are given in figure below:



'x' may be empty in the case that **i=** 0. Also '**z'** may be empty if j = k = m. However, ***y*** cannot be empty, since '**i'** is strictly less than '**j**'.

Thus for any integer $q \geq 0$, xy^qz is also accepted by DFSM 'M'; that is for a language L to be a regular, xy^qz is also in L for all $q \geq 0$.

Show that $L = \{ a^nb^n \mid n \geq 0 \}$ is not regular.

Assume that given language **L** is regular language and there exist some '**k**' number of states, such that any string **w**, where $|\mathbf{w}| \geq \mathbf{k}$ must satisfy the conditions of the theorem.

Let $\mathbf{w} = a^k b^k$

Since $|w| = k + k = 2k \geq k$, we can split '**w**' into xyz such that $|\mathbf{xy}| \leq \mathbf{k}$ and $|\mathbf{y}| \geq 1$ as

w = xyz

Since |xy| <= **k**, ***y*** must occur within the first ***k*** characters and *so* **y** = a^p for some *p*. Also $y \neq \varepsilon$, ***p*** must be greater than 0.

x = a^{k-p} **y** = a^p **z** = b^k

According to pumping lemma, language to be regular, $xy^qz \in L$ for all $q \geq 0$.

Let q =**2** and the resulting string $\boldsymbol{w} = a^{k-p}(a^p)^2 b^k$ where $p \geq 1$ $= a^{k+p} b^k$ must be in L.

But it is not since it has more ***a***'s than more ***b***'s.

Thus there exists at least one long string ***w*** in ***L*** that fails to satisfy the conditions of the Pumping Theorem. So $a^n b^n \mid n \geq 0$ is not regular.

NOTE: Show that $L = \{w \in (a, b)^* \mid n_a(w) = n_{b(w)}\}$ is not regular.

We can prove that L is not regular by taking string $w = a^n b^n \mid n>=0$. Refer previous problem.

Show that L = {w € **{), (}***: the parentheses are balanced} are not regular

Assume that given language **L** is regular language and there exist some '**k**' number of states, such that any string **w**, where |**w**| ≥ **k** must satisfy the conditions of the theorem.

Let **w** = $(^k)^k$

Since |w| = k + k = 2k ≥ k, we can split '**w**' into xyz such that |**xy**| ≤ **k** and |**y** | ≥ 1 as

w = xyz

Since |xy| <= **k**, ***y*** must occur within the first ***k*** characters and *so* **y** = $(^p$ for some *p*. Also *y* ≠ε, ***p*** must be greater than 0.

x = $(^{k-p}$ **y** = $(^p$ **z** = $)^k$

According to pumping lemma, language to be regular, xy^qz € L for all q ≥ 0.

Let q =**2** and the resulting string $\boldsymbol{w} = (^{k-p}\ (^{2p}\ )^k$ where $p \geq 1$ $= (^{k+p})^k$ must be in L.

But it is not since it has more ***(***'s than more ***)***'s.

Thus there exists at least one long string ***w*** in ***L*** that fails to satisfy the conditions of the Pumping Theorem. So {w € **{), (}***: the balanced parentheses}is not regular.

Show that $L = \{ ww^R \mid w \in (a, b)^* \}$ is not regular.

Assume that given language **L** is regular language and there exist some '**k**' number of states, such that any string **w**, where |**w**| ≥ **k** must satisfy the conditions of the theorem.

Let us consider one string defined in L; **w** = $a^k b^k b^k a^k$

Since |w| = k + k+ k + k = 4k ≥ k, we can split '**w**' into xyz such that |**xy**| ≤ **k** and |**y** | ≥ 1 as

w = xyz

Since |xy| <= **k**, ***y*** must occur within the first ***k*** characters and *so* **y** = a^p for some *p*. Also *y* ≠ε,

p must be greater than 0.

x = a^{k-p} **y** = a^{p} **z** = $b^{k}\ b^{k}\ a^{k}$

According to pumping lemma, language to be regular, $xy^{q}z \in L$ for all $q \geq 0$.

Let q =**2** and the resulting string ***w*** = $a^{k-p}\ (a^{p})^{2}\ b^{k}\ b^{k}\ a^{k}$ where $p \geq 1$ = $a^{k+p}\ b^{k}\ b^{k}\ a^{k}$ must be in L

But it is not since the first half of the string has more ***a***'s than the second half does. So it is not in L.

Thus there exists at least one long string ***w*** in ***L*** that fails to satisfy the conditions of the Pumping Theorem. So L = { ww^{R} | w € $(a, b)^{*}$ is not regular.}

Show that L = { $a^{n}b^{m}$ | n > m } is not regular.

Assume that given language **L** is regular language and there exist some '**k**' number of states, such that any string **w**, where |**w**| ≥ **k** must satisfy the conditions of the theorem.

Let **w** = $a^{k+1}\ b^{k}$ (Language contains strings of more **a**'s than **b**'s)

Since |w| = k +1+ k = 2k +1 ≥ k, we can split '**w**' into xyz such that |**xy**| ≤ **k** and |**y** | ≥ 1 as

w = xyz

Since |xy| <= **k**, ***y*** must occur within the first ***k*** characters and *so* **y** = a^{p} for some *p*. Also $y \neq \varepsilon$, ***p*** must be greater than 0.

x = a^{k-p} **y** = a^{p} **z** = ab^{k}

According to pumping lemma, language to be regular, $xy^{q}z \in L$ for all $q \geq 0$.

Let q = **0** and the resulting string ***w*** = $a^{k-p}\ (a^{p})^{0}\ ab^{k}$ where $p \geq 1$ = $a^{k+1-p}\ b^{k}$ must be in L

But it is not since p > 0 and k + 1 - p <= k, so the resulting string no longer has more ***a***'s than ***b***'s and so is not in L.

Thus there exists at least one long string **w** in **L** that fails to satisfy the conditions of the Pumping Theorem. So L = { $a^{n}b^{m}$ | n > m } is not regular.

NOTE: Show that L= $\{a^{i}\ b^{j} \mid i \neq j\}$ is not regular**.**

ie: **i ≠ j** means i > j or i < j; so we can take string 'w' = $a^{k+1}b^{k}$ or w= $a^{k}b^{k+1}$.

Solution is similar to the previous problems

Show that L = {0^m | m is prime} is not regular.

Here the language **L** contains stings of ***0***'s of length prime.

L = {0, 00, 000, 00000, 0000000,……}

Assume that given language **L** is regular language and there exist some '**k**' number of states, such that any string **w**, where |**w**| ≥ **k** must satisfy the conditions of the theorem.

Let **w** = 0^k (***k*** is prime)

Since |w| = k ≥ k, we can split '**w**' into xyz such that |**xy**| ≤ **k** and |**y** | ≥ 1 as

w = xyz

Since |xy| <= **k**, ***y*** must occur within the ***k*** characters and *so* **y** = 0^p for some *p*. Also *y* ≠ε,

p must be greater than 0.

x = 0^i **y** = 0^p **z** = 0^{k-i-p} where i + p <= k and |p| > 0

According to pumping lemma, language to be regular, $xy^{\mathbf{q}}z$ € L for all q ≥ 0.

$0^i (0^p)^q 0^{k-i-p} = 0^{k+p(q-1)}$ € L

ie: k + p(q – 1) is also prime for all q ≥ 0

If we choose q = k + 1 then k + p(q – 1) becomes k + kp = k (p + 1) is also prime

We know that p > 0; suppose p = 1 then 2k is also prime, but it is not true, which is a contradiction to our assumption. So the language L = {0^m | m is prime} is not regular.

Show that L = {$a^{n!}$ | n ≥ 0} is not regular.

Assume that given language **L** is regular language and there exist some '**k**' number of states, such that any string **w**, where |**w**| ≥ **k** must satisfy the conditions of the theorem.

Let w = $a^{k!}$ (the length of string is k!)

Since |w| = k! ≥ k, we can split '**w**' into xyz such that |**xy**| ≤ **k** and |**y** | ≥ 1 as

w = xyz

Since |xy| <= **k**, ***y*** must occur within the ***k*** characters and *so* **y** = a^p for some *p*. Also *y* ≠ε,

p must be greater than 0.

x = a^i **y** = a^p **z** = $a^{k!-i-p}$ where i + p <= k and |p| > 0

According to pumping lemma, language to be regular, $xy^qz \in L$ for all $q \geq 0$.

$a^i (a^p)^q a^{k! - i - p} = a^{k! + p(q-1)} \in L$ for all $q \geq 0$

If we choose $q = 0$ then $a^{k! - p} \in L$ for all $p \geq 1$

If $p = 1$, then $k! - 1 > k!$

$a^{k! - 1}$ does not belongs to L, which is a contradiction to our assumption. So the language $L = \{a^{n!} \mid n \geq 0\}$ is not regular.

Show that $L = \{ ww \mid w \in (a, b)^* \}$ is not regular.

Assume that given language **L** is regular language and there exist some '**k**' number of states, such that any string **w**, where $|w| \geq k$ must satisfy the conditions of the theorem.

Let us consider one string defined in L; $\mathbf{w} = a^k b^k a^k b^k$

Since $|w| = k + k + k + k = 4k \geq k$, we can split '**w**' into xyz such that $|xy| \leq k$ and $|y| \geq 1$ as

$w = xyz$

Since $|xy| <= k$, ***y*** must occur within the first ***k*** characters and *so* $\mathbf{y} = a^p$ for some *p*. Also $y \neq \varepsilon$, ***p*** must be greater than 0.

$\mathbf{x} = a^{k-p}$ $\mathbf{y} = a^p$ $\mathbf{z} = b^k a^k b^k$

According to pumping lemma, language to be regular, $xy^qz \in L$ for all $q \geq 0$.

Let $q = 2$ and the resulting string $\boldsymbol{w} = a^{k-p} (a^p)^2 b^k a^k b^k$ where $p \geq 1$ $= a^{k+p} b^k a^k b^k$ must be in L

But it is not since the first half of the string has more ***a***'s than the second half does. So it is not in L. Thus there exists at least one long string ***w*** in ***L*** that fails to satisfy the conditions of the Pumping Theorem. So $L = \{ ww \mid w \in (a, b)^* \}$ is not regular.}

Show that $L = \{w \in (a, b)^* \mid n_a(w) < n_{b(w)} \}$ is not regular

Assume that given language **L** is regular language and there exist some '**k**' number of states, such that any string **w**, where $|w| \geq k$ must satisfy the conditions of the theorem.

Let $\mathbf{w} = a^k b^{k+1}$ (Language contains strings of more **b**'s than **a**'s)

Since $|w| = k + k+1 = 2k + 1 \geq k$, we can split '**w**' into xyz such that $|xy| \leq k$ and $|y| \geq 1$ as

$w = xyz$

Since $|xy| <= k$, ***y*** must occur within the first ***k*** characters and *so* $\mathbf{y} = a^p$ for some *p*. Also $y \neq \varepsilon$,

p must be greater than 0.

x = a^{k-p} **y** = a^{p} **z** = b^{k+1}

According to pumping lemma, language to be regular, $xy^{q}z \in L$ for all $q \geq 0$.

Let q = **2** and the resulting string ***w*** = $a^{k-p}(a^{p})^{2} b^{k+1}$ where $p \geq$ **1; w** = $a^{k+1+p} b^{k+1}$ must be in L

But it is not since p > 0 and k + 1 + p > k+1, so the resulting string no longer has more ***b***'s than ***a***'s and so is not in L.

Thus there exists at least one long string **w** in **L** that fails to satisfy the conditions of the Pumping Theorem. So L= $\{w \in (a, b)^{*} \mid n_a(w) < n_{b(w)}\}$ is not regular.

Show that L= $\{0^{n} \mid n$ is a perfect square$\}$ is not regular. **OR**

Show that L= $\{0^{n} \mid n = k^{2}$ where $k \geq 1\}$ is not regular

Here the language **L** contains stings of ***0***'s of perfect square length.

ie: L = {0, 0000, 000000000,…… }

Assume that given language **L** is regular language and there exist some '**k**' number of states, such that any string **w**, where $|w| \geq$ **k** must satisfy the conditions of the theorem.

Let **w** = $0^{k^{2}}$

Since $|w| = k^{2} \geq k$, we can split '**w**' into xyz such that $|xy| \leq$ **k** and $|y| \geq 1$ as

w = xyz

Since $|xy| \leq$ **k**, ***y*** must occur within the first ***k*** characters and *so* **y** = a^{p} for some *p*. Also $y \neq \varepsilon$, ***p*** must be greater than 0.

x = 0^{i} y = 0^{p} z = $0^{k^{2}-i-p}$

According to pumping lemma, language to be regular, $xy^{q}z \in L$ for all $q \geq 0$.

Let q = **2** and the resulting string ***w*** = $0^{i}(0^{p})^{2} 0^{k^{2}-i-p}$ where $p \geq$ **1; w** = $0^{k^{2}+p}$ must be in L.

But it is not since p > 0 and when p= 1; $k^{2} < k^{2}+1 < (k+1)^{2}$, so the resulting string no longer appears in L

Thus there exists at least one long string **w** in **L** that fails to satisfy the conditions of the Pumping Theorem. So L= $\{0^{n} \mid$ **n** is a perfect square$\}$ is not regular.