St. Joseph's College of Engineering & Technology Palai

Department of Computer Science & Engineering

S4 CS

CS010 406 Theory of Computation

Module 2

# Theory of Computation - Module 2

## Syllabus

Introduction to Automata Theory - Definition of Automation - Finite automata - Language acceptibility by finite automata - Deterministic and non deterministic finite automation -

Regular expressions - Finite automation with $\epsilon$ transitions - Conversion of NFA to DFA - Minimisation of DFA - DFA to Regular expressions conversion -

Pumping lemma for regular languages -

Applications of finite automata - NFA with o/p (moore / mealy)

## Contents

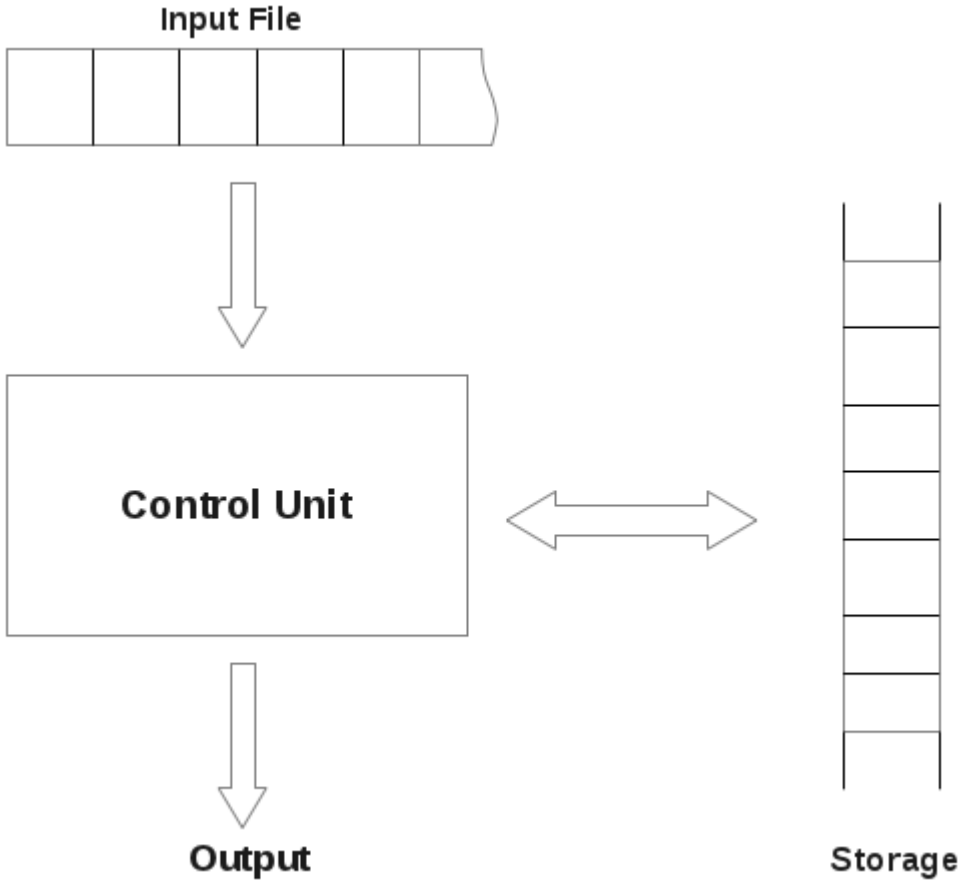
# Part I. Introduction to Automata Theory

## 1 Automation

An automation is defined as a system that performs certain functions without human intervention. It accepts some input as row materials and converts it to a product under the guidance of control signals. An automation can be designed to perform a variety of tasks related to various domain of human life.

In terms of computer science, an automation is an abstract model of a digital computer. An automation has some features.

following figure shows the representation of an automation.




### Input

An automation has a mechanism for reading input. It is assumed that input is in a file. The input file is divided into cells. Each cell can hold one symbol. Input mechanism can read input file from left to right.

### Output

The automation can produce output of some form.

### Storage

An automation can have a temporary storage device. the storage device can consist of unlimited number of cells. The automation can read and change the contents of the storage cells.

### Control Unit

Automation has a control unit. At a given time, control unit is in some internal state.

## 2 Automata Theory

automata theory is the study of abstract computing devices or machines.

The study of automata is important because ,

1. Automata theory plays an important role when we make software for designing and checking the behaviour of a digital circuit.

2. The lexical analysis of a compiler breaks a program into logical units, such as variables, keywords and punctuation using this mechanism.

3. Automata theory works behind software for scanning large bodies of text, such as web pages to find occurrence of words, phrases etc..

4. Automata theory is a key to software for verifying systems of all types (software testing).

5. Automata theory is the most useful concept of software for natural language processing.

### Definition of an Automation

an automation is defined as a program where energy, material and information are transformed, transmitted and used for performing some functions without direct human intervention.

Example: Automatic machine tools,

Automatic packing machines,

Automatic photo copying machines.

Different classes of automata are ,

Finite automata, and

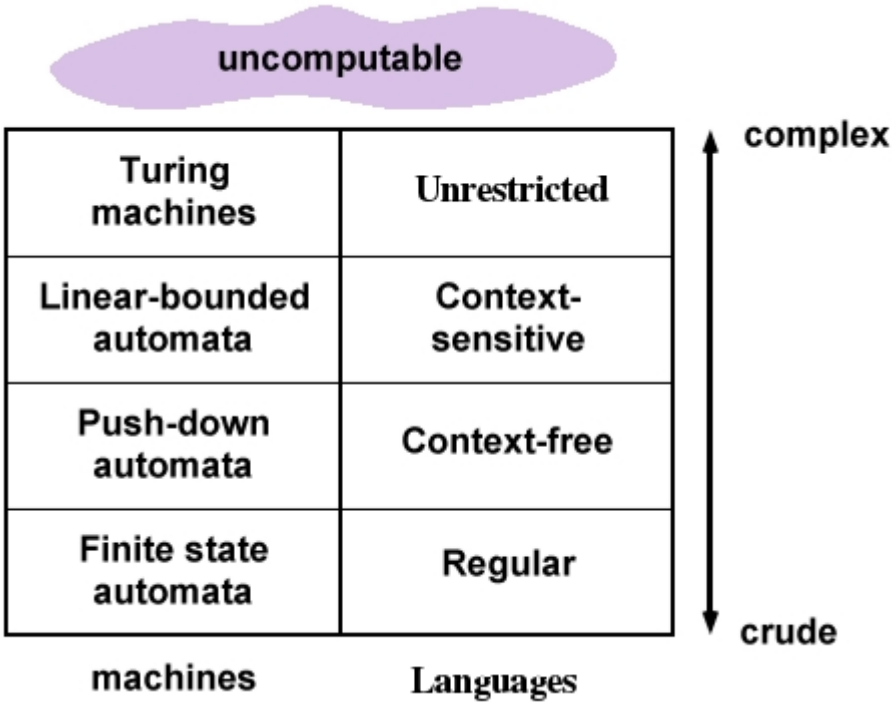
Pushdown automata.

We will learn both of them in detail.

# Part II. Finite Automata

Different classes of automata are there. examples: finite automata, pushdown automata. Of them finite automata is the simplest one.

In Module I, during the discussion of chomsky classification, it was seen that type-3 or regular languages are recognised using finite automata.




### Definition

A finite automation, M is a 5 tuple structure,

$$M\,(\,Q,\,\sum,\,q_0,\,\delta,\,F\,)$$

where M is the finite automation,

$Q$ is a finite set of states,

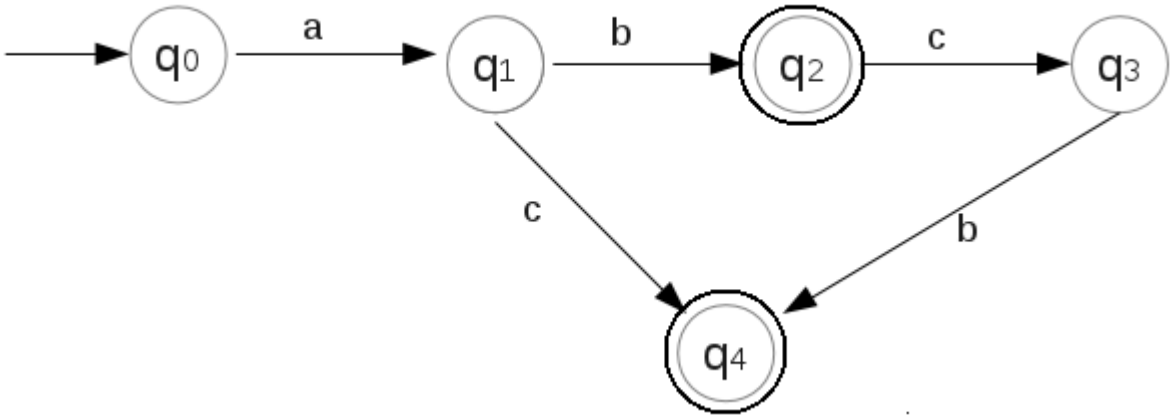
$\sum$ is a set of input symbols,

$q_0$ is the start state,

$\delta$ is the set of transition functions,

$F$ is the set of final states.

Consider the following example for a finite automation, M.



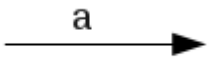

In the above finite automation, M,

1. $Q = \{q_0, q_1, q_2, q_3, q_4\}$

Q is the set of states in the finite automata. In the above finite automata, $q_0, q_1, q_2, q_3, q_4$ are the states. The states are shown in circles.


2. $\sum = a, b, c$

$\sum$ is the set of input symbols in the finite automata. In the above, a, b, c are the input symbols. The arrowsare labelled with input symbols.




3. $q_0 = \{q_0\}$

$q_0$ is the start state. In the above, $q_0$ is the start state. For our study, a finite automata contains only one start state. A state with arrow from free space (not coming from any other state) is designated as start state.


4. $\delta$ describes the operation of finite automation. $\delta$ is the transition function.

For example,

$$\delta(q_0, a) = q_1$$

This means, given the current state $q_0$ and the input symbol, a, the finite automation moves (transits) to state $q_1$.

Similarly,

$$\delta(q_1, b) = q_2$$

$$\delta(q_2, c) = q_3$$

$$\delta(q_1, c) = q_4$$

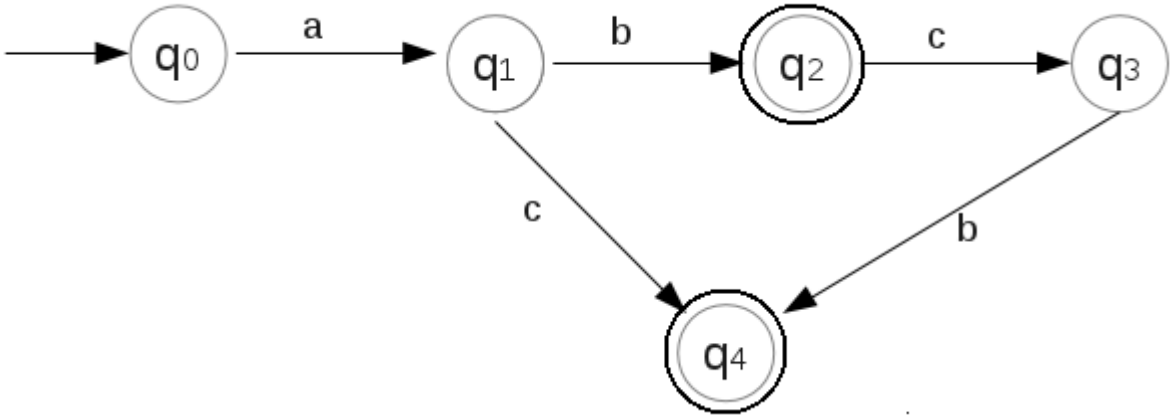
$$\delta(q_3, b) = q_4$$

5. $F = \{q_2, q_4\}$

F is the set of final states or accepting states. In the above finite automation, $q_2$and $q_4$ are the final states. They are shown enclosed in double circles.

## Transition Diagrams and Transition Tables

Consider the following finite automation, M.




The above representation of finite automation is called a transition diagram.

A finite automata can also be represented as a transition table. This is shown below:

| | Input Symbol | | |
|---|---|---|---|
| Current State | a | b | c |
| $\longrightarrow q_0$ | $q_1$ | $\phi$ | $\phi$ |
| $q_1$ | $\phi$ | $q_2$ | $q_4$ |
| $*q_2$ | $\phi$ | $\phi$ | $q_3$ |
| $q_3$ | $\phi$ | $q_4$ | $\phi$ |
| $*q_4$ | $\phi$ | $\phi$ | $\phi$ |

$\longrightarrow q_0$ denotes that $q_0$ is the start state.

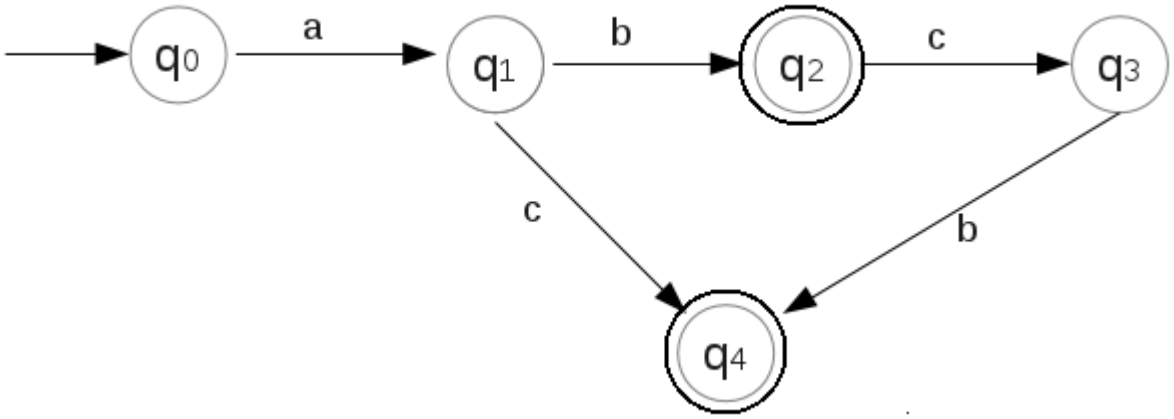
$*$ before $q_2$ and $q_4$ indicates that $q_2$ and $q_4$ are the final states.

# 3 Language Acceptibility by Finite Automata

## String Processing by Finite Automation

**Example 1:**

Consider the following finite automation, M.




Check whether the string abcb is accepted by the above finite automation.

In the above, $q_0$ is the start state. the string abcb has the first symbol, a.

So,

$\delta(q_0, \underline{a}bcb) = q_1$

$\delta(q_1, a\underline{b}cb) = q_2$

$\delta(q_2, ab\underline{c}b) = q_3$

$\delta(q_3, abc\underline{b}) = q_4$



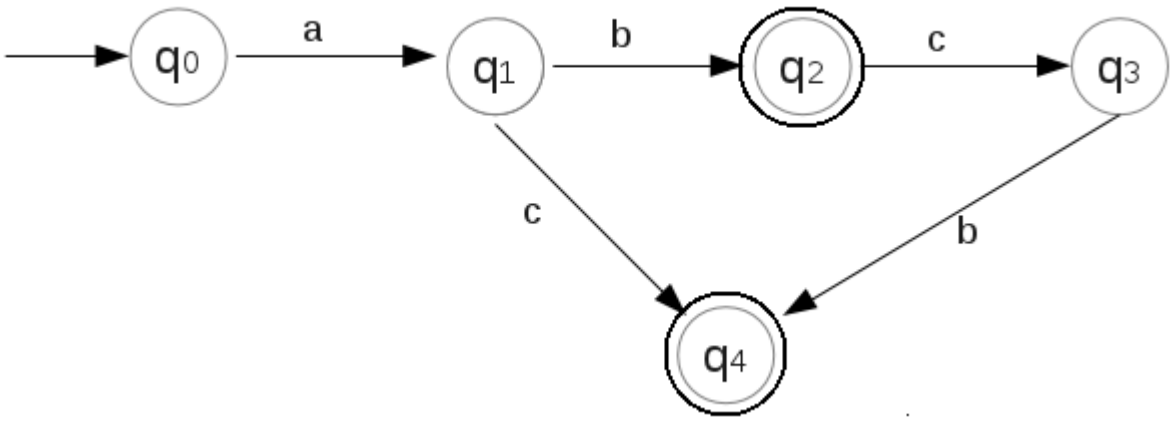
Thus,

$q_0 \overset{a}{\Longrightarrow} q_1 \overset{b}{\Longrightarrow} q_2 \overset{c}{\Longrightarrow} q_3 \overset{b}{\Longrightarrow} q_4$

Thus, after processing the string abcb, we reached the state, $q_4$. Here $q_4$ is a final state. So the string abcb is accepted by the finite automation.

**Example 2:**

Consider the following finite automation, M




Check whether the string abc is accepted by M.

On processing the string, beginning from the start state, $q_0$,

$\delta(q_0, \underline{a}bc) = q_1$

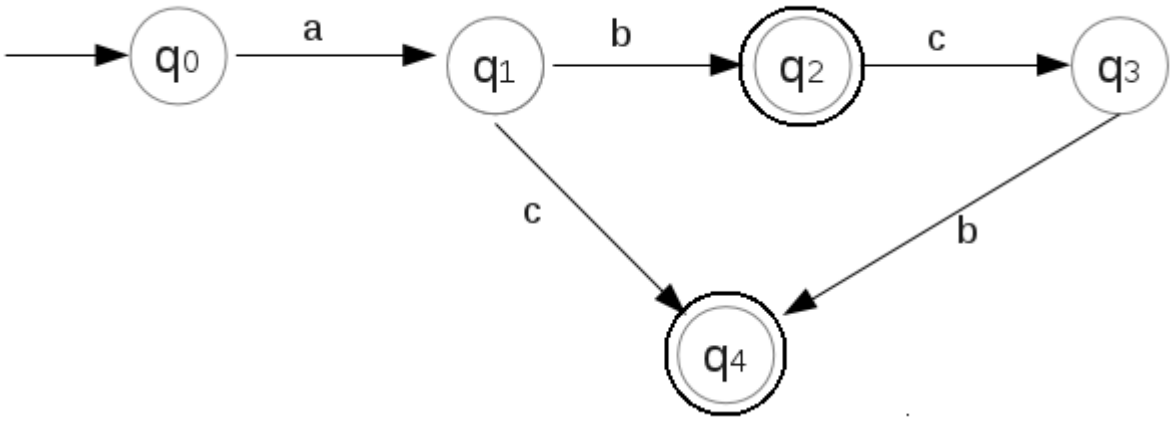
$\delta(q_1, a\underline{b}c) = q_2$

$\delta(q_2, ab\underline{c}) = q_3$

Here M halts at $q_3$. But $q_3$ is not a final state. Hence the string abc is rejected by the above finite automation, M.

**Example 3:**

Conisder the finite automation, M,




Check whether the string abca is accepted by the finite automation, M.

Beginning from the start state, $q_0$,

$\delta(q_0, \underline{a}bca) = q_1$

$\delta(q_1, a\underline{b}ca) = q_2$

$\delta(q_2, ab\underline{c}a) = q_3$

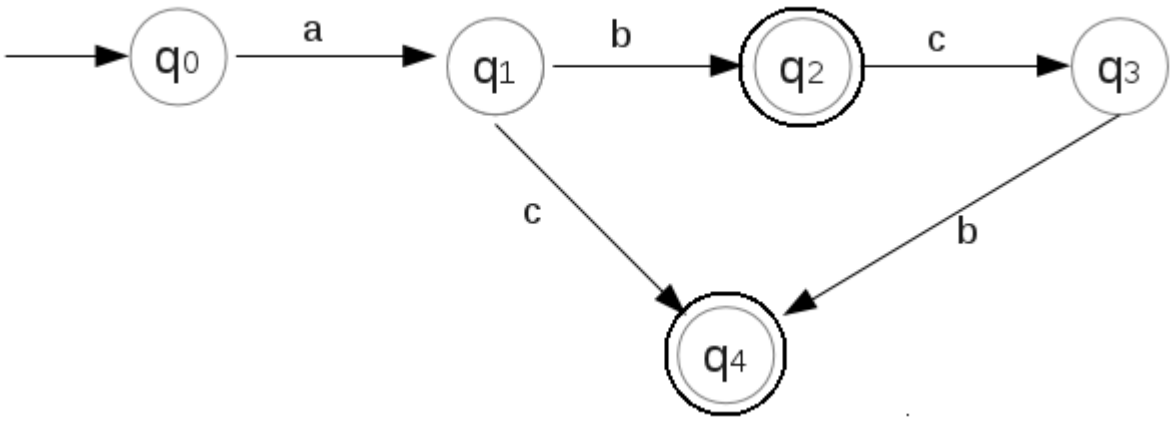
$\delta(q_3, abc\underline{a}) =$

Here for current state, $q_3$, and for input symbol, a, there is no transition. This means M is unable to process the string abca completely. So the string abca is rejected by M.

## Language of Finite Automation

In the above, finite automation is used to check the validity of a string. A language is a finite set of strings. So we can use finite automation to check whether a string belongs to a language.

Consider the following finite automation,




Let L be the language corresponding to the above finite automation, M. Then a string belongs to the language L only if it is accepted by the finite automation, M.

We can say that,

The string abcb belongs to the language, L.

The string abc does not belong to the language, L.

The string abca does not belong to the language, L.

The string ab belongs to the language, L.

The string ac belongs to the language, L.

The string abcbc does not belong to the language , L.

## 4 Types of Finite Automata

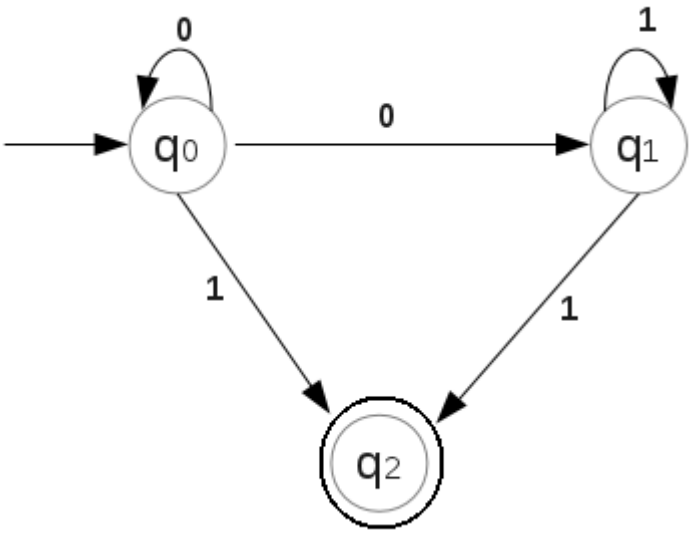
Finite automation is of different types. They are,

Non-deterministic Finite Automata (NFA), and

Deterministic Finite Automata (DFA).

### 4.1 Non- Deterministic Finite Automata (NFA)

Consider the following finite automation, M,



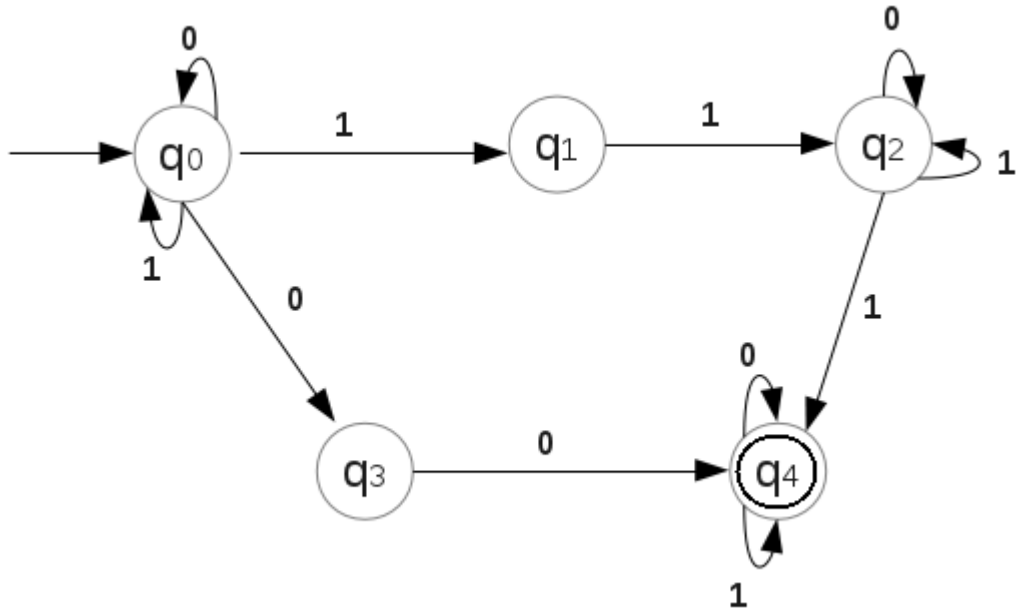

In the above, from state $q_0$ with input symbol 0, there are two possible next states. The next state can be either $q_0$ or $q_1$.

Also from state $q_1$, on input symbol, 1, there are two possible next states. The next state can be either $q_1$or $q_2$.

Thus some moves of the finite automation cannot be detrmined uniquely by the input symbol and the current state. Such finite automation are called Non-deterministic Finite automata (NFA or NDFA).

**Example 1:**

Consider the following NFA,




Check whether the string 0100 is accepted by the above NFA.

Beginning from the start symbol, $q_0$,

$\delta(q_0, \underline{0}100) = q_0$

$\delta(q_0, 0\underline{1}00) = q_0$

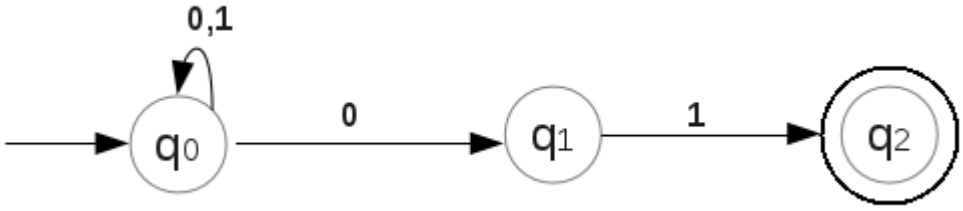
$\delta(q_0, 01\underline{0}0) = q_3$

$\delta(q_3, 010\underline{0}) = q_4$

$q_4$ is a final state. So the string 0100 is accepted by the above finite automation.

**Example 2:**

Consider the following NFA,




Check whether the string 01101 is accepted by the above NFA.

$\delta(q_0, \underline{0}1101) = q_0$

$\delta(q_0, 0\underline{1}101) = q_0$

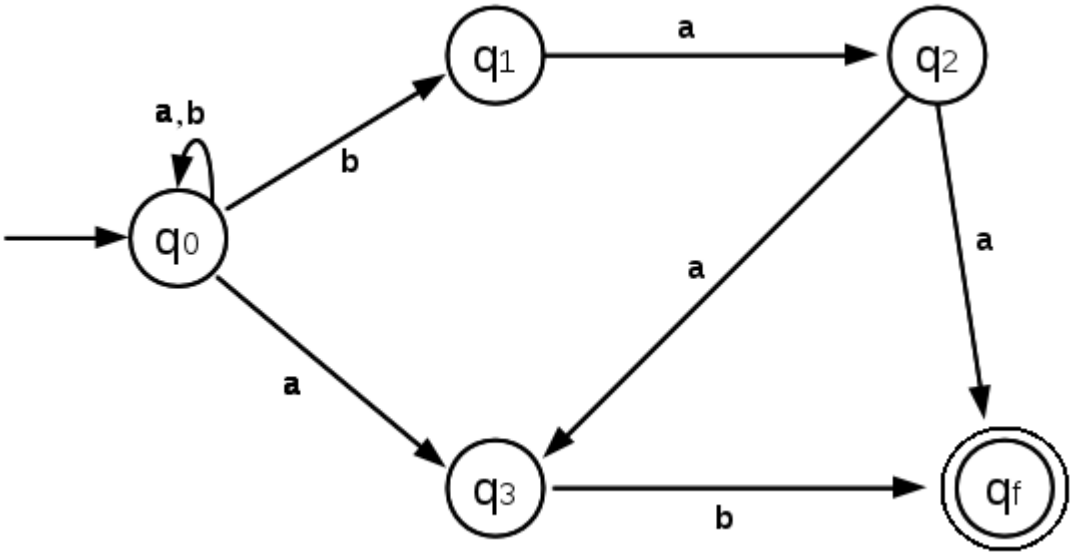
$\delta(q_0, 01\underline{1}01) = q_0$

$\delta(q_0, 011\underline{0}1) = q_1$

$\delta(q_1, 0110\underline{1}) = q_2$

$q_2$ is a final state. So the string 01101 is accepted by the above NFA.

**Example 3:**




Check whether the string abbaa is accepted by the above NFA.

$\delta(q_0, \underline{a}bbaa) = q_0$

$\delta(q_0, a\underline{b}baa) = q_0$

$\delta(q_0, ab\underline{b}aa) = q_1$

$\delta(q_1, abb\underline{a}a) = q_2$

$\delta(q_1, abba\underline{a}) = q_f$

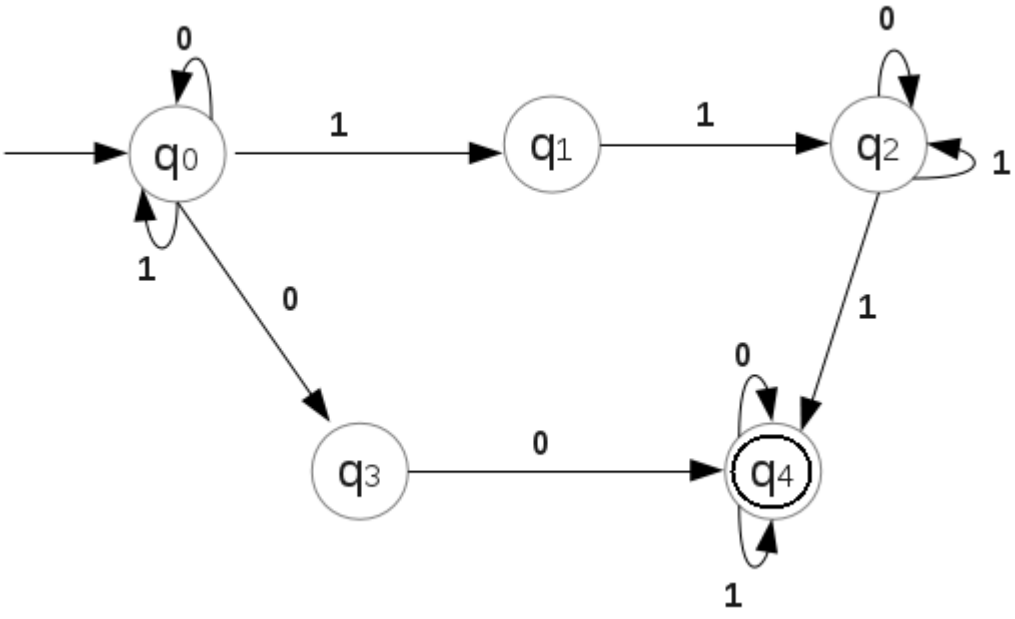
$q_f$ is a final state. So the string abbaa is accepted by the above NFA.

### 4.1.1 NFA with Epsilon Transitions

An NFA can contain certain transitions on input symbol, $\varepsilon$ (epsilon).

$\varepsilon$ means 'null'.

For example, consider the following NFA,




Check whether the string 100110 is accepted by the above NFA.

$\delta(q_0, \underline{1}00110) = q_0$

$\delta(q_0, 1\underline{0}0110) = q_0$

$\delta(q_0, 10\underline{0}110) = q_0$

$\delta(q_0, 100\underline{1}10) = q_1$

$\delta(q_1, 1001\underline{1}0) = q_2$

$\delta(q_2, 10011\underline{0}) = q_2$

$\delta(q_2, \varepsilon) = q_4$

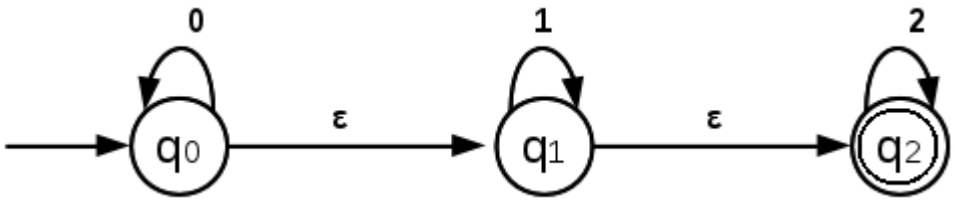
$q_4$ is a final state. So the string 100110 is accepted by the above NFA.

Here note that, we made an $\varepsilon$ move to reach the final state, $q_4$.

Thus, $\varepsilon$ transition means a transition without scanning the symbol in the input string.

**Example 2:**

Consider the following NFA,




Check whether the string 0012 is accepted by the above NFA.

Beginning with the start state, $q_0$,

$\delta(q_0, \underline{0}012) = q_0$

$\delta(q_0, 0\underline{0}12) = q_0$

$\delta(q_0, \varepsilon) = q_1$

$\delta(q_1, 00\underline{1}2) = q_1$

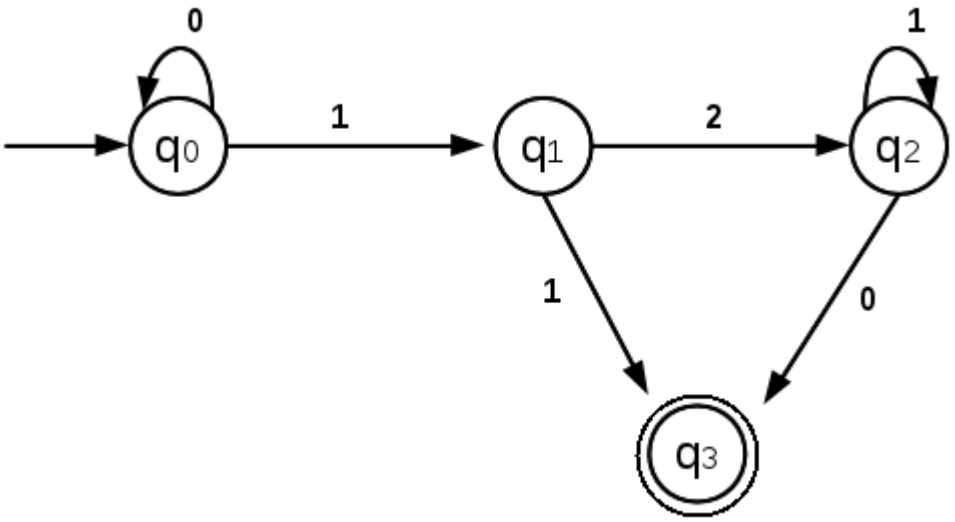
$\delta(q_1, \varepsilon) = q_2$

$\delta(q_2, 001\underline{2}) = q_2$

Here $q_2$ is a final state. So the string 0012 is accepted by the NFA.

## 4.2 Deterministic Finite Automata

Consider the following finite automation,




In the above, input symbols are 0, 1 and 2. From every state, there is only one transition for an input symbol.

From state, $q_0$, for symbol, 0, there is only one transition, that is to state $q_0$ itself.

From state, $q_0$, for symbol, 1, there is only one transition, that is to state $q_1$.

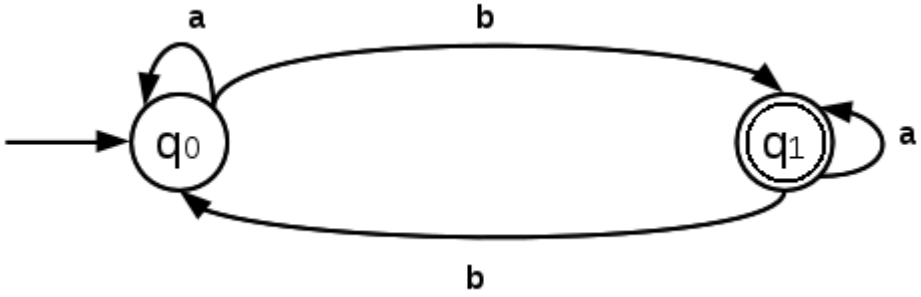
From state, $q_1$, for symbol, 1, there is only one transition, that is to state $q_3$, and so on.

Thus all the moves of the above finite automation can be determined uniquely by the current state and input symbol. Such a finite automation is called Deterministic finite Automation (DFA).

Thus in a DFA, at every step, next move is uniquely determined. No $\varepsilon$ transitions are present in a DFA.

**Example 1:**

The following shows a DFA,




Above transition diagram can be represented using the transition table as follows:

| | Input | Symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow q_0$ | $q_0$ | $q_1$ |
| $*q_1$ | $q_1$ | $q_0$ |

Check whether the string aaba is accepted by the above DFA.

Beginning from the start state, $q_0$,

$$\delta(q_0, \underline{a}aba) = q_0$$

$$\delta(q_0, a\underline{a}ba) = q_0$$

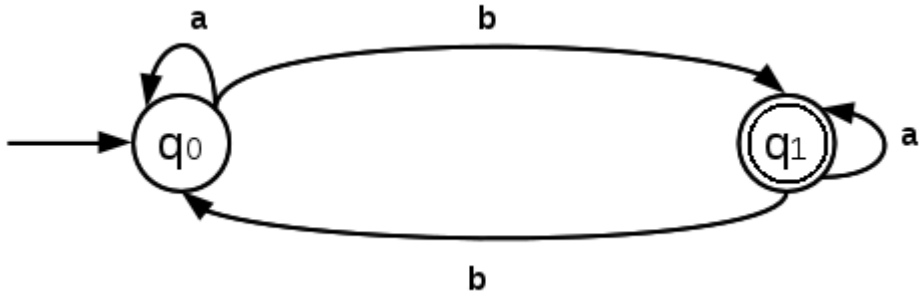
$$\delta(q_0, aa\underline{b}a) = q_1$$

$$\delta(q_1, aab\underline{a}) = q_1$$

After processing all the characters in the input string, final state $q_1$is reached. So the string aaba is accepted by the above finite automation.

**Example 2:**

Consider the following DFA,




Check whether the string aabba is accepted by the above DFA.

$\delta(q_0, \underline{a}abba) = q_0$

$\delta(q_0, a\underline{a}bba) = q_0$

$\delta(q_0, aa\underline{b}ba) = q_1$

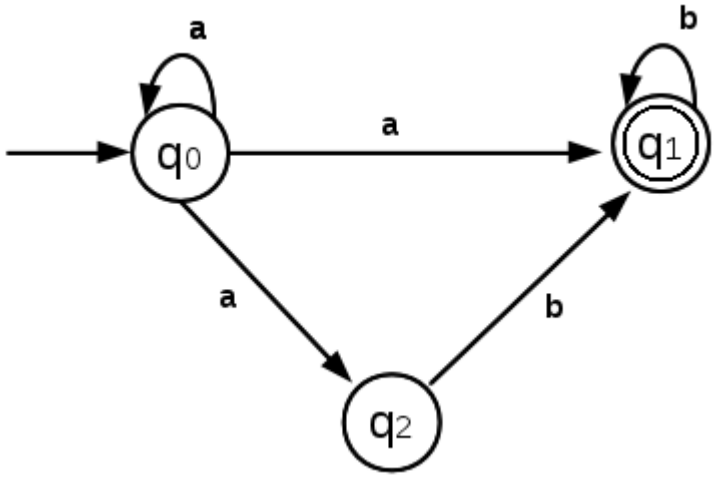
$\delta(q_1, aab\underline{b}a) = q_0$

$\delta(q_0, aabb\underline{a}) = q_0$

After processing all the symbols in the input string, aabba, the state $q_0$ is reached. But $q_0$ is not a final state. So the DFA rejects the string aabba.

# Part III. NFA to DFA Conversion

Consider the following NFA,




In the above NFA, from state $q_0$, there are 3 possible moves for input symbol, a. ie.. to $q_0, q_1, q_2$. When we simulate this NFA using a program, the move from $q_0$ for symbol a is non-deterministic. That means, which transition is to be made next cannot be determined in one move.

But in a DFA, from a state, for an input symbol, only one transition exists. So it is very easy to simulate a DFA using a program. So we need to convert an NFA to an equivalent DFA.

For every NFA, there exists an equivalent DFA.

## 5 Epsilon Closure

For converting an NFA to DFA, we first need to learn how to find the epsilon closure of a state.

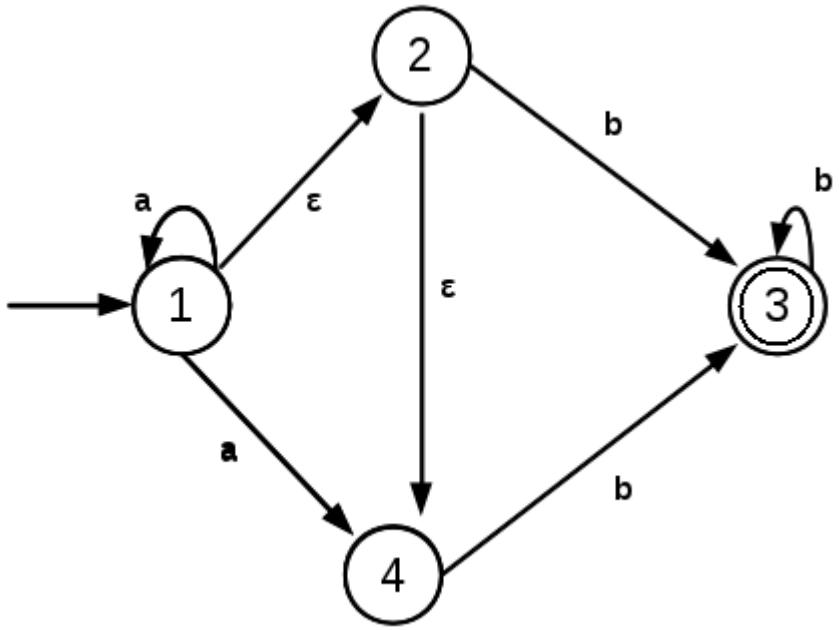
Epsilon closure ($\varepsilon$- closure) of a state $q$ is the set that contains the state $q$ itself and all other states that can be reached from state $q$ by following $\varepsilon$ transitions.

We use the term, ECLOSE( $q_0$) to denote the Epsilon closure of state $q_0$.

For example,

**Example 1:**

Consider the following NFA,




Find the epsilon closure of all states.

Here,

$ECLOSE(1) = \{1, 2, 4\}$

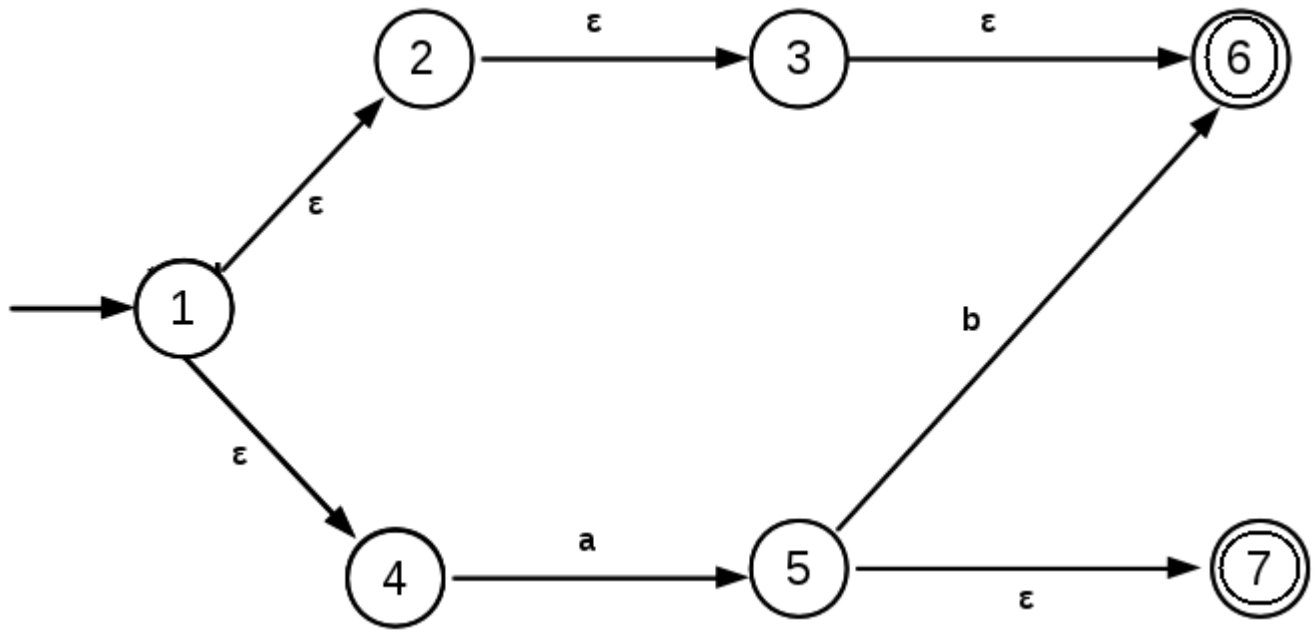
$ECLOSE(2) = \{2, 4\}$

$ECLOSE(3) = \{3\}$

$ECLOSE(4) = \{4\}$

**Example 2:**

Consider the following NFA,




Find the epsilon closure of all states.

$ECLOSE(1) = \{1, 2, 3, 6, 4\}$

$ECLOSE(2) = \{2, 3, 6\}$

$ECLOSE(3) = \{3, 6\}$

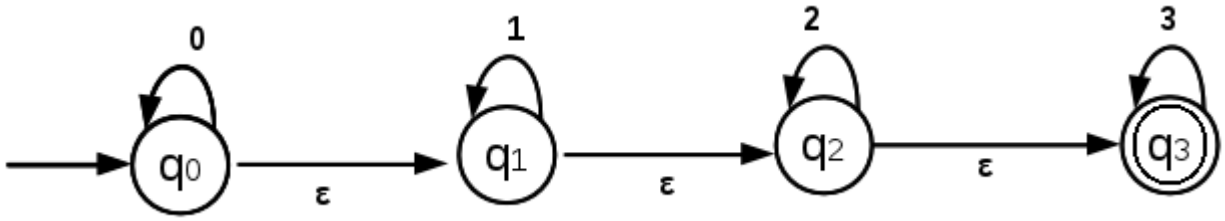
$ECLOSE(4) = \{4\}$

$ECLOSE(5) = \{5, 7\}$

$ECLOSE(6) = \{6\}$

**Example 3:**

Consider the following NFA,




Compute the Epsilon closure of all states.

$ECLOSE(q_0) = \{q_0, q_1, q_2, q_3\}$

$ECLOSE(q_1) = \{q_1, q_2, q_3\}$

$ECLOSE(q_2) = \{q_2, q_3\}$

$ECLOSE(q_3) = \{q_3\}$



## 6 Conversion of NFA to DFA

The steps involved in the conversion of NFA to DFA are,

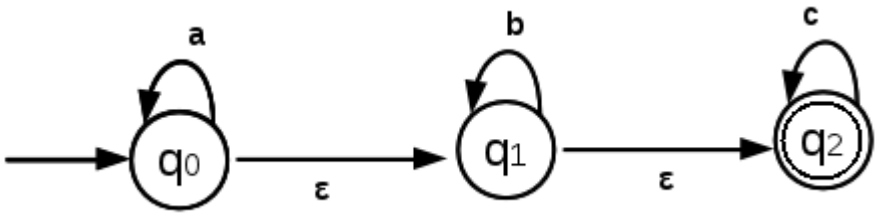
1. Transform the NFA with Epsilon transitions to NFA without epsilon transitions.
2. convert the resulting NFA to DFA.

These steps are exaplained in detail as follows:

### 1. Transform the NFA with $\varepsilon$ transitions to NFA without $\varepsilon$ transitions.

**Example 1:**

Consider the following NFA with epsilon transitions:




The above NFA has states, $q_0, q_1, q_2$. The start state is $q_0$. The final state is $q_2$.

Step a: Find the Epsion closure of all states.

$ECLOSE(q_0) = \{q_0, q_1, q_2\}$

$ECLOSE(q_1) = \{q_1, q_2\}$

$$ECLOSE(q_2) = \{q_2\}$$

The states of the new NFA will be $\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\}$.

The start state of the new NFA will be the epsilon closure of the start state $q_0$.

Thus the start state of the new NFA will be $\{q_0, q_1, q_2\}$

The final states of the new NFA will be those new states that contain the final states of the old NFA.

Thus the final states of the new NFA are $\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\}$, since they contain the final state $q_2$ of the old NFA.

Next we need to find the transitions of these new states on input symbols a, b and c.

Consider state $\{q_0, q_1, q_2\}$,

$$\begin{aligned}
\delta'(\{q_0, q_1, q_2\}, a) &= ECLOSE[\delta(\{q_0, q_1, q_2\}, a)] \\
&= ECLOSE[\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)] \\
&= ECLOSE[q_0 \cup \phi \cup \phi] \\
&= ECLOSE[q_0] \\
&= \{q_0, q_1, q_2\} \\
\delta'(\{q_0, q_1, q_2\}, b) &= ECLOSE[\delta(\{q_0, q_1, q_2\}, b)] \\
&= ECLOSE[\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)] \\
&= ECLOSE[\phi \cup q_1 \cup \phi] \\
&= ECLOSE[q_1] \\
&= \{q_1, q_2\} \\
\delta'(\{q_0, q_1, q_2\}, c) &= ECLOSE[\delta(\{q_0, q_1, q_2\}, c)] \\
&= ECLOSE[\delta(q_0, c) \cup \delta(q_1, c) \cup \delta(q_2, c)] \\
&= ECLOSE[\phi \cup \phi \cup q_2] \\
&= ECLOSE[q_2] \\
&= \{q_2\}
\end{aligned}$$

Consider the state $\{q_1, q_2\}$

$$\begin{aligned}
\delta'(\{q_1, q_2\}, a) &= ECLOSE[\delta(\{q_1, q_2\}, a)] \\
&= ECLOSE[\delta(q_1, a) \cup \delta(q_2, a)] \\
&= ECLOSE[\phi \cup \phi] \\
&= ECLOSE[\phi] \\
&= \phi \\
\delta'(\{q_1, q_2\}, b) &= ECLOSE[\delta(\{q_1, q_2\}, b)] \\
&= ECLOSE[\delta(q_1, b) \cup \delta(q_2, b)] \\
&= ECLOSE[q_1 \cup \phi] \\
&= ECLOSE[q_1] \\
&= \{q_1, q_2\}
\end{aligned}$$

$$\begin{aligned}\delta'(\{q_1,q_2\},c) &= ECLOSE[\delta(\{q_1,q_2\},c)]\\ &= ECLOSE[\delta(q_1,c)\cup\delta(q_2,c)]\\ &= ECLOSE[\phi\cup q_2]\\ &= ECLOSE[q_2]\\ &= \{q_2\}\end{aligned}$$

Conisider the state $q_2$

$$\begin{aligned}\delta'(q_2,a) &= ECLOSE[\delta(q_2,a)]\\ &= ECLOSE[\phi]\\ &= \phi\end{aligned}$$

$$\begin{aligned}\delta'(q_2,b) &= ECLOSE[\delta(q_2,b)]\\ &= ECLOSE[\phi]\\ &= \phi\end{aligned}$$

$$\begin{aligned}\delta'(q_2,c) &= ECLOSE[\delta(q_2,c)]\\ &= ECLOSE[q_2]\\ &= \{q_2\}\end{aligned}$$

Now we have the new NFA without epsilon transitions.

The transition table for the new NFA is,

| | Input Symbol | | |
|---|---|---|---|
| Current state | a | b | c |
| $\longrightarrow *\{q_0,q_1,q_2\}$ | $\{q_0,q_1,q_2\}$ | $\{q_1,q_2\}$ | $\{q_2\}$ |
| $*\{q_1,q_2\}$ | $\phi$ | $\{q_1,q_2\}$ | $\{q_2\}$ |
| $*\{q_2\}$ | $\phi$ | $\phi$ | $\{q_2\}$ |

Let us say,

$\{q_0,q_1,q_2\}$ as $q_x$

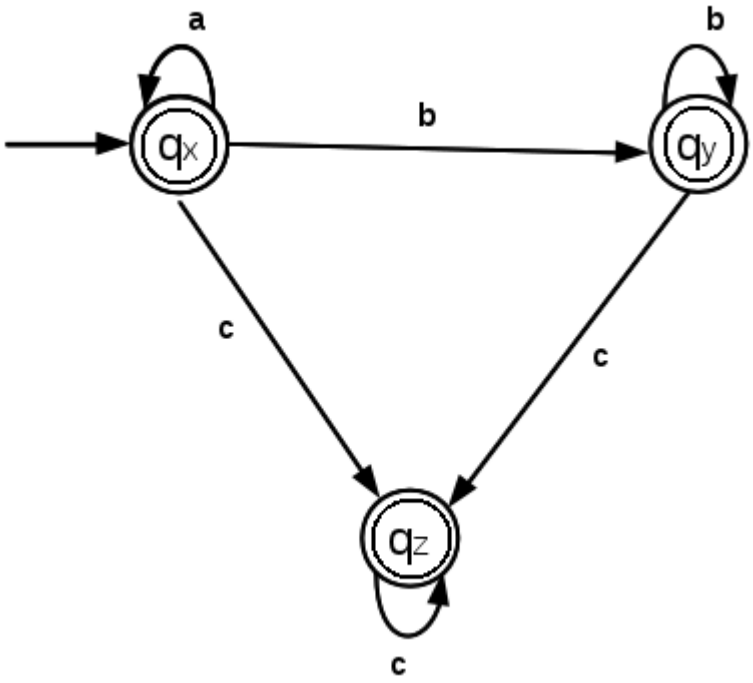
$\{q_1,q_2\}$ as $q_y$

$\{q_2\}$ as $q_z$

Then the transition table will become,

| | Input Symbol | | |
|---|---|---|---|
| Current state | a | b | c |
| $\longrightarrow *q_x$ | $q_x$ | $q_y$ | $q_z$ |
| $*q_y$ | $\phi$ | $q_y$ | $q_z$ |
| $*q_z$ | $\phi$ | $\phi$ | $q_z$ |

The transition diagram for the new NFA is,

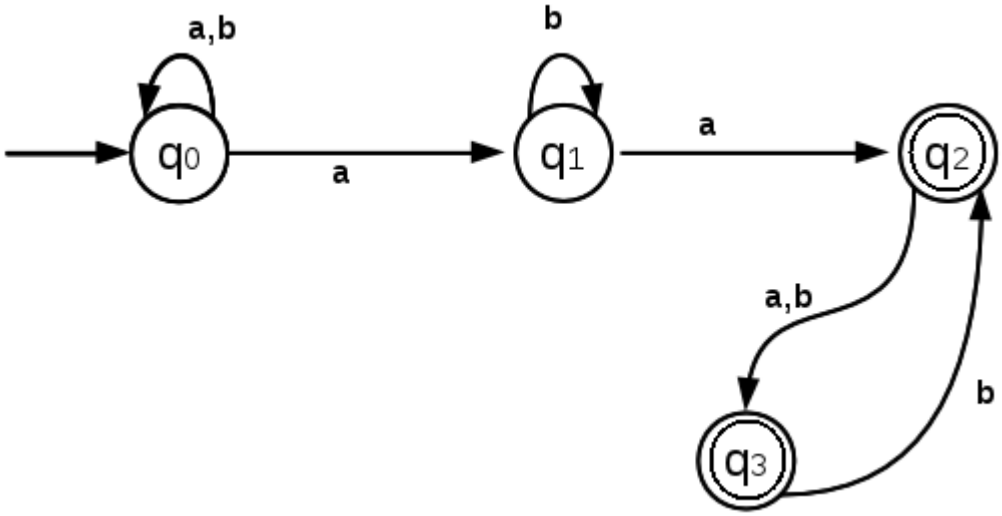



Above is the NFA without epsilon transitions.

Note that above NFA is also a DFA.

**Example 2:**

Convert the following NFA to DFA.




Step 1: Transform the NFA with Epsilon transitions to NFA without epsilon transitions.

Note that above NFA does not contain any epsilon transitions.

Step 2: Convert the resulting NFA to DFA.

Step a:

Consider the start state $q_0$,

Seek all the transitions from state $q_0$ for all input symbols a and b.

We get,

$\delta(q_0, a) = \{q_0, q_1\}$

$\delta(q_0, b) = \{q_0\}$

Now we got,

| | Input | symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow q_0$ | $\{q_0, q_1\}$ | $q_0$ |

Here we got a new state, $\{q_0, q_1\}$

Step b:

Consider the state $\{q_0, q_1\}$,

Seek all the transitions from state $\{q_0, q_1\}$for all input symbols a and b.

We get,

$\delta(\{q_0, q_1\}, a) = \delta(q_0, a) \cup \delta(q_1, a)$

$= \{q_0, q_1\} \cup q_2$

$= \{q_0, q_1, q_2\}$

$\delta(\{q_0, q_1\}, b) = \delta(q_0, b) \cup \delta(q_1, b)$

$= q_0 \cup q_1$

$= \{q_0, q_1\}$

Now we got,

| | Input | symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow q_0$ | $\{q_0, q_1\}$ | $q_0$ |
| $\{q_0, q_1\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0, q_1\}$ |

Here we got a new state, $\{q_0, q_1, q_2\}$

Step c:

Consider the state $\{q_0, q_1, q_2\}$,

Seek all the transitions from state $\{q_0, q_1, q_2\}$for all input symbols a and b.

We get,

$\delta(\{q_0, q_1, q_2\}, a) = \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)$

$= \{q_0, q_1\} \cup q_2 \cup q_3$

$= \{q_0, q_1, q_2, q_3\}$

$\delta(\{q_0, q_1, q_2\}, b) = \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)$

$= q_0 \cup q_1 \cup q_3$

$= \{q_0, q_1, q_3\}$

Now we got,

| | Input | symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow q_0$ | $\{q_0, q_1\}$ | $q_0$ |
| $\{q_0, q_1\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0, q_1\}$ |
| $\{q_0, q_1, q_2\}$ | $\{q_0, q_1, q_2, q_3\}$ | $\{q_0, q_1, q_3\}$ |

Here we got 2 new states, $\{q_0, q_1, q_2, q_3\}$ and $\{q_0, q_1, q_3\}$.

Step d:

Consider each one of these states, $\{q_0, q_1, q_2, q_3\}$ and $\{q_0, q_1, q_3\}$

Step d.i:

Consider the state $\{q_0, q_1, q_2, q_3\}$,

Seek all the transitions from state $\{q_0, q_1, q_2, q_3\}$for all input symbols a and b.

We get,

$$\delta(\{q_0, q_1, q_2, q_3\}, a) = \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a)$$
$$= \{q_0, q_1\} \cup q_2 \cup q_3 \cup \phi$$
$$= \{q_0, q_1, q_2, q_3\}$$
$$\delta(\{q_0, q_1, q_2, q_3\}, b) = \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b) \cup \delta(q_3, b)$$
$$= q_0 \cup q_1 \cup q_3 \cup q_2$$
$$= \{q_0, q_1, q_2, q_3\}$$

Now we got,

| | Input | symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow q_0$ | $\{q_0, q_1\}$ | $q_0$ |
| $\{q_0, q_1\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0, q_1\}$ |
| $\{q_0, q_1, q_2\}$ | $\{q_0, q_1, q_2, q_3\}$ | $\{q_0, q_1, q_3\}$ |
| $\{q_0, q_1, q_2, q_3\}$ | $\{q_0, q_1, q_2, q_3\}$ | $\{q_0, q_1, q_2, q_3\}$ |

Step d.i:

Consider the state $\{q_0, q_1, q_3\}$,

Seek all the transitions from state $\{q_0, q_1, q_3\}$for all input symbols a and b.

We get,

$$\delta(\{q_0, q_1, q_3\}, a) = \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_3, a)$$
$$= \{q_0, q_1\} \cup q_2 \cup \phi$$
$$= \{q_0, q_1, q_2\}$$
$$\delta(\{q_0, q_1, q_3\}, b) = \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_3, b)$$
$$= q_0 \cup q_1 \cup q_2$$
$$= \{q_0, q_1, q_2\}$$

Now we got,

| | Input | symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow q_0$ | $\{q_0, q_1\}$ | $q_0$ |
| $\{q_0, q_1\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0, q_1\}$ |
| $\{q_0, q_1, q_2\}$ | $\{q_0, q_1, q_2, q_3\}$ | $\{q_0, q_1, q_3\}$ |
| $\{q_0, q_1, q_2, q_3\}$ | $\{q_0, q_1, q_2, q_3\}$ | $\{q_0, q_1, q_2, q_3\}$ |
| $\{q_0, q_1, q_3\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0, q_1, q_2\}$ |

Now there are no new states generated. above is the transition table corresponding to the new DFA.

In the above DFA, the start state is $q_0$.

The final states of the DFA are $\{q_0, q_1, q_2\}$,$\{q_0, q_1, q_2, q_3\}$,$\{q_0, q_1, q_3\}$, since they contain at least one final state of the NFA ($q_2$ or $q_3$).

Let us say,

$q_0$ as A,

$\{q_0, q_1\}$ as B,

$\{q_0, q_1, q_2\}$ as C,

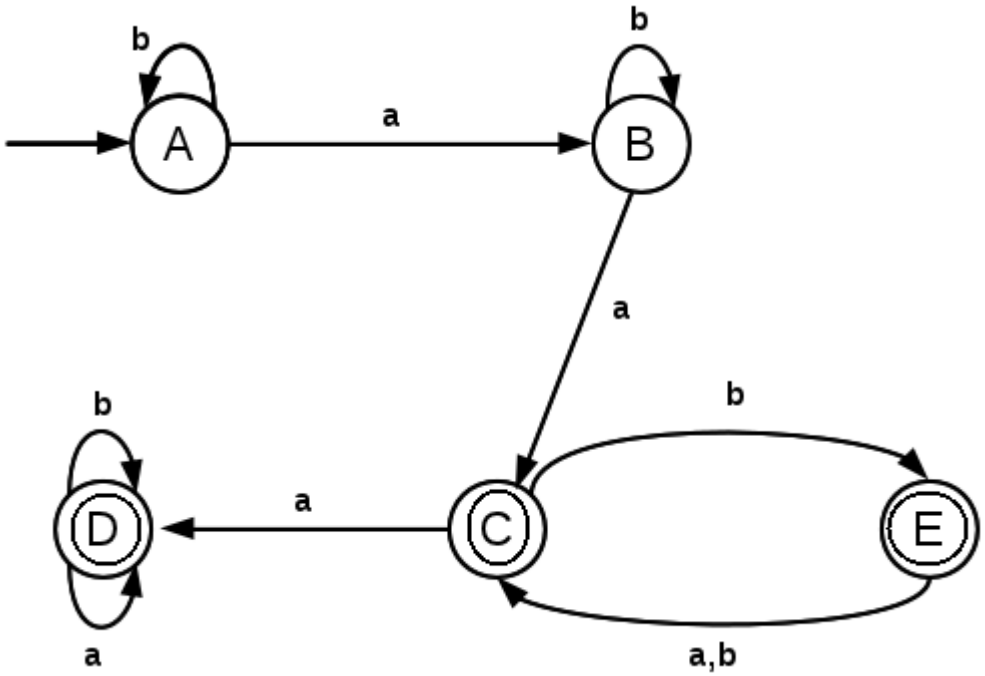
$\{q_0, q_1, q_2, q_3\}$ as D, and

$\{q_0, q_1, q_3\}$ as E

Then the transition table will become,

| | Input | symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow$A | B | A |
| B | C | B |
| * C | D | E |
| * D | D | D |
| * E | C | C |

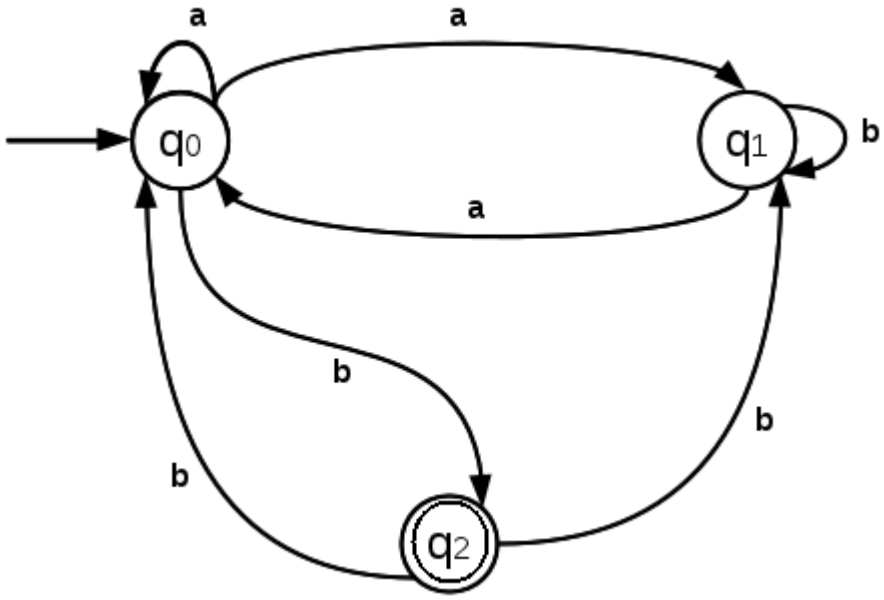
The transition diagram corresponding to the DFA is,




From the above diagram, it is a DFA since, there is only one transition corresponding to an input symbol from a state.

**Example 3:**

Consider the following NFA,




Step 1: Transform the NFA with Epsilon transitions to NFA without epsilon transitions.

Note that above NFA does not contain any epsilon transitions.

Step 2: Convert the resulting NFA to DFA.

Consider the start state $q_0$,

Seek all the transitions from state $q_0$ for all input symbols a and b.

We get,

$\delta(q_0, a) = \{q_0, q_1\}$

$\delta(q_0, b) = \{q_2\}$

Now we got,

| | Input | symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow q_0$ | $\{q_0, q_1\}$ | $q_2$ |

Step b:

Step b.i:

$\delta(\{q_0, q_1\}, a) = \delta(q_0, a) \cup \delta(q_1, a)$

$= \{q_0, q_1\} \cup q_0$

$= \{q_0, q_1\}$

$\delta(\{q_0, q_1\}, b) = \delta(q_0, b) \cup \delta(q_1, b)$

$= q_2 \cup q_1$

$= \{q_1, q_2\}$

Now we got,

| | Input | symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow q_0$ | $\{q_0, q_1\}$ | $q_2$ |
| $\{q_0, q_1\}$ | $\{q_0, q_1\}$ | $\{q_1, q_2\}$ |

Step b. i. i:

$$\delta(\{q_1, q_2\}, a) = \delta(q_1, a) \cup \delta(q_2, a)$$
$$= q_0 \cup \phi$$
$$= q_0$$
$$\delta(\{q_1, q_2\}, b) = \delta(q_1, b) \cup \delta(q_2, b)$$
$$= q_1 \cup \{q_0, q_1\}$$
$$= \{q_0, q_1\}$$

Now we got,

| | Input | symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow q_0$ | $\{q_0, q_1\}$ | $q_2$ |
| $\{q_0, q_1\}$ | $\{q_0, q_1\}$ | $\{q_1, q_2\}$ |
| $\{q_1, q_2\}$ | $q_0$ | $\{q_0, q_1\}$ |

Step b.ii

$$\delta(q_2, a) = \phi$$
$$\delta(q_2, b) = \{q_0 \cup q_1\}$$

Now we got,

| | Input | symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow q_0$ | $\{q_0, q_1\}$ | $q_2$ |
| $\{q_0, q_1\}$ | $\{q_0, q_1\}$ | $\{q_1, q_2\}$ |
| $*\{q_1, q_2\}$ | $q_0$ | $\{q_0, q_1\}$ |
| $*q_2$ | $\phi$ | $\{q_0, q_1\}$ |

Thus there are no more new states. The above is the transition table for the new NFA.

The start state is $q_0$.

The final states are $q_2$ and $\{q_1, q_2\}$, since they contain the final state, $q_2$ of the NFA.

Let we say,

$q_0$as A

$\{q_0, q_1\}$ as B

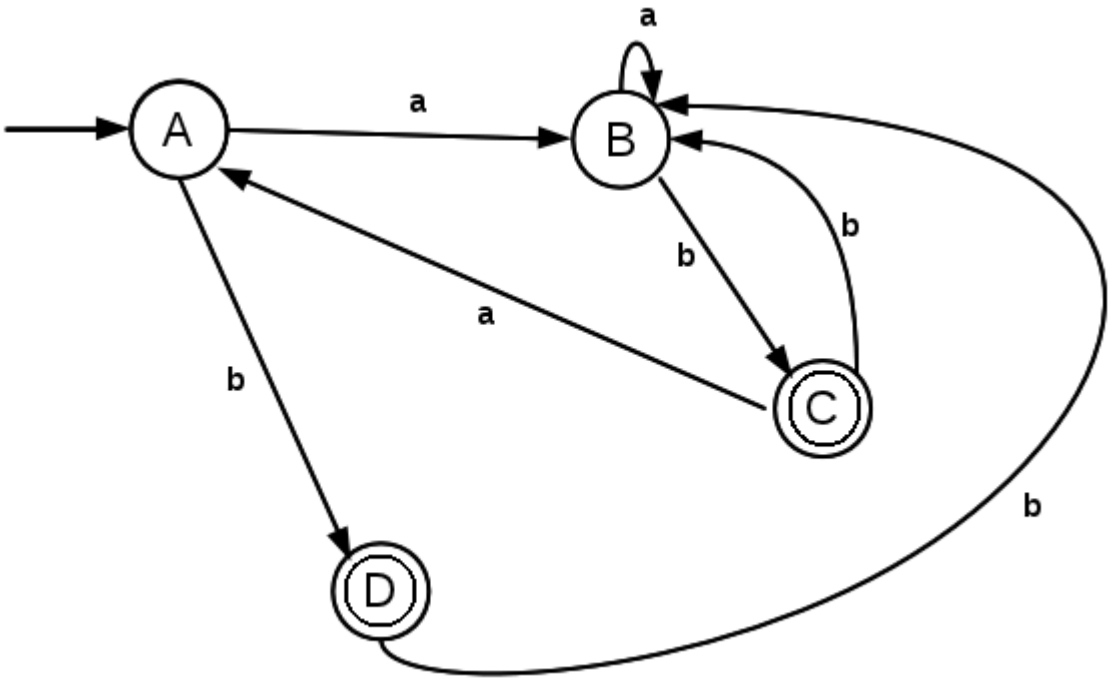
$\{q_1, q_2\}$ as C

$q_2$ as D.

The new transition table is,

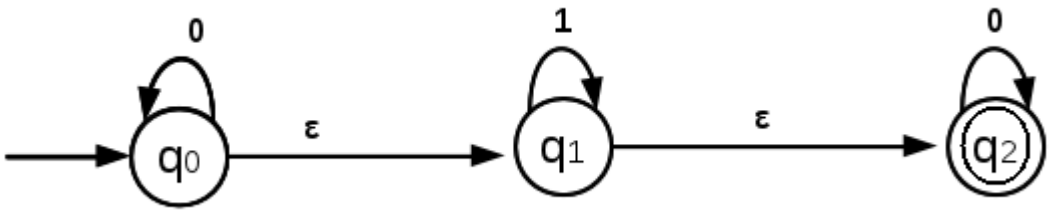
| | Input | symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow$A | B | D |
| B | B | C |
| * C | A | B |
| * D | $\phi$ | B |

The transition diagram for the DFA is




**Example 4:**

Consider the NFA,




Convert the above NFA to DFA.

Step 1: Transform the NFA with Epsilon transitions to NFA without epsilon transitions.

Step a:

The states of the new NFA will be $ECLOSE(q_0), ECLOSE(q_1), ECLOSE(q_2)$.

Find the Epsion closure of all states.

$ECLOSE(q_0) = \{q_0, q_1, q_2\}$

$ECLOSE(q_1) = \{q_1, q_2\}$

$ECLOSE(q_2) = \{q_2\}$

The states of the new NFA will be $\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\}$.

The start state of the new NFA will be the epsilon closure of the start state $q_0$.

Thus the start state of the new NFA will be $\{q_0, q_1, q_2\}$

The final states of the new NFA will be those new states that contain the final states of the old NFA.

Thus the final states of the new NFA are $\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\}$, since they contain the final state $q_2$ of the

old NFA.

Next we need to find the transitions of these new states on input symbols 0 and 1.

Consider state $\{q_0, q_1, q_2\}$,

$$\begin{aligned}
\delta'(\{q_0,q_1,q_2\},0) &= ECLOSE[\delta(\{q_0,q_1,q_2\},0)] \\
&= ECLOSE[\delta(q_0,0) \cup \delta(q_1,0) \cup \delta(q_2,0)] \\
&= ECLOSE[q_0 \cup \phi \cup q_2] \\
&= ECLOSE[q_0,q_2] \\
&= \{q_0,q_1,q_2\} \\
\delta'(\{q_0,q_1,q_2\},1) &= ECLOSE[\delta(\{q_0,q_1,q_2\},1)] \\
&= ECLOSE[\delta(q_0,1) \cup \delta(q_1,1) \cup \delta(q_2,1)] \\
&= ECLOSE[\phi \cup q_1 \cup \phi] \\
&= ECLOSE[q_1] \\
&= \{q_1,q_2\} \\
\delta'(\{q_1,q_2\},0) &= ECLOSE[\delta(\{q_1,q_2\},0)] \\
&= ECLOSE[\delta(q_1,0) \cup \delta(q_2,0)] \\
&= ECLOSE[\phi \cup q_2] \\
&= ECLOSE[q_2] \\
&= \{q_2\} \\
\delta'(\{q_1,q_2\},1) &= ECLOSE[\delta(\{q_1,q_2\},1)] \\
&= ECLOSE[\delta(q_1,1) \cup \delta(q_2,1)] \\
&= ECLOSE[q_1 \cup \phi] \\
&= ECLOSE[q_1] \\
&= \{q_1,q_2\} \\
\delta'(q_2,0) &= ECLOSE[\delta(q_2,0)] \\
&= ECLOSE[q_2] \\
&= \{q_2\} \\
\delta'(q_2,1) &= ECLOSE[\delta(q_2,1)] \\
&= ECLOSE[\phi] \\
&= \phi
\end{aligned}$$

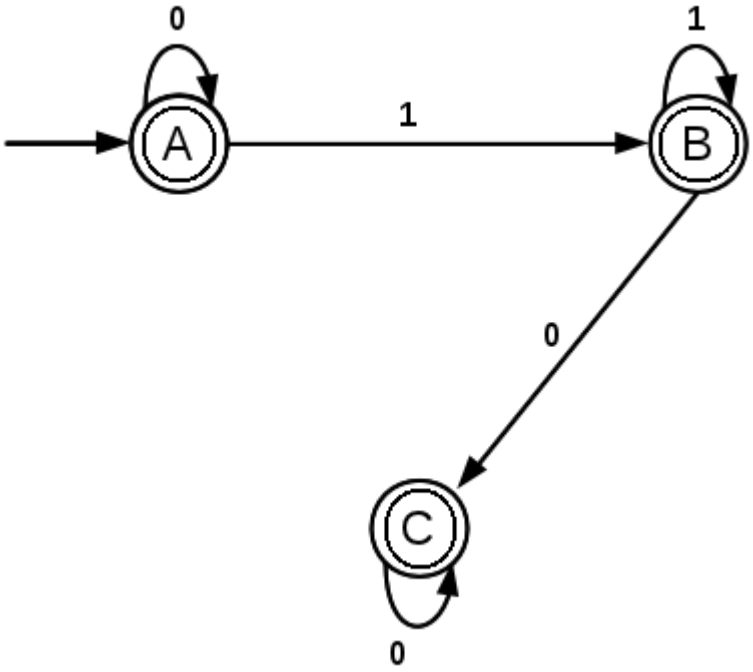
Let us say,

$\{q_0, q_1, q_2\}$ as A,

$\{q_1, q_2\}$ as B, and

$\{q_2\}$ as C.

Then the NFA without epsilon transitions is,



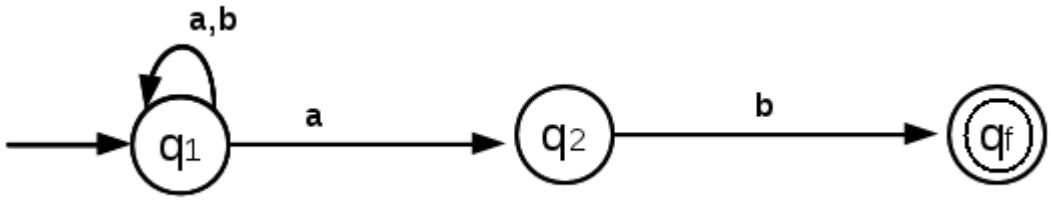

The above is an NFA without epsilon transitions.

Step 2: Convert the resulting NFA to DFA.

Note that above NFA is also a DFA, since from a state there is only one transition corresponding to an input symbol.

**Example 5:**

Consider the following NFA,




Convert this NFA to DFA.

Step 1: Transform the NFA with Epsilon transitions to NFA without epsilon transitions.

Above NFA does not contain epsilon transitions. So we need to eliminate epsilon transitions.

Step 2: Transform above NFA to DFA.

Begin from the start state, $q_1$,

$\delta(q_1, a) = \{q_1, q_2\}$

$\delta(q_1, b) = \{q_1\}$

Now we got,

| Current State | Input a | symbol b |
|---|---|---|
| $\longrightarrow q_1$ | $\{q_1, q_2\}$ | $q_1$ |

Step b:

$$\delta(\{q_1, q_2\}, a) = \delta(q_1, a) \cup \delta(q_2, a)$$
$$= \{q_1, q_2\} \cup \phi$$
$$= \{q_1, q_2\}$$
$$\delta(\{q_1, q_2\}, b) = \delta(q_1, b) \cup \delta(q_2, b)$$
$$= q_1 \cup q_f$$
$$= \{q_1, q_f\}$$

Now we got,

| Current State | Input a | symbol b |
|---|---|---|
| $\longrightarrow q_1$ | $\{q_1, q_2\}$ | $q_1$ |
| $\{q_1, q_2\}$ | $\{q_1, q_2\}$ | $\{q_1, q_f\}$ |

Step c:

$$\delta(\{q_1, q_f\}, a) = \delta(q_1, a) \cup \delta(q_f, a)$$
$$= \{q_1, q_2\} \cup \phi$$
$$= \{q_1, q_2\}$$
$$\delta(\{q_1, q_f\}, b) = \delta(q_1, b) \cup \delta(q_f, b)$$
$$= q_1 \cup \phi$$
$$= \{q_1\}$$

Now we got,

| Current State | Input a | symbol b |
|---|---|---|
| $\longrightarrow q_1$ | $\{q_1, q_2\}$ | $q_1$ |
| $\{q_1, q_2\}$ | $\{q_1, q_2\}$ | $\{q_1, q_f\}$ |
| $*\{q_1, q_f\}$ | $\{q_1, q_2\}$ | $q_1$ |

The start state is $q_1$.

The final state is $\{q_1, q_f\}$.

Let us say,

$q_1$ as A,

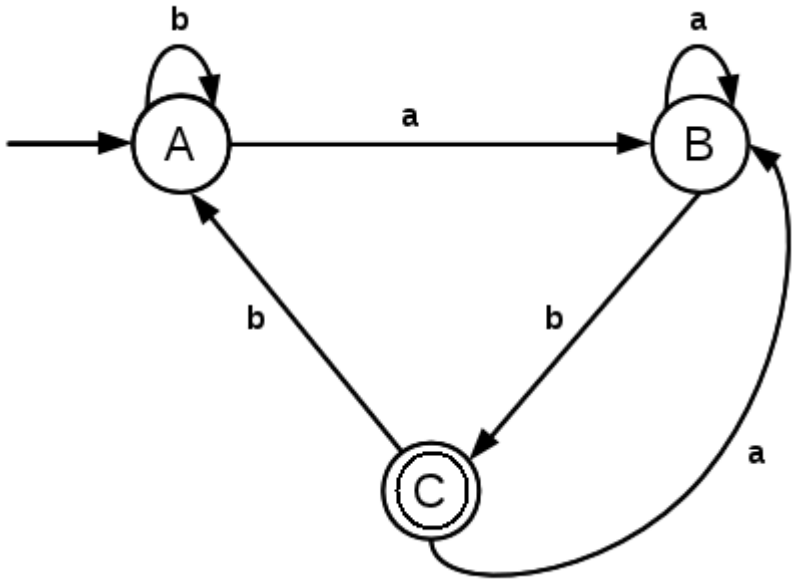
$\{q_1, q_2\}$ as B,

$\{q_1, q_f\}$ as C.

Then the transition table will become,

| Current State | Input symbol | |
|---|---|---|
| | a | b |
| ⟶A | B | A |
| B | B | C |
| * C | B | A |

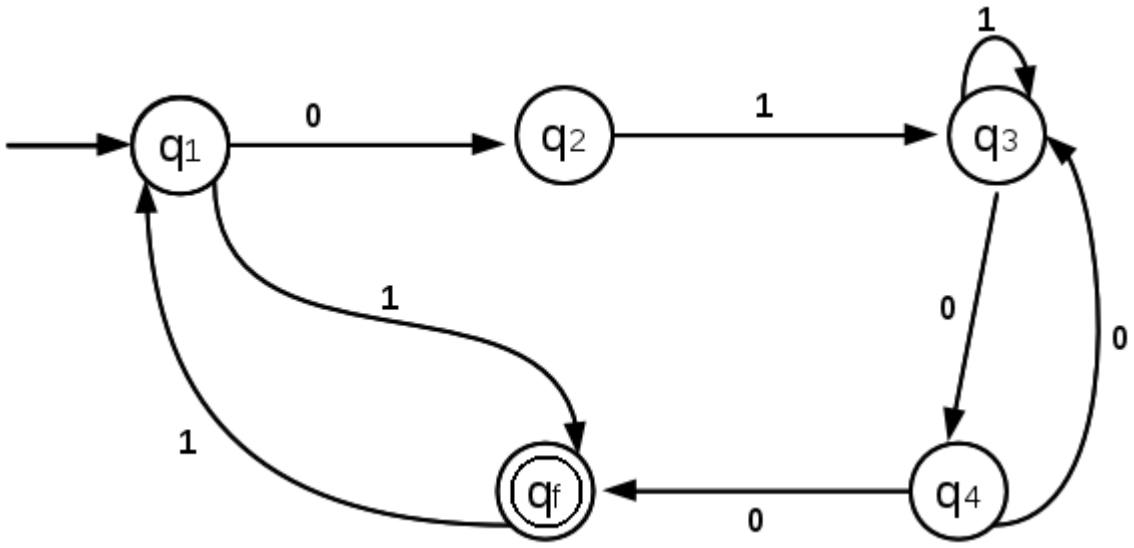
The transition diagram is,



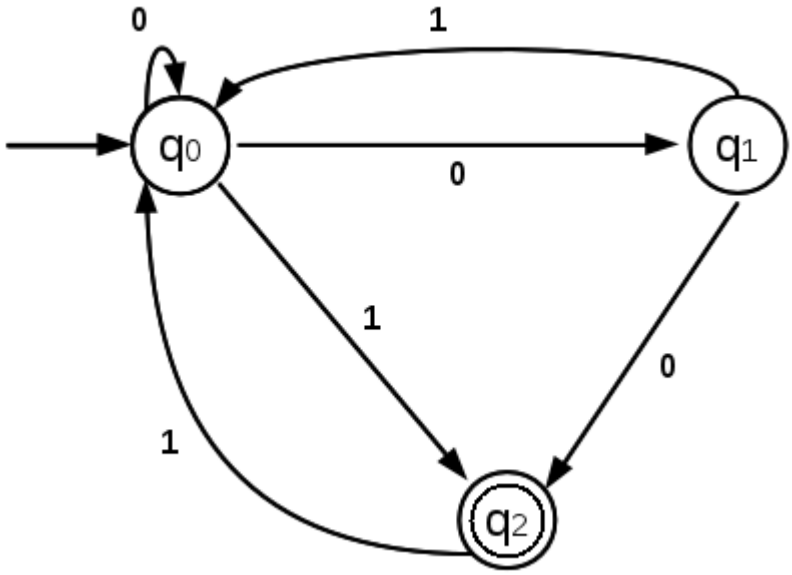

The above is a DFA, since there is only one transition corresponding to an input symbol from every state.

**Exercises:**

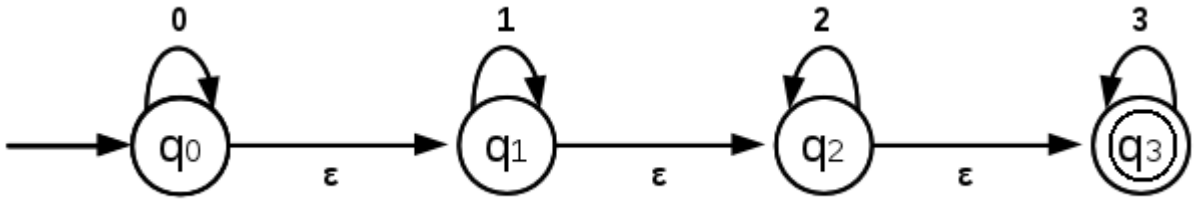
1. Convert the following NFA to DFA.




2. Convert the following NFA to DFA.




3. Convert the following NFA to DFA.



# Part IV. Minimization of DFA

For a given language, many DFAs may exist that accept it. The DFA we produce from an NFA may contain many dead states, inaccessible states and indistinguishable states. all these unnecessary states can be eliminated from a DFA through a process called minimization.

For practical applications, it is desirable that number of states in the DFA is minimum.

The algorithm for minimising a DFA is as follows:

Step 1: Eliminate any state that cannot be reached from the start state.

Step 2: Partition the remaining states into blocks so that all states in the same block are equivalent, and no pair of states from different blocks are equivalent.

The process is demonstrated using an example:



**Example 1:**

Minimise the following DFA,

| Current State | Input symbol a | b |
|---|---|---|
| $\longrightarrow q_0$ | $q_5$ | $q_1$ |
| $q_1$ | $q_2$ | $q_6$ |
| $*q_2$ | $q_2$ | $q_0$ |
| $q_4$ | $q_5$ | $q_7$ |
| $q_5$ | $q_6$ | $q_2$ |
| $q_6$ | $q_4$ | $q_6$ |
| $q_7$ | $q_2$ | $q_6$ |
| $q_3$ | $q_6$ | $q_2$ |

Step 1: Eliminate any state that cannot be reached from the start state.

This is done by enumerating all the simple paths in the graph of the DFA beginning from the start state. Any state that is not part of some path is unreachable.

In the above, the state $q_3$ cannot be reached. So remove the corresponding to $q_3$ from the transition table.

Now the new transition table is,

| Current State | Input symbol a | b |
|---|---|---|
| $\longrightarrow q_0$ | $q_5$ | $q_1$ |
| $q_1$ | $q_2$ | $q_6$ |
| $*q_2$ | $q_2$ | $q_0$ |
| $q_4$ | $q_5$ | $q_7$ |
| $q_5$ | $q_6$ | $q_2$ |
| $q_6$ | $q_4$ | $q_6$ |
| $q_7$ | $q_2$ | $q_6$ |

Step 2:

Divide the rows of the transition table into 2 sets as,

1. One set containing only those rows which starts from non-final states.

Set 1

| | | |
|---|---|---|
| $q_0$ | $q_5$ | $q_1$ |
| $q_1$ | $q_2$ | $q_6$ |
| $q_4$ | $q_5$ | $q_7$ |
| $q_5$ | $q_6$ | $q_2$ |
| $q_6$ | $q_4$ | $q_6$ |
| $q_7$ | $q_2$ | $q_6$ |

2. Another set containing those rows which start from final states.

Set 2

| | | |
|---|---|---|
| $*q_2$ | $q_2$ | $q_0$ |

Step 3a. Consider Set 1.

Find out the similar rows:

| | | | |
|---|---|---|---|
| $q_0$ | $q_5$ | $q_1$ | Row 1 |
| $q_1$ | $q_2$ | $q_6$ | Row 2 |
| $q_4$ | $q_5$ | $q_7$ | Row 3 |
| $q_5$ | $q_6$ | $q_2$ | Row 4 |
| $q_6$ | $q_4$ | $q_6$ | Row 5 |
| $q_7$ | $q_2$ | $q_6$ | Row 6 |

Row 2 and Row 6 are similar, since $q_1$ and $q_7$ transit to same states on inputs a and b.

So remove one of them (for instance, $q_7$) and replace $q_7$ with $q_1$ in the rest.

We get,

Set 1

| | | | |
|---|---|---|---|
| $q_0$ | $q_5$ | $q_1$ | Row 1 |
| $q_1$ | $q_2$ | $q_6$ | Row 2 |
| $q_4$ | $q_5$ | $q_1$ | Row 3 |
| $q_5$ | $q_6$ | $q_2$ | Row 4 |
| $q_6$ | $q_4$ | $q_6$ | Row 5 |

Now Row 1 and Row 3 are similar. So remove one of them (for instance, $q_4$) and replace $q_4$ with $q_0$ in the rest.

We get,

Set 1

| | | | |
|---|---|---|---|
| $q_0$ | $q_5$ | $q_1$ | Row 1 |
| $q_1$ | $q_2$ | $q_6$ | Row 2 |
| $q_5$ | $q_6$ | $q_2$ | Row 3 |
| $q_6$ | $q_0$ | $q_6$ | Row 4 |

Now there are no more similar rows.

Step 3b.

Consider Set 2,

Set 2

| | | |
|---|---|---|
| $*q_2$ | $q_2$ | $q_0$ |

Do the same process for Set 2.

But it contains only one row. It is already minimized.

Step 4:

Combine Set 1 and Set 2

We get,

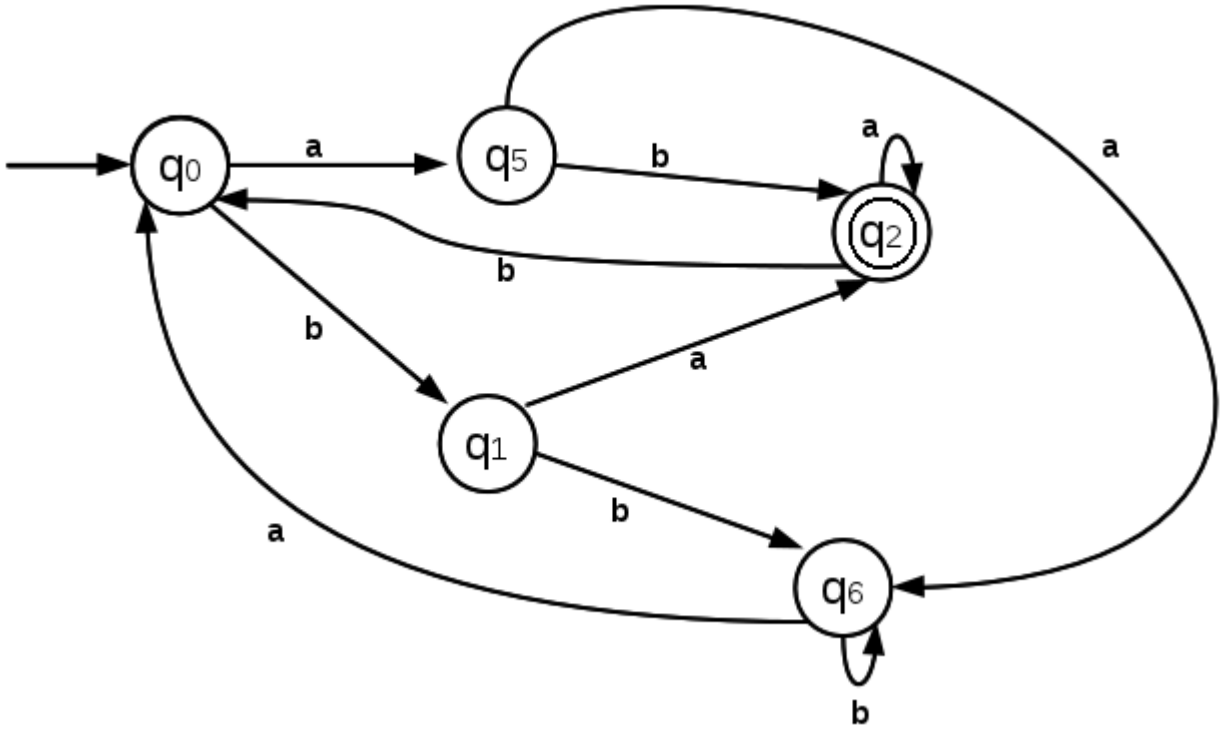
| | | | |
|---|---|---|---|
| $q_0$ | $q_5$ | $q_1$ | Row 1 |
| $q_1$ | $q_2$ | $q_6$ | Row 2 |
| $q_5$ | $q_6$ | $q_2$ | Row 3 |
| $q_6$ | $q_0$ | $q_6$ | Row 4 |
| $*q_2$ | $q_2$ | $q_0$ | Row 5 |

The DFA corresponding to this is,

| | Input | symbol |
|---|---|---|
| Current State | a | b |
| $\longrightarrow q_0$ | $q_5$ | $q_1$ |
| $q_1$ | $q_2$ | $q_6$ |
| $q_5$ | $q_6$ | $q_2$ |
| $q_6$ | $q_0$ | $q_6$ |
| $*q_2$ | $q_2$ | $q_0$ |

Now this is a minimised DFA.

Transition diagram is,




**Example 2:**

Minimise the following DFA,

| | Input | symbol |
|---|---|---|
| Current State | 0 | 1 |
| $\longrightarrow q_0$ | $q_1$ | $q_3$ |
| $q_1$ | $q_0$ | $q_3$ |
| $q_2$ | $q_1$ | $q_4$ |
| $*q_3$ | $q_5$ | $q_5$ |
| $q_4$ | $q_3$ | $q_3$ |
| $*q_5$ | $q_5$ | $q_5$ |

Step 1: Eliminate any state that cannot be reached from the start state.

This is done by enumerating all the simple paths in the graph of the DFA beginning from the start state. Any state that is not part of some path is unreachable.

In the above, the states $q_2$ and $q_4$ cannot be reached. So remove the rows corresponding to $q_2$ and $q_4$ from the transition table.

We get,

| Current State | Input symbol | |
|---|---|---|
| | 0 | 1 |
| $\longrightarrow q_0$ | $q_1$ | $q_3$ |
| $q_1$ | $q_0$ | $q_3$ |
| $*q_3$ | $q_5$ | $q_5$ |
| $*q_5$ | $q_5$ | $q_5$ |

Step 2: Divide the rows of the transition table into 2 sets as,

1. One set containing only those rows which starts from non-final states.

   Set 1

| | | |
|---|---|---|
| $q_0$ | $q_1$ | $q_3$ |
| $q_1$ | $q_0$ | $q_3$ |

2. Another set containing those rows which start from final states.

   Set 2

| | | |
|---|---|---|
| $*q_3$ | $q_5$ | $q_5$ |
| $*q_5$ | $q_5$ | $q_5$ |

Step 3a. Consider Set 1.

Find out the similar rows:

Set 1

| | | | |
|---|---|---|---|
| $q_0$ | $q_1$ | $q_3$ | Row 1 |
| $q_1$ | $q_0$ | $q_3$ | Row 2 |

There are no similar rows.

Step 3b. Consider Set 2.

Find out the similar rows:

Set 2

| | | | |
|---|---|---|---|
| $*q_3$ | $q_5$ | $q_5$ | Row 1 |
| $*q_5$ | $q_5$ | $q_5$ | Row 2 |

Row 1 and Row 2 are similar, since $q_3$ and $q_5$ transit to same states on inputs 0 and 1.

So remove one of them (for instance, $q_5$) and replace $q_5$ with $q_3$ in the rest.

We get,

| | | | |
|---|---|---|---|
| $*q_3$ | $q_3$ | $q_3$ | Row 1 |

Step 4:

Combine Set 1 and Set 2

We get,

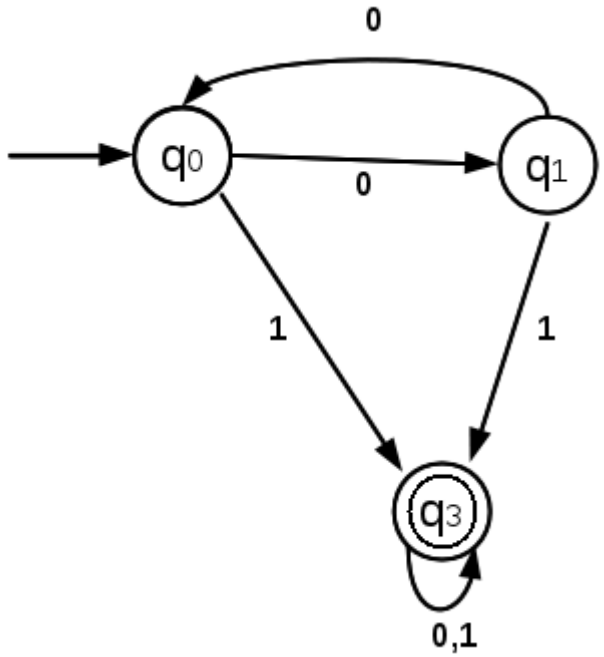
| | | | |
|---|---|---|---|
| $q_0$ | $q_1$ | $q_3$ | Row 1 |
| $q_1$ | $q_0$ | $q_3$ | Row 2 |
| $*q_3$ | $q_3$ | $q_3$ | Row 3 |

The DFA corresponding to this is,

| Current State | Input symbol | |
|---|---|---|
| | a | b |
| $\longrightarrow q_0$ | $q_1$ | $q_3$ |
| $q_1$ | $q_0$ | $q_3$ |
| $*q_3$ | $q_3$ | $q_3$ |

Now this is a minimised DFA.

Transition diagram is,




**Exercise:**

1. Minimise the following DFA represnted as transition table,

| Current State | Input symbol | |
|---|---|---|
| | 0 | 1 |
| $\longrightarrow q_0$ | $q_1$ | $q_2$ |
| $q_1$ | $q_2$ | $q_3$ |
| $q_2$ | $q_2$ | $q_4$ |
| $*q_3$ | $q_3$ | $q_3$ |
| $*q_4$ | $q_4$ | $q_4$ |
| $q_5$ | $q_5$ | $q_4$ |

2. Minimise the following DFA represnted as transition table,

| Current State | Input symbol 0 | 1 |
|---|---|---|
| $\longrightarrow q_0$ | $q_1$ | $q_3$ |
| $q_1$ | $q_2$ | $q_4$ |
| $*q_2$ | $q_1$ | $q_4$ |
| $q_3$ | $q_2$ | $q_4$ |
| $*q_4$ | $q_4$ | $q_4$ |

3. Minimise the following DFA represnted as transition table,

| Current State | Input symbol 0 | 1 |
|---|---|---|
| $\longrightarrow q_0$ | $q_1$ | $q_3$ |
| $q_1$ | $q_0$ | $q_3$ |
| $q_2$ | $q_1$ | $q_4$ |
| $*q_3$ | $q_5$ | $q_5$ |
| $q_4$ | $q_3$ | $q_3$ |
| $*q_5$ | $q_5$ | $q_5$ |

4. Minimise the following DFA represnted as transition table,

| Current State | Input symbol 0 | 1 |
|---|---|---|
| $\longrightarrow q_0$ | $q_1$ | $q_5$ |
| $q_1$ | $q_6$ | $q_2$ |
| $*q_2$ | $q_0$ | $q_2$ |
| $q_3$ | $q_2$ | $q_6$ |
| $q_4$ | $q_7$ | $q_5$ |
| $q_5$ | $q_2$ | $q_6$ |
| $q_6$ | $q_6$ | $q_4$ |
| $q_7$ | $q_6$ | $q_2$ |

5. Minimise the following DFA represnted as transition table,

| Current State | Input symbol a | Input symbol b |
|---|---|---|
| $\longrightarrow q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_0$ | $q_2$ |
| $q_2$ | $q_3$ | $q_1$ |
| $*q_3$ | $q_3$ | $q_0$ |
| $q_4$ | $q_3$ | $q_5$ |
| $q_5$ | $q_6$ | $q_4$ |
| $q_6$ | $q_5$ | $q_6$ |
| $q_7$ | $q_6$ | $q_3$ |

6. Minimise the following DFA represnted as transition table,

| Current State | Input symbol a | Input symbol b |
|---|---|---|
| $\longrightarrow q_0$ | $q_1$ | $q_2$ |
| $q_1$ | $q_1$ | $q_3$ |
| $q_2$ | $q_3$ | $q_4$ |
| $q_3$ | $q_1$ | $q_5$ |
| $q_4$ | $q_4$ | $q_2$ |
| $*q_5$ | $q_6$ | $q_6$ |

7. Minimise the following DFA represnted as transition table,

| Current State | Input symbol a | Input symbol b |
|---|---|---|
| $\longrightarrow q_0$ | $q_1$ | $q_2$ |
| $q_1$ | $q_4$ | $q_3$ |
| $q_2$ | $q_4$ | $q_3$ |
| $*q_3$ | $q_5$ | $q_6$ |
| $*q_4$ | $q_7$ | $q_6$ |
| $q_5$ | $q_3$ | $q_6$ |
| $q_6$ | $q_6$ | $q_6$ |
| $q_7$ | $q_4$ | $q_6$ |

# Part V. Regular Expressions

We found in earlier sections that a language can be described by using an NFA or DFA. But a more concise way of describing a lnguage is by using regular expressions.

Thus one way of describing a language is via the notion of regular expressions.

For example,

$ab$,

$a$,

$ab^*$,

$(ab)^+$,

$abcd^*$,

$x/y$,

$a/bcd^*$,

$letter(letter/digit)^*$,

$a(b/c)d^*$

are different examples of regular expressions.

We will learn the meaning of all these symbols later.

A regular expression generates a set of strings.

For example, the regular expression, $a^*$ generates the strings, $\varepsilon, a, aa, aaa, aaaa.....$

The regular expression, $a(b/a)$ generates the strings $ab, ba$.

## 7 Definition of a Regular Expression

Let $\sum$ denotes the character set, ie. a, b, c....z.

The regular expression is defined by the following rules:

Rule 1: Every character of $\sum$ is a regular expression.

For example, a is a regular expression, b is a regular expression, c is a regular expression and so on.

Rule 2: Null string, $\varepsilon$ is a regular expression.

Rule 3: If R1 and R2 are regular expressions, then R1 R2 is also a regular expression.

For example, let R1 denotes $abc$ and

let R2 denotes $(pq)^*$,

then $abc(pq)^*$ is also a regular expression.

Rule 4: If R1 and R2 are regular expressions, then R1 / R2 is also a regular expression.

For example, let R1 denotes $(ab)^+$ and

let R2 denotes $cd$,

then $(ab)^+/(cd)$ is also a regular expression.

Rule 5: If R is a regular expression, then (R) is also a regular expression.

For example, let R denotes adc,

then (adc) is also a regular expression.

Rule 6: If R is a regular expression, then $R^*$ is also a regular expression.

For example, let R denotes ab,

then $(ab)^*$ is also a regular expression.

Rule 7: If R is a regular expression, then $R^+$ is also a regular expression.

For example, let R denotes pqr,

then $(pqr)^+$ is also a regular expression.

Rule 8: Any combination of the preceding rules is also a regular expression.

For example, let R1 denotes $(ab)^*$,

let R2 denotes $d^+$,

let R3 denotes $pq$,

then $((ab)^*/(d^+pq)^*)^*$ is also a regular expression.

## Strings Generated from a Regular Expression

A regular expression generates a set of strings.

1. A regular expression, $a^*$ means zero or more repretitions of a.

   So the strings generated from this regular expression are,

   $\varepsilon$,

   a,

   aa,

   aaa,

   aaaa, and so on.

2. A regular expression, $p^+$ means one or more repretitions of a.

   So the strings generated from this regular expression are,

   p,

   pp,

   ppp,

   pppp, and so on.

3. A regular expression, $a/b$ means the string can be either a or b.

So the strings generated from this regular expression are,

a,

b.

4. A regular expression, (a/b)c means a string that starts with either a or b and will be followed by c.

So the strings generated from this regular expression are,

ac,

bc

5. A regular expression, (a/b)c(d/e) means that a string will begin with either a or b, followed by c, and will end with either d or e.

So the strings generated from this regular expression are,

acd,

ace,

bcd,

bce.

Following shows some examples for the strings generated from regular expressions:

| Regular Expression | Strings Generated |
|---|---|
| $a$ | $a$ |
| $a/b/c$ | $a, b, c$ |
| $(ab)/(ba)$ | $ab, ba$ |
| $ab(d/e)$ | $abd, abe$ |
| $(aa)^*$ | $\varepsilon, aa, aaaa, aaaaaa, .....$ |
| $a^+$ | $a, aa, aaa, aaaa, .....$ |
| $(a/b)(c/d/e)u$ | $acu, adu, aeu, bcu, bdu, beu$ |
| $abc^*de$ | $abde, abcde, abccde, abcccde, .....$ |
| $(a/b)c^*$ | $a, ac, acc, accc, ....., b, bc, bcc, .....$ |
| $(abc)^*$ | $\varepsilon, abc, abcabc, abcabcabc, .....$ |
| $(0/(11))2((33)/1)$ | $0233, 021, 11233, 1121$ |
| $a^*a$ | $a, aa, aaa, aaaa, .....$ |
| $aa^*$ | $a, aa, aaa, aaaa, .....$ |
| $(abc)k^*$ | $abc, abck, abckk, abckkk, .....$ |
| $(ab)^+$ | $ab, abab, ababab, .....$ |

**Example 1:**

Write regular expression for the language,

L={aa, aaaa, aaaaaa, aaaaaaaa, .....}

Here aa repeats a number of times starting from 1. So the regular expression is,

$$(aa)^+$$

**Example 2:**

Write regular expression for the language,

L={0, 1, 2}

The regular expression is,

$$0/1/2$$

**Example 3:**

Write regular expression for the language, L over {0,1}, such that every string in L begins with 00 and ends with 11.

The regular expression is,

$$00(0/1)^*11.$$

**Example 4:**

Write regular expression for the language, L over {0,1}, such that every string in L contains alternating 0s and 1s.

The regular expression is,

$$(0(10)^*)/(0(10)^*1)/(1(01)^*)/(1(01)^*0)$$

**Example 5:**

Write regular expression for the language, L over {0,1}, such that every string in L ends with 11.

The regular expression is,

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

**Example 6:**

Write regular expression for the language, L over {a,b,c}, such that every string in L contains a substring ccc.

The regular expression is,

$$(a/b/c)^*ccc(a/b/c)^*$$

**Example 7:**

Write regular expression for the language, $L = \{a^n b^m | n \geq 4, m \leq 3\}$

This means that language contains the set of strings such that every string starts with at least 4 a's and end with at most 3 b's.

The regular expression is,

$$a^4a^*(\varepsilon/b/b^2/b^3)$$

**Example 8:**

Write regular expression for the language, $L = \{a^nb^m|$(n+m) is even $\}$

n+m can be even in two cases.

Case 1: Both n and m are even,

The regular expression is,

$$(aa)^*(bb)^*$$

Case 2: Both n and m are odd,

The regular expression is,

$$(aa)^*a(bb)^*b$$

Then the regular expression for L is,

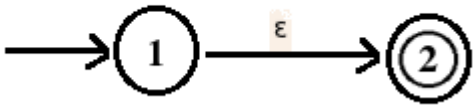
$$((aa)^*(bb)^*)/((aa)^*a(bb)^*b)$$

## 8 Transforming Regular Expressions to Finite Automata

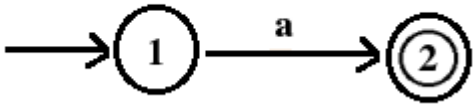
For every regular expression, a corresponding finite automata exist.

Here we will learn how to construct a finite automation corresponding to a regular expression.

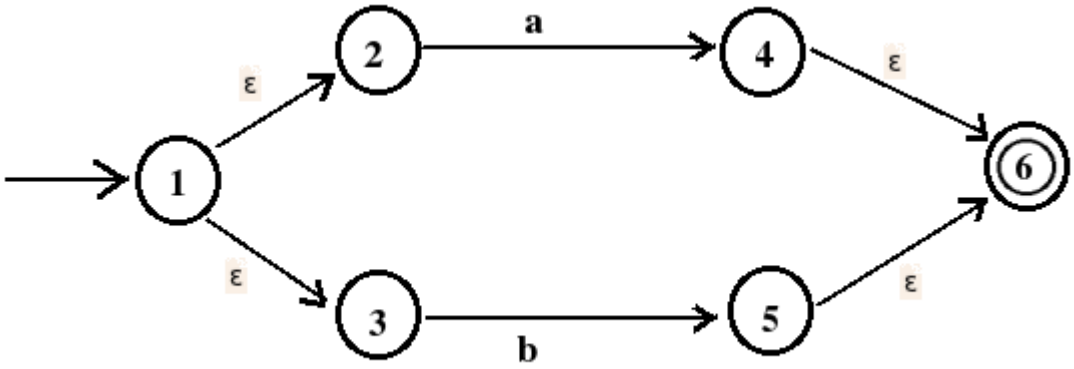
1. The NFA corresponding to regular expression, ε is



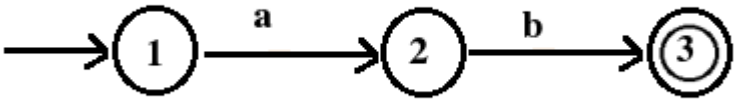

2. The NFA corresponding to regular expression, a is



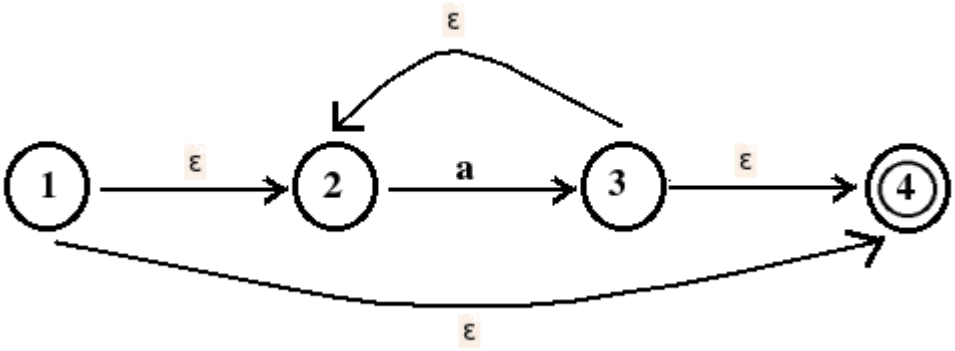

3. The NFA corresponding to regular expression, a | b is




4. The NFA corresponding to regular expression, ab is



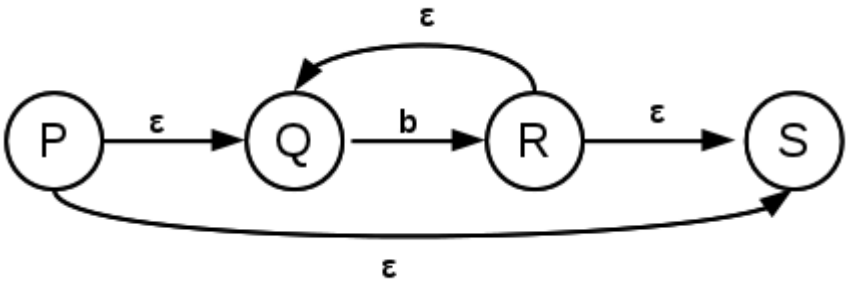

5. The NFA corresponding to regular expression, $a^*$ is



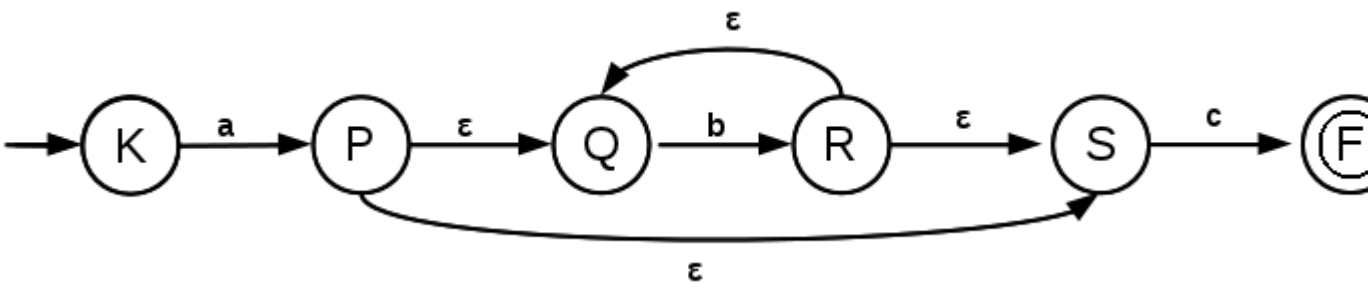

**Example 1:**

Construct a finite automation for the regular expression, $ab^*c$.

The finite automation corresponding to $b^*$ is



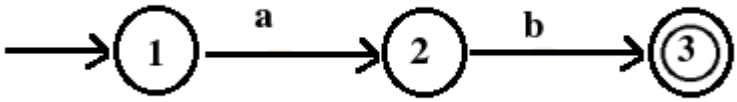

Then the finite automation corresponding to $ab^*c$ is



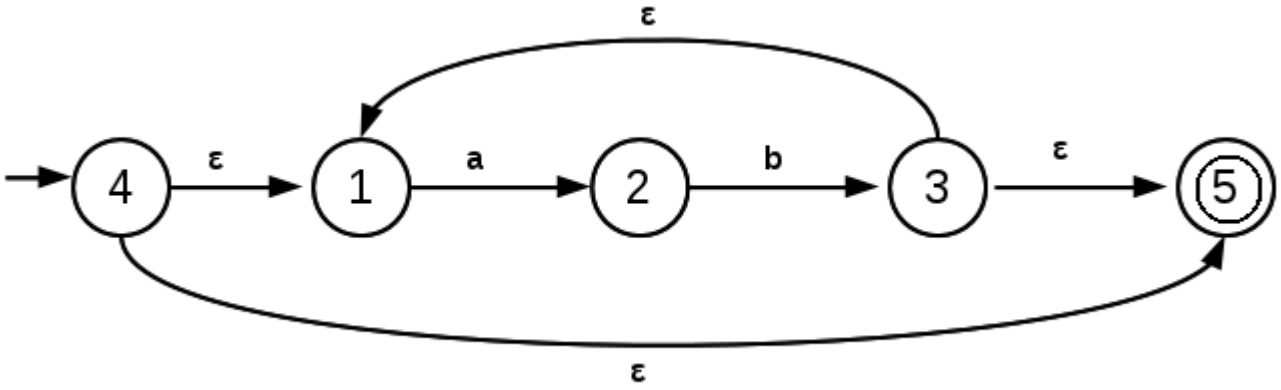

**Example 2:**

Construct a finite automation for the regular expression, $(ab)^*$.

The finite automation corresponding to $ab$ is



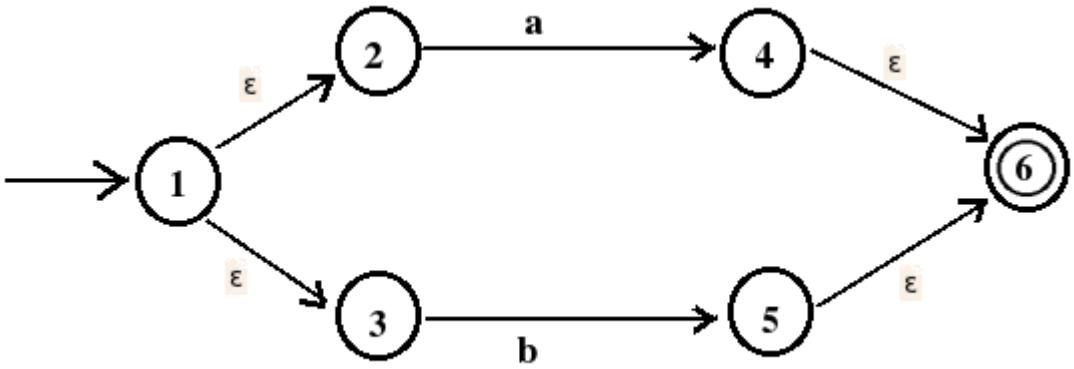

The finite automation corresponding to $(ab)^*$ is,



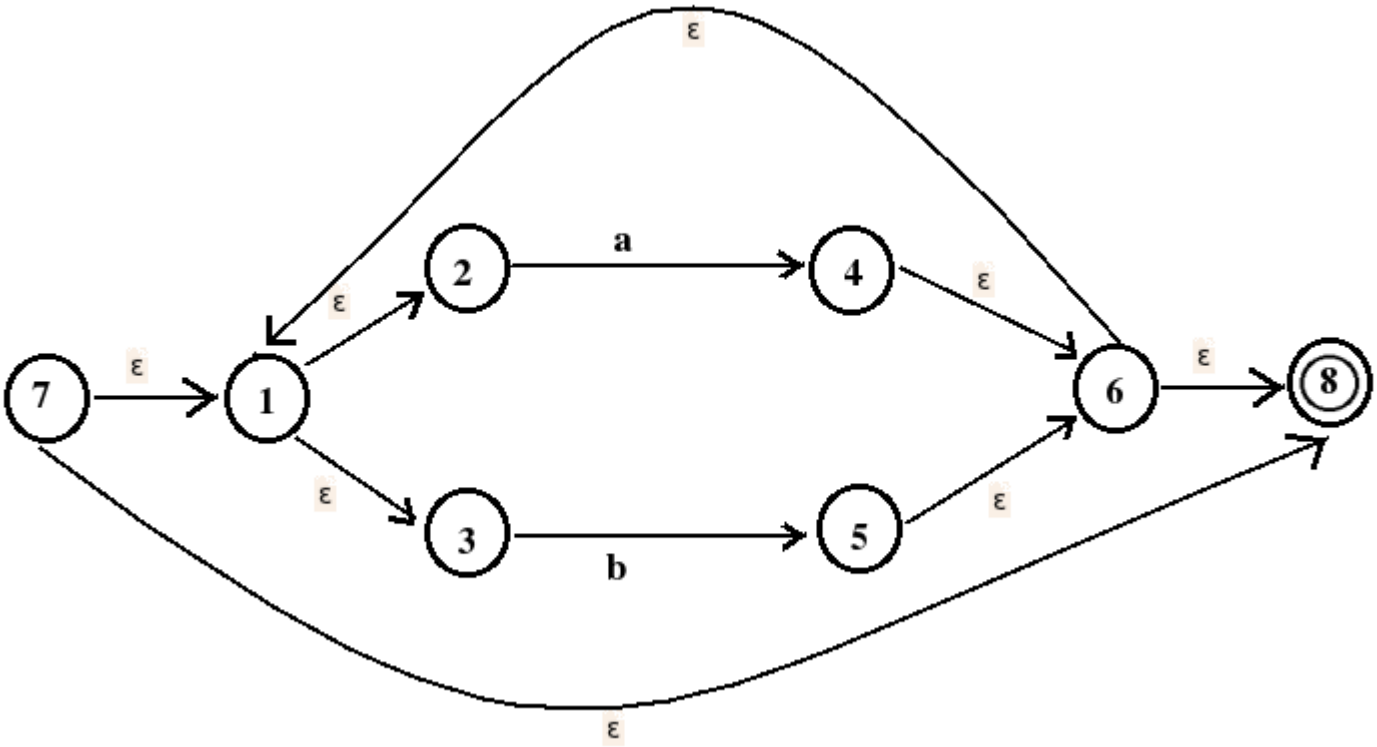

**Example 3:**

Find the NFA corresponding to regular expression $(a|b)^*a$.

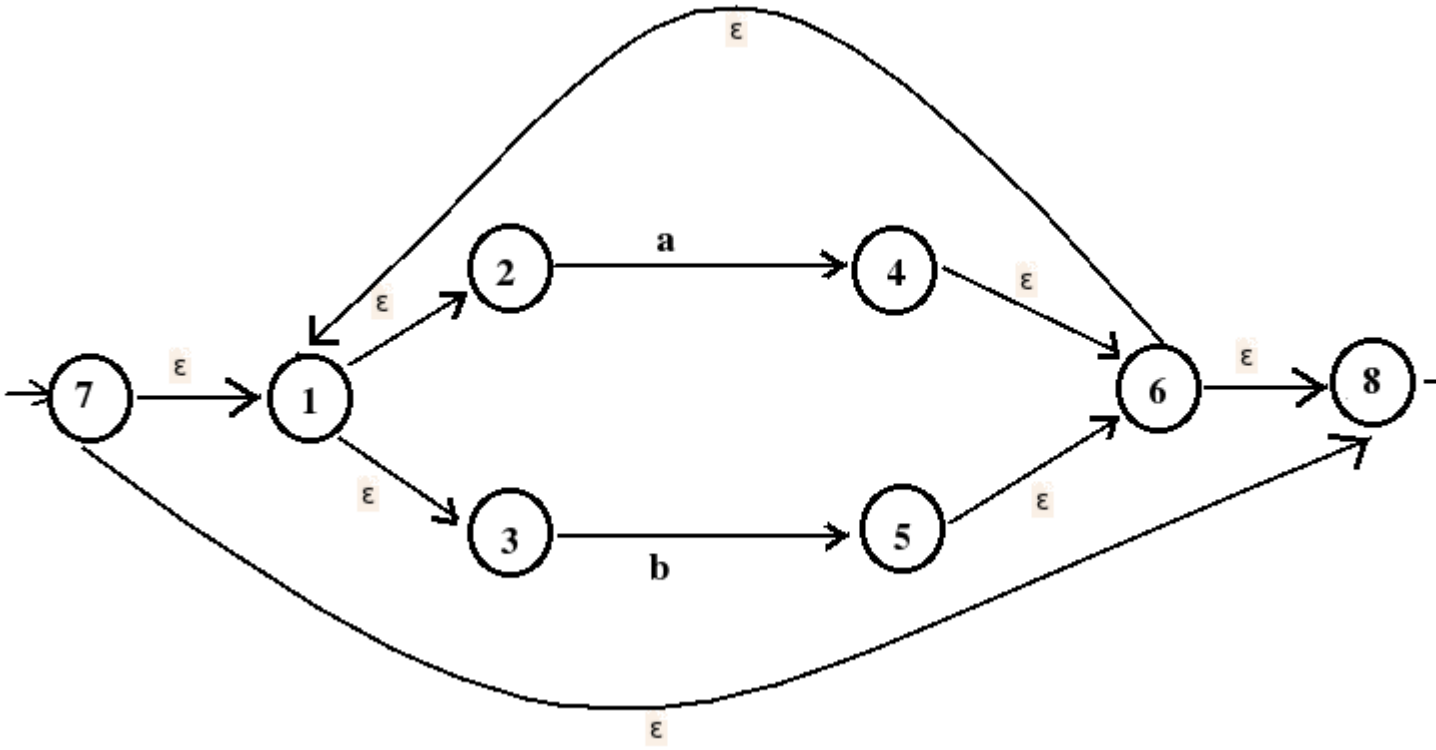
The NFA for a|b is



The NFA corresponding to $(a|b)^*$is




The NFA corresponding to $(a|b)^*a$ is




## 9 DFA to Regular Expression Conversion

Note: In this section, we use + symbol instead of / operator. For example, a/b is written as a+b.

We can find out the regular expression for every deterministic finite automata (DFA).

**Arden's Theorem**

This theorem is used to find the regular expression for the given DFA.

Let P and Q are two regular expressions,

then the equation, $R = Q + RP$ has the solution,

$R = QP^*$.

Proof:

$Q + RP = Q + (QP^*)P$,

since $R = QP^*$.

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

$= Q[\varepsilon/(P^*P)]$

$= QP^*$

$= R$



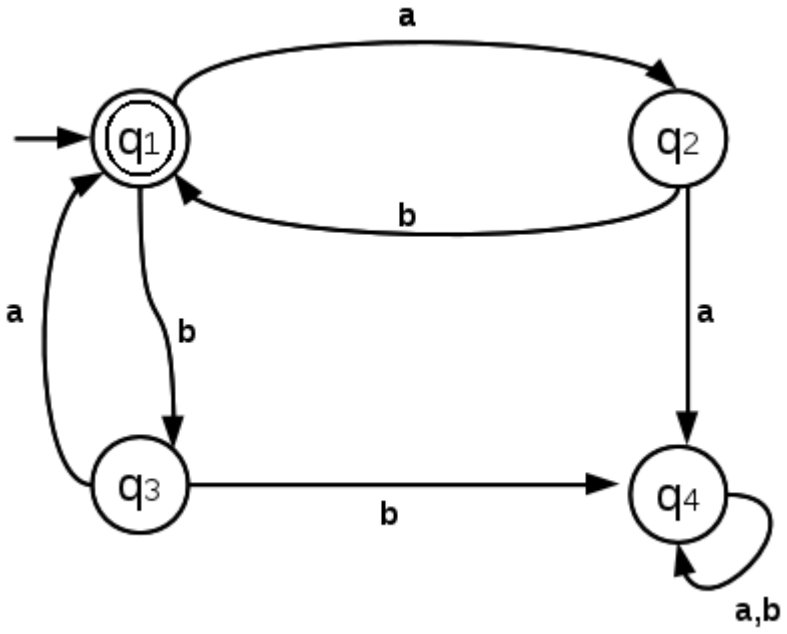
Thus if we are given, $R = Q + RP$ , we may write it as,

$R = QP^*$

The process is demonstrated using the following examples:

**Example 1:**

Find the regular expression corresponding to the following DFA,




Here we write equations for every state.

We write,

$q_1 = q_2b + q_3a + \varepsilon$

The term $q_2b$ because there is an arrow from $q_2$ to $q_1$on input symbol b.

The term $q_3a$ because there is an arrow from $q_3$ to $q_1$on input symbol a.

The term $\varepsilon$ because $q_1$is the start state.

$q_2 = q_1a$

The term $q_1a$ because there is an arrow from $q_1$ to $q_2$on input symbol a.

$q_3 = q_1b$

The term $q_1b$ because there is an arrow from $q_1$ to $q_3$on input symbol b.

$q_4 = q_2a + q_3b + q_4a + q_4b$

The term $q_2a$ because there is an arrow from $q_2$ to $q_4$on input symbol a.

The term $q_3b$ because there is an arrow from $q_3$ to $q_4$on input symbol b.

The term $q_4b$ because there is an arrow from $q_4$ to $q_4$on input symbol b.

The final state is $q_1$.

Putting $q_2$and $q_3$ in the first equation (corresponding to the final state), we get,

$q_1 = q_1ab + q_1ba + \varepsilon$

$q_1 = q_1(ab + ba) + \varepsilon$

$q_1 = \varepsilon + q_1(ab + ba)$

From Arden's theorem,

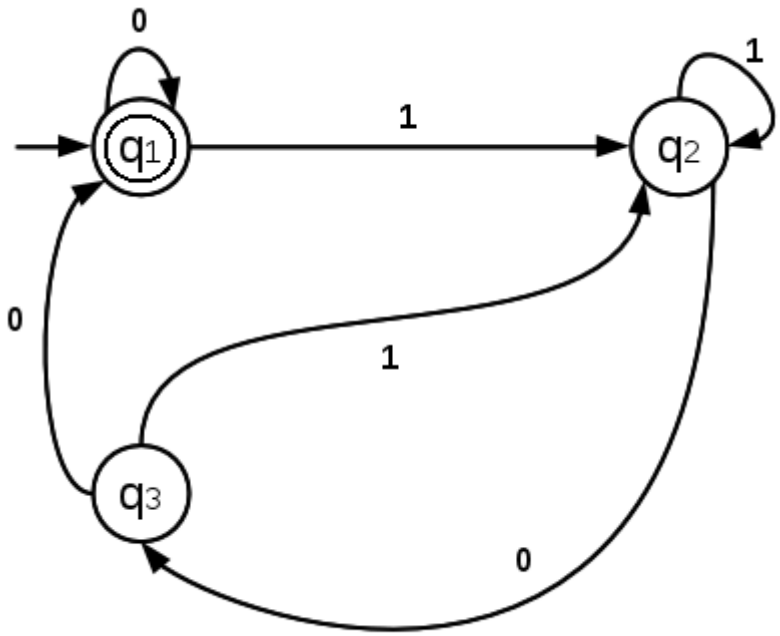
$q_1 = \varepsilon(ab + ba)^*$

$q_1 = (ab + ba)^*$

So the regular expression is, $((ab)/(ba))^*$

**Example 2;**

Find the regular expression corresponding to the following DFA,




Let us write the equations

$q_1 = q_10 + q_30 + \varepsilon$

$$q_2 = q_1 1 + q_2 1 + q_3 1$$

$$q_3 = q_2 0$$

$$q_2 = q_1 1 + q_2 1 + q_3 1$$

$$q_2 = q_1 1 + q_2 1 + (q_2 0) 1$$

$$q_2 = q_1 1 + q_2 (1 + 01)$$

$$q_2 = q_1 1 (1 + 01)^* \text{ (From Arden's theorem)}$$

Consider the equation corresponding to final state,

$$q_1 = q_1 0 + q_3 0 + \varepsilon$$

$$q_1 = q_1 0 + (q_2 0) 0 + \varepsilon$$

$$q_1 = q_1 0 + (q_1 1 (1 + 01)^* ) 0) 0 + \varepsilon$$

$$q_1 = q_1 (0 + 1(1 + 01)^* ) 00) + \varepsilon$$

$$q_1 = \varepsilon + q_1 (0 + 1(1 + 01)^* ) 00)$$

$$q_1 = \varepsilon (0 + 1(1 + 01)^* ) 00)^*$$

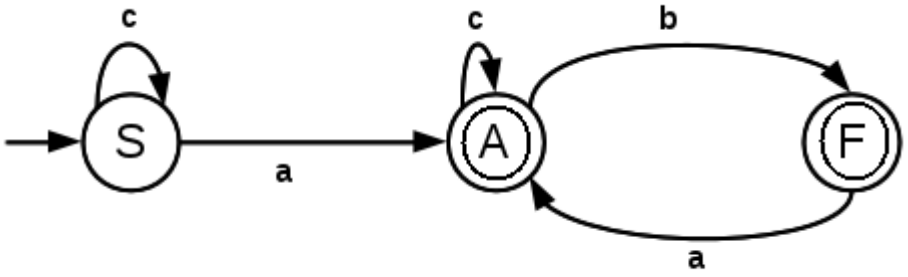
$$q_1 = (0 + 1(1 + 01)^* ) 00)^*$$

Since $q_1$ is a final state, the regular expression is,

$$(0/(1(1 + 01)^* ) 00))^*$$

**Example 3:**

Find the regular expression corresponding to the following DFA,




The equations are,

$$S = Sc + \varepsilon$$

$$A = Ac + Fa + Sa$$

$$F = Ab$$

$$S = Sc + \varepsilon$$

$$S = \varepsilon + Sc$$

$$S = \varepsilon c^*$$

$$S = c^*$$

$$A = Ac + Fa + Sa$$

$$A = Ac + Fa + c^*a$$

$$A = Ac + Aba + c^*a$$

$$A = A(c + ba) + c^*a$$

$$A = c^*a + A(c + ba)$$

$$A = c^*a(c + ba)^*$$

$$F = Ab$$

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

There are two final states in the DFA, they are A and F.

The regular expression corrresponding to A is $c^*a(c + ba)^*$

The regular expression corrresponding to F is $(c^*a(c + ba)^*)b$

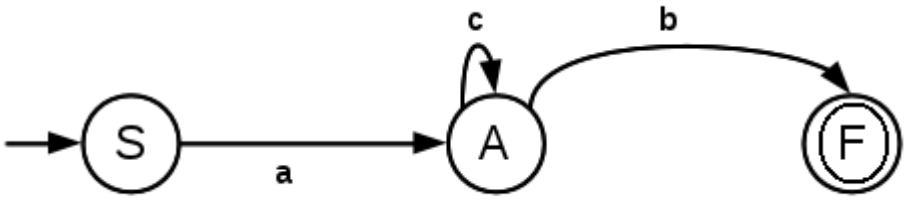
Then the regular expression for the DFA is

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

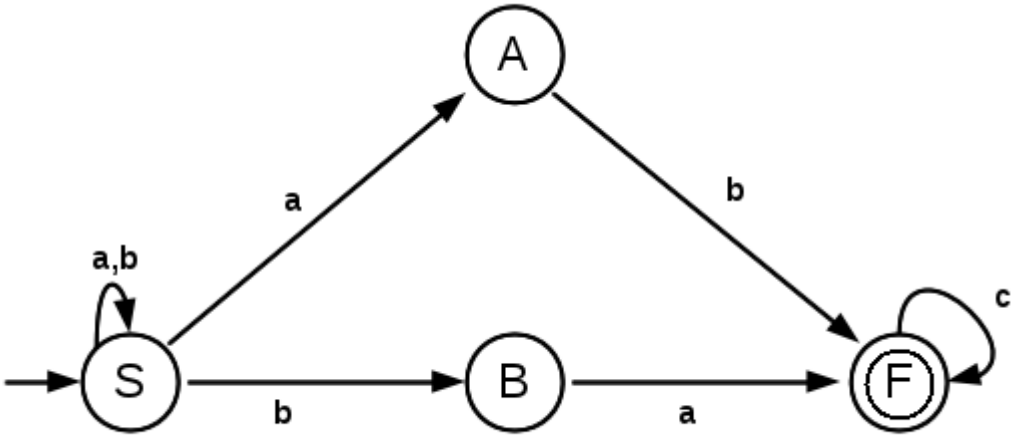
That is, $[c^*a(c/(ba))^*]/[(c^*a(c/(ba))^*)b]$

**Exercises:**

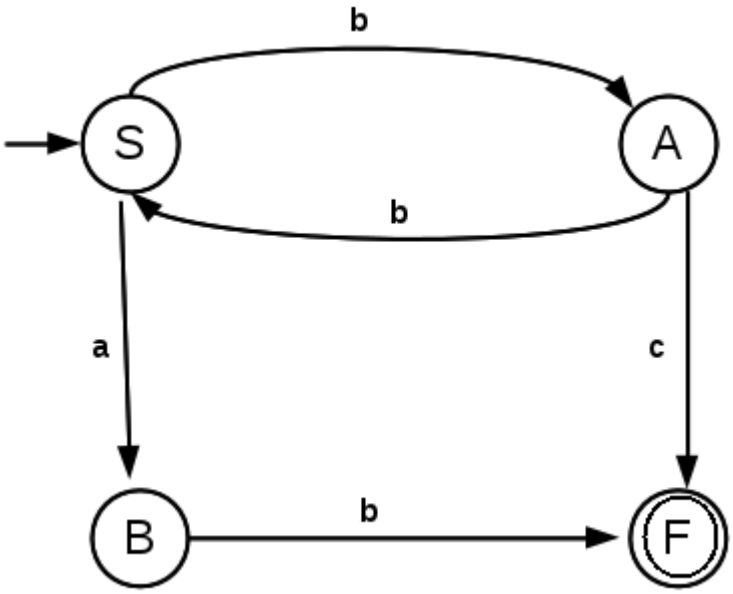
1. Find the regular expression corresponding to the following DFA,



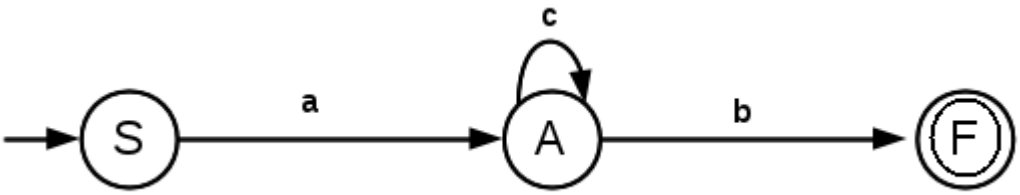

2. Find the regular expression corresponding to the following DFA,



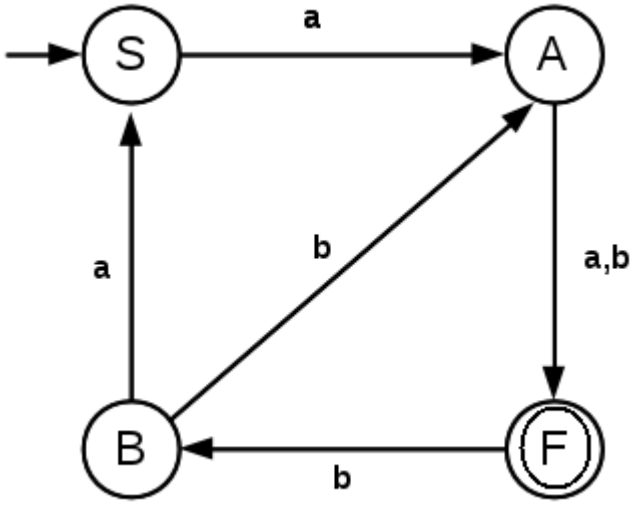
3. Find the regular expression corresponding to the following DFA,



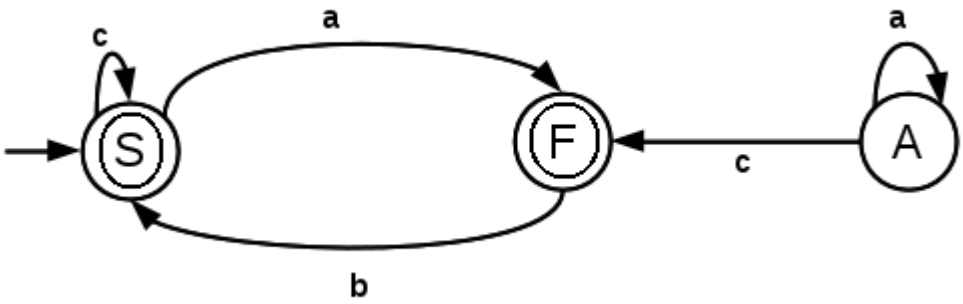

4. Find the regular expression corresponding to the following DFA,




5. Find the regular expression corresponding to the following DFA,

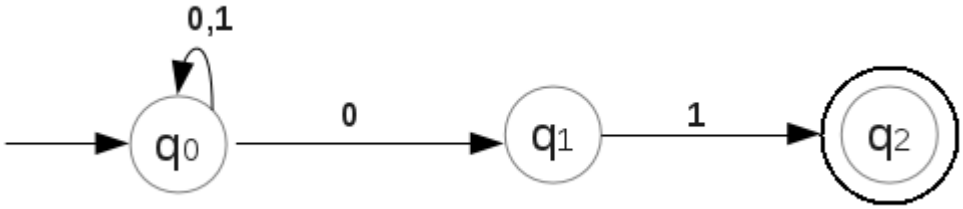



6. Find the regular expression corresponding to the following DFA,


c
a
a
S
F
A
c
b

# Part VI. Automata with Output

Consider the following finite automation,




Check whether the string 01101 is accepted by the above automation.

$\delta(q_0, \underline{0}1101) = q_0$

$\delta(q_0, 0\underline{1}101) = q_0$

$\delta(q_0, 01\underline{1}01) = q_0$

$\delta(q_0, 011\underline{0}1) = q_1$

$\delta(q_1, 0110\underline{1}) = q_2$

$q_2$ is a final state. So the string 01101 is accepted by the above NFA.

Here note the output we obtained from the automation. THe output is that the string 01101 is accepted by the automation.

Thus in our previous discussion on finite automata, there are only two possible outputs, ie, accept or reject.

Thus here the task done by the machine is simply recognize a language. But machines can do more than this.

So here we learn finite automata with output capabilities.

We will learn two models of finite automata with output capabilities. They are,

Moore Machine, and

Mealy Machine.



## 10 Moore Machine

Moore machine is a finite automation.

In this finite automation, output is associated with every state.

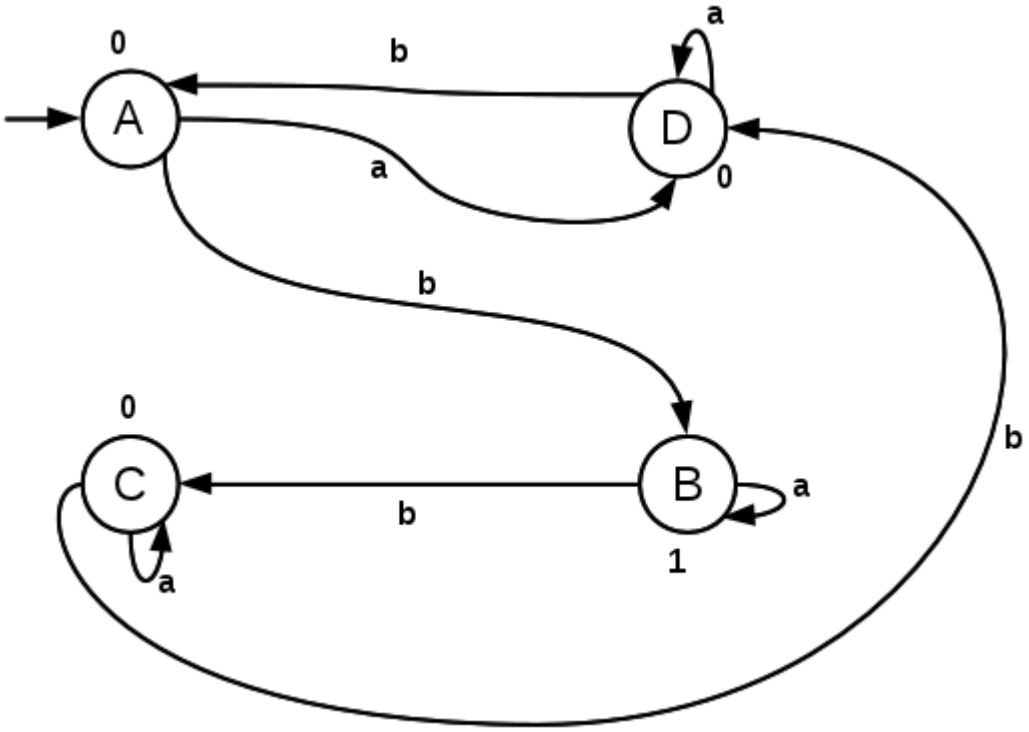
In Moore machine, every state has a fixed output.

Example,

The following is an example transition table for a Moore Machine.

| | Input Symbol | | |
|---|---|---|---|
| Current State | a | b | Output |
| ⟶A | D | B | 0 |
| B | B | C | 1 |
| C | C | D | 0 |
| D | D | A | 0 |

The transition diagram corresponding to this is ,




A is the start state.

There is no final state in a Moore machine.

Output is shown above every state.

## Definition of a Moore Machine

Moore Machine is a six tuple machine and is defined as,

$$M_0\,(\,Q,\,\sum,\,\Delta,\,\delta,\,\lambda',\,q_0\,)$$

where $M_0$ is the Moore Machine,

$Q$ is a finite set of states,

$\sum$ is a set of input symbols,

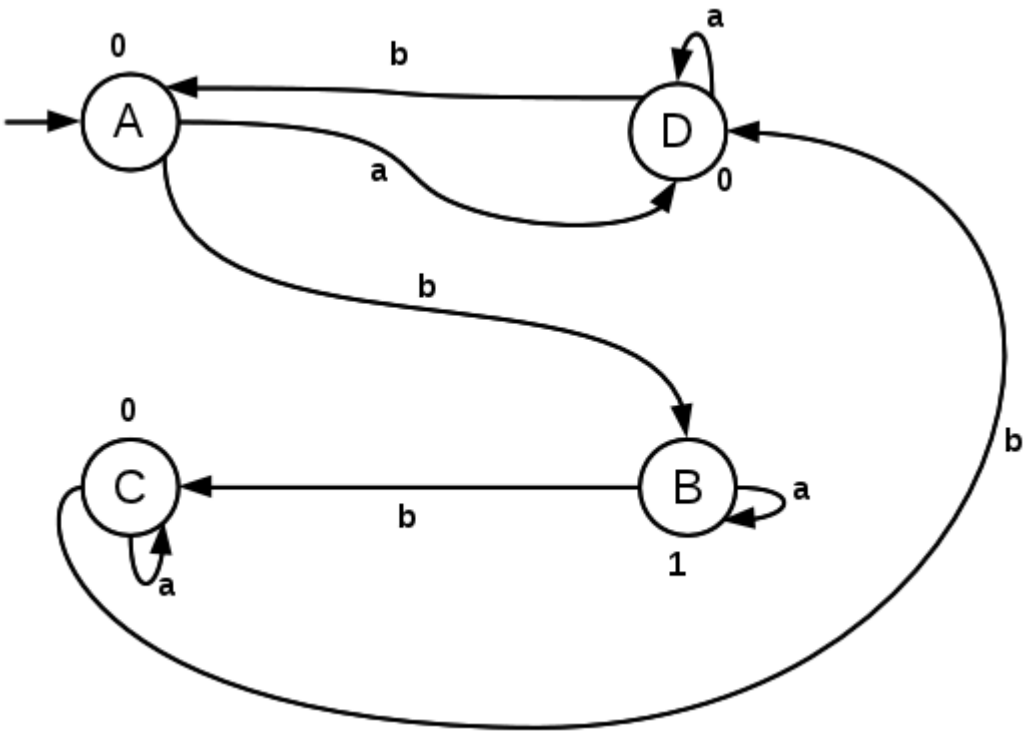
$\Delta$ is set of outputs,

$\delta$ is the set of transition functions,

$\lambda'$ is the mapping function which maps a state to an output,

$q_0$ is the start state.

Consider the following Moore Machine,




In the above Moore Machine,

The set of states,

$Q = \{A, B, C, D\}$

The set of input symbols,

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

The set of outputs,

$\Delta = \{0, 1\}$

The set of transitions,

$\delta(A, a) = D$

$\delta(B, b) = C$

$\delta(C, a) = C$ and so on.

$\lambda'(A) = 0$

$\lambda'(B) = 1$

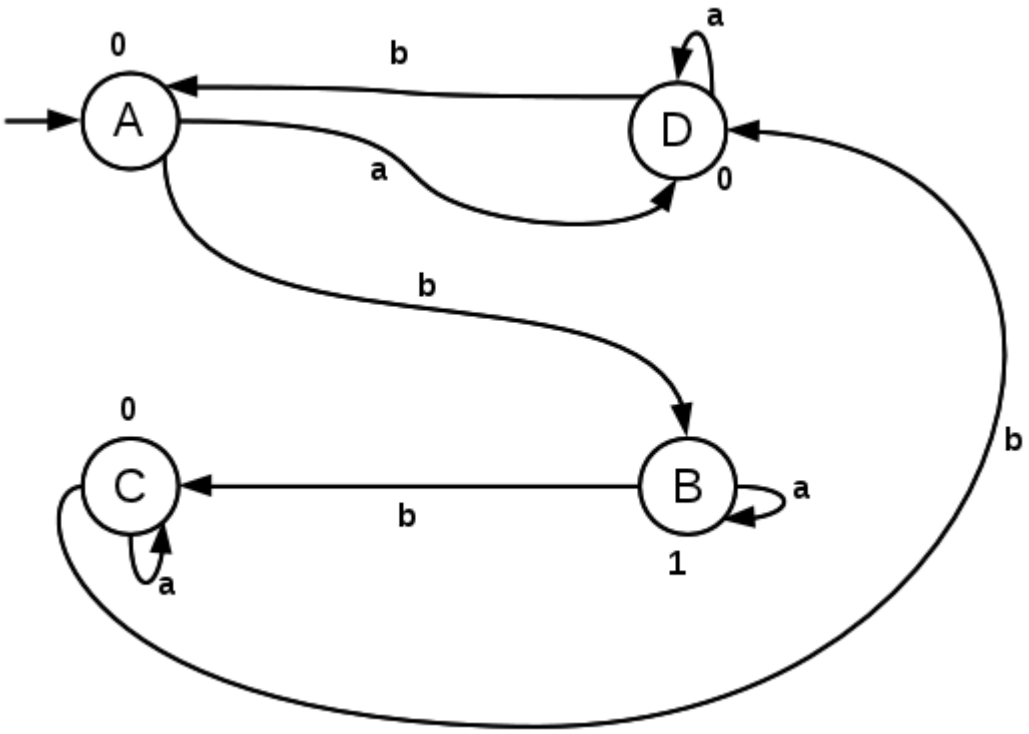
$\lambda'(C) = 0$

$\lambda'(D) = 0$

## String Processing through Moore Machine

**Example 1:**

Consider the Moore Machine given below:




Process the string $abbb$ using the Moore Machine and find the output string.

Begin from the start symbol, A,

$$\delta(A, \underline{a}bbb) = D \qquad \lambda'(A) = 0$$
$$\delta(D, a\underline{b}bb) = A \qquad \lambda'(D) = 0$$
$$\delta(A, ab\underline{b}b) = B \qquad \lambda'(A) = 0$$
$$\delta(B, abb\underline{b}) = C \qquad \lambda'(B) = 1$$
$$\lambda'(C) = 0$$

The output string we got is 00010.

**Example 2:**

Consider the following Moore Machine,

| Current State | Input Symbol | | Output |
|---|---|---|---|
| | a | b | |
| ⟶A | B | C | 0 |
| B | C | D | 0 |
| C | D | E | 0 |
| D | E | B | 0 |
| E | B | C | 1 |

Process the string aabbba through the Moore Machine and find the output.

Note that here transition table corresponding to the Moore Machine is given.

Begin from the start symbol, A,

$$\delta(A, \underline{a}abbba) = B \qquad \lambda'(A) = 0$$
$$\delta(B, a\underline{a}bbba) = C \qquad \lambda'(B) = 0$$
$$\delta(C, aa\underline{b}bba) = E \qquad \lambda'(C) = 0$$
$$\delta(E, aab\underline{b}ba) = C \qquad \lambda'(E) = 1$$
$$\delta(C, aabb\underline{b}a) = E \qquad \lambda'(C) = 0$$
$$\delta(E, aabbb\underline{a}) = B \qquad \lambda'(E) = 1$$
$$\lambda'(B) = 0$$

The output string we got is 0001010.



## 11 Mealy Machine

Mealy machine is a finite automation.

In this finite automation, output is associated with every transition.

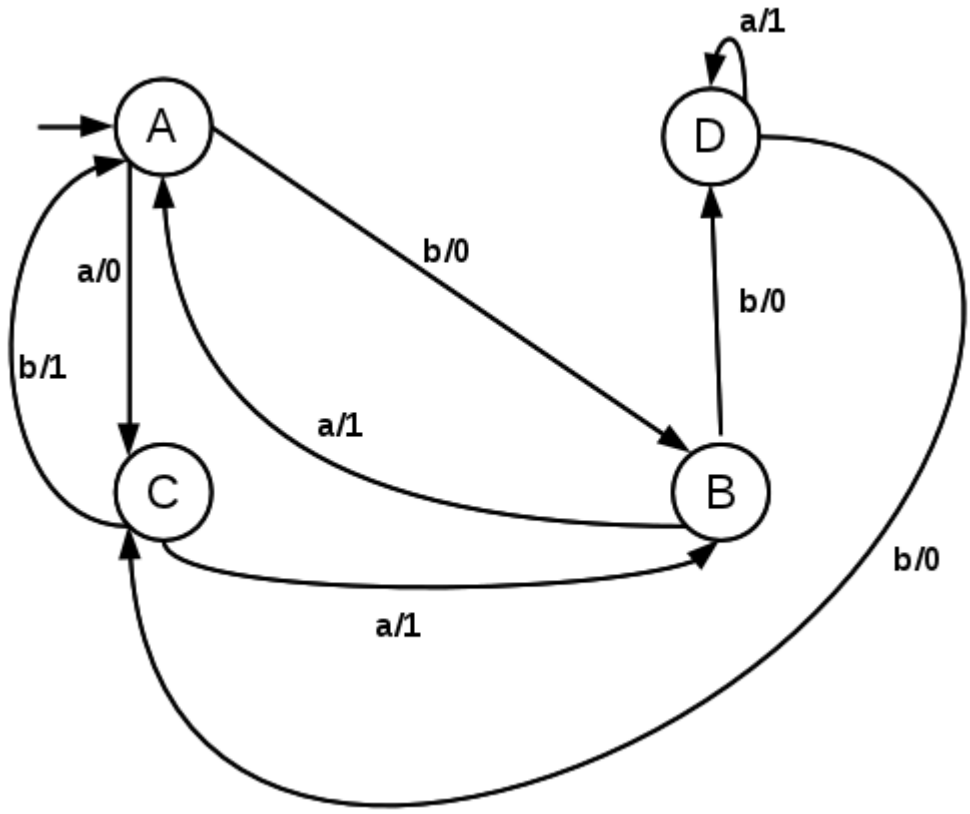
In Mealy machine, every transition for a particular input symbol has a fixed output.

Example,

The following is an example transition table for a Mealy Machine.

| | For | Input = a | For | Input=b |
|---|---|---|---|---|
| Current State | State | Output | State | Output |
| ⟶A | C | 0 | B | 0 |
| B | A | 1 | D | 0 |
| C | B | 1 | A | 1 |
| D | D | 1 | C | 0 |

The transition diagram corresponding to this Mealy Machine is




A is the start state.

There is no final state in a Mealy machine.

Output is shown with every transition. (a/1 means a is the input symbol and output is 1).

## Definition of a Mealy Machine

Mealy Machine is a six tuple machine and is defined as,

$$M_e\,(\,Q,\,\sum,\,\Delta,\,\delta,\,\lambda',\,q_0\,)$$

where $M_e$ is the Mealy Machine,

$Q$ is a finite set of states,

$\sum$ is a set of input symbols,

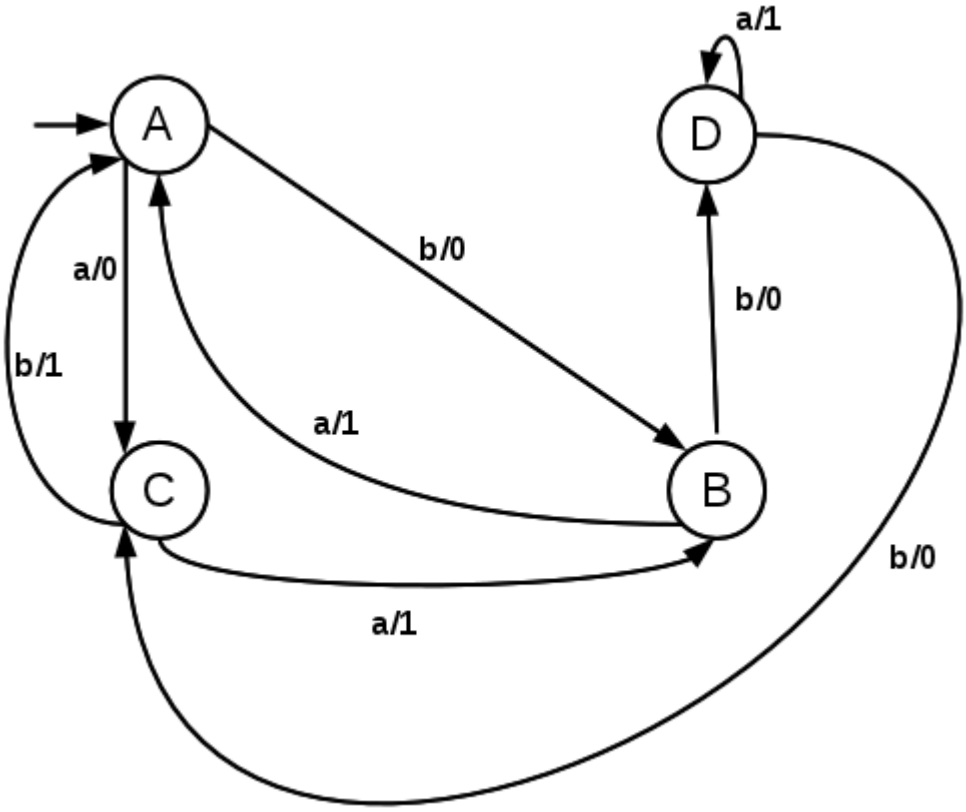
$\Delta$ is set of outputs,

$\delta$ is the set of transition functions,

$\lambda'$ is the mapping function which maps a state and an input symbol to an output,

$q_0$ is the start state.

Consider the following Mealy Machine,




In the above Moore Machine,

The set of states,

$Q = \{A, B, C, D\}$

The set of input symbols,

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

The set of outputs,

$\Delta = \{0, 1\}$

The set of transitions,

$\delta(A, a) = C$

$\delta(B, b) = D$

$\delta(C, a) = B$ and so on.

$\lambda'(A, a) = 0$

$\lambda'(B, a) = 1$

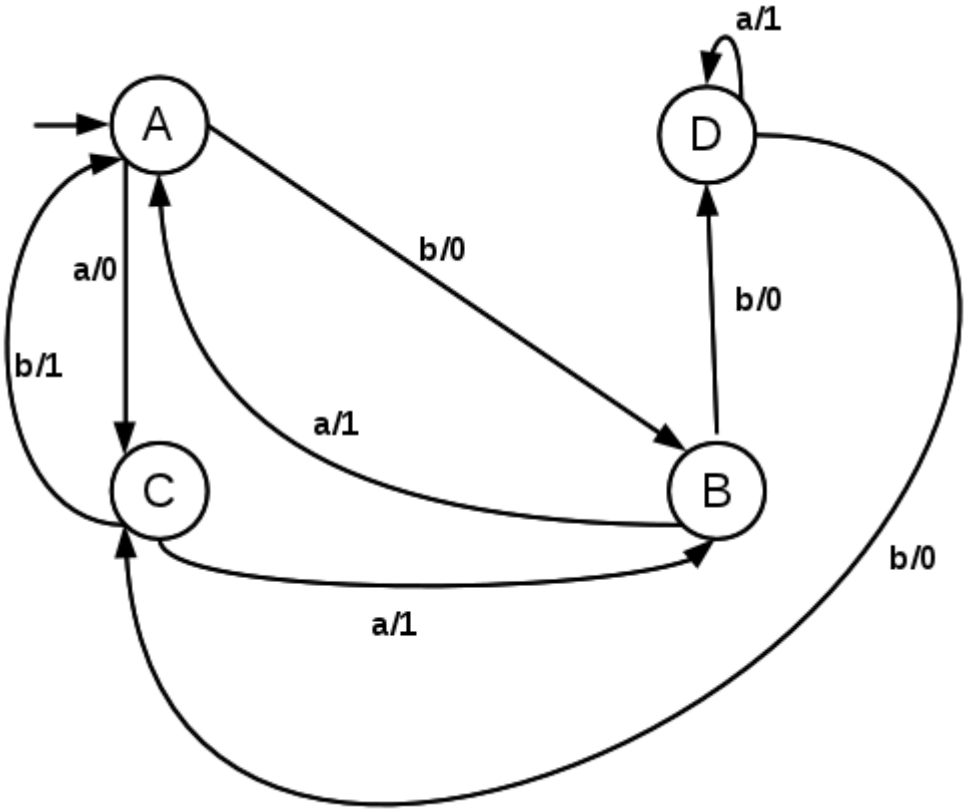
$\lambda'(C, a) = 1$

$\lambda'(D, b) = 0$ and so on.

### String Processing through Mealy Machine

Example 1:

Consider the Mealy Machine given below:




Process the string abbb through the above Mealy Machine and find out the output.

Begin from the start symbol, A,

$\delta(A, \underline{a}bbb) = C$ $\lambda'(A, a) = 0$

$\delta(C, a\underline{b}bb) = A$ $\lambda'(C, b) = 1$

$\delta(A, ab\underline{b}b) = B$ $\lambda'(A, b) = 0$

$\delta(B, abb\underline{b}) = D$ $\lambda'(B, b) = 0$

The output string we got is 0100.

## 12 Moore Machine to Mealy Machine Conversion

**Example 1:**

Consider the Moore Machine given below:

| Current State | Input Symbol | | Output |
|---|---|---|---|
| | a | b | |
| ⟶A | D | C | 0 |
| B | A | B | 1 |
| C | C | A | 1 |
| D | C | B | 0 |

Convert this Moore Machine to a Mealy Machine.

If we look at the above transition table, we can see that the machine has output 0 for the states A and D.

The machine has output 1 for the states B and C.

To convert this to a Mealy Machine, the output is taken as 0, whenever there is a transition to A or D.

Also, the output is taken as 1, whenver there is a transition to B or C.

The Mealy Machine is as follows:

| | For | Input = a | For | Input=b |
|---|---|---|---|---|
| Current State | State | Output | State | Output |
| $\longrightarrow$A | D | 0 | C | 1 |
| B | A | 0 | B | 1 |
| C | C | 1 | A | 0 |
| D | C | 1 | B | 1 |

**Example 2:**

Convert the Moore Machine given below to a Mealy Machine:

| | Input | Symbol | |
|---|---|---|---|
| Current State | a | b | Output |
| $\longrightarrow$A | B | C | 0 |
| B | A | C | 1 |
| C | C | B | 1 |

If we look at the above transition table, we can see that the machine has output 0 for the state A.

The machine has output 1 for the states B and C.

To convert this to a Mealy Machine, the output is taken as 0, whenever there is a transition to A.

Also, the output is taken as 1, whenver there is a transition to B or C.

The Mealy Machine is as follows:

| | For | Input = a | For | Input=b |
|---|---|---|---|---|
| Current State | State | Output | State | Output |
| $\longrightarrow$A | B | 1 | C | 1 |
| B | A | 0 | C | 1 |
| C | C | 1 | B | 1 |

# 13 Mealy Machine to Moore Machine Conversion



**Example 1:**

Consider the Mealy Machine given below:

| Current State | For Input = a | | For Input=b | |
|---|---|---|---|---|
| | State | Output | State | Output |
| ⟶A | B | 1 | C | 1 |
| B | C | 0 | A | 1 |
| C | A | 1 | B | 0 |

Conver this to a Moore Machine.

Here, State A has output 1 at two places.

State B has output 0 at one place and 1 at another place. So this state B is decomposed into two states, B0 and B1. In the new Moore Machine, state B0 has output 0 and B1 has output 1.

State C has output 0 at one place and output 1 at another place. So this state is decomposed into two states, C0 and C1. In the new Moore Machine, state C0 has output 0 and C1 has output 1.

The Moore Machine is given below;

| Current State | Input Symbol | | Output |
|---|---|---|---|
| | a | b | |
| ⟶A | B1 | C1 | 1 |
| B0 | C0 | A | 0 |
| B1 | C0 | A | 1 |
| C0 | A | B0 | 0 |
| C1 | A | B0 | 1 |

**Example 2:**

Convert the following Mealy Machine to Moore Machine.

| Current State | For Input = a | | For Input=b | |
|---|---|---|---|---|
| | State | Output | State | Output |
| ⟶A | B | 1 | C | 1 |
| B | C | 0 | A | 1 |
| C | A | 1 | D | 0 |
| D | D | 0 | B | 1 |

Here, State A has output 1 at two places.

State B has output 1 at two places.

State C has output 0 at one place and output 1 at another place. So this state is decomposed into two states, C0 and C1. In the new Moore Machine, state C0 has output 0 and C1 has output 1.

State D has output 0 at both places.

The Moore Machine is given below;

| Current State | Input Symbol a | Input Symbol b | Output |
|---|---|---|---|
| $\longrightarrow$A | B | C1 | 1 |
| B | C0 | A | 1 |
| C0 | A | D | 0 |
| C1 | A | D | 1 |
| D | D | B | 0 |

# Part VII. Applications of Finite Automata

Finite automata has several applications in many areas such as

compiler design,

special purpose hardware design,

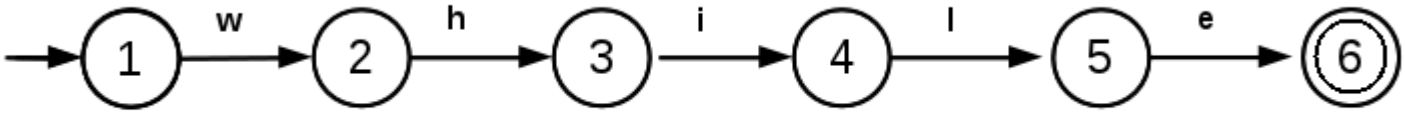
protocol specification etc..

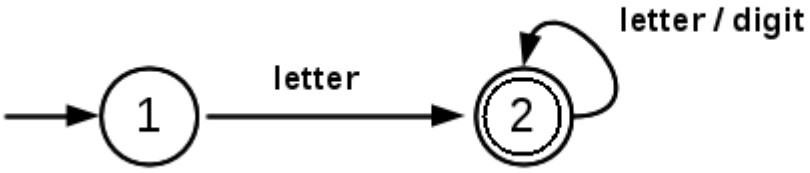
Some of the applications are explained below:

**1. Compiler Design**

Lexical analysis or scanning is an important phase of a compiler. In lexical analysis, a program such as a C program is scanned and the different tokens(constructs such as variables, keywords, numbers) in the program are identified. A DFA is used for this operation.

For example, finite automation to recognize the tokens, 'while' keyword and variables are shown below:



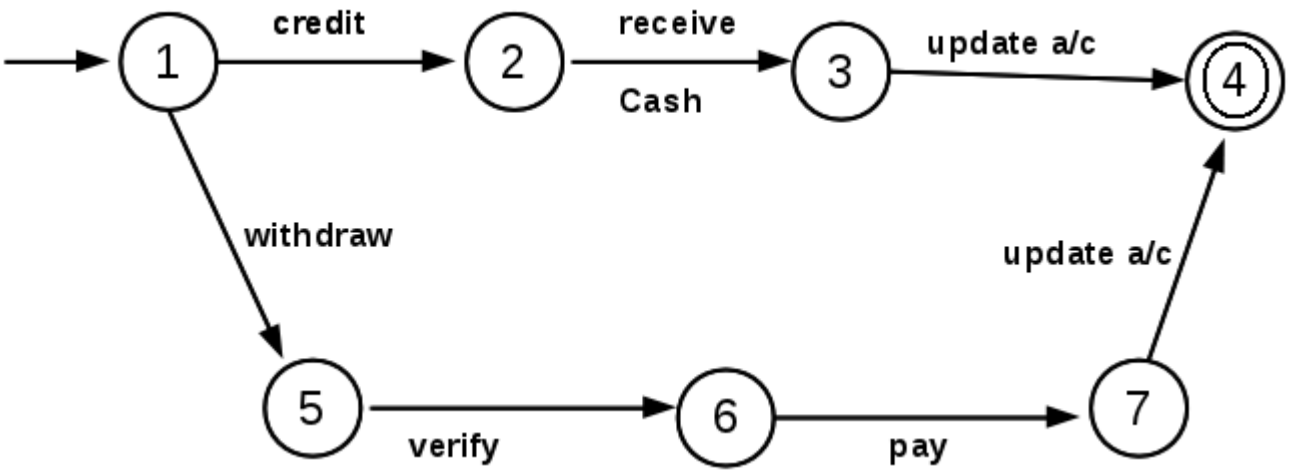

The well known lexical analyser tool, Lex uses DFA to perform lexical analysis.

**2. Hardware Design**

In the design of computers, finite automation is used to design control unit of a computer. A typical sequence of operations of a computer consists of a repretition of instructions and every instruction involves the actions fetch, decode, fetch the operand, and execute. Every state of a finite automation represents a specific stage of instruction cycle.

**3. Protocol Specification**

A system is composed of an interconnected set of subsystems. A protocol is a set of rules for proper coordination of activities of a system. Such a protocol can be described using finite automations. An example of a bank is ahown below:




State 1:

State 1 represents the entry of a customer.

State 2:

After he wishes to credit some cash to his account, system is in state 2.

State 3:

Then the cashier receives cash. It reaches state 3.

State 4:

Then the cashier updates the acoount and reach the final state 4.

State 5:

If the customer wants to withdraw cash, he submits the slip ans system reaches state 5.

State 6:

Then the slip is verified to confirm that he has sufficient balance and reaches state 6.

State 7:

Then the customer is paid cash and reaches state 7.

State 8:

Then the customer account is updated and system reaches final state 4 of the transaction.

# Part VIII. Pumping Lemma for Regular Languages

## Regular Languages

Regular languages are those classes of languages accepted by DFA's and by NFAs. These languages can be defined using regular expressions.

Not every language is regular.

For example,

Language defined by, $L = \{a^n b^n \ for \ n = 0, 1, 2, 3, ....\}$ is not regular.

Language defined by, $L = \{a^n ba^n \ for \ n = 0, 1, 2, 3, ....\}$ is not regular.

Language defined by, $L = \{a^n b^n aab^{n+1} \ for \ n = 1, 2, 3, ....\}$ is not regular.

A powerful technique, known as Pumping Lemma can be used to show that certain languages are not regular.

## Pumping Lemma for Regular Languages



## Definition

Let L be a regular language. Then, there exists a constant n such that

for every string w in L and $|w| \geq n$,

$w$ can be broken into three parts, x , y and z such that

$w = xyz$, $y \notin \varepsilon$, and $|xy| \leq n$.

Then for all $i \geq 0$, the string $xy^i z$ also belongs to L.

That is, we always find a non-empty string y not too far from the beginning of w that can be "pumped"; that is, repeating y any number of times, keeps the resulting string in the language.

#### A Simple Definition

Let L be a language and $w$ be a string in the language. Break $w$ into three parts. Then write, $w = xyz$.

Then $xy^i z$ must also be a string in the language. That is, strings $xy^0 z$, $xy^1 z$ , $xy^2 z$ , $xy^3 z$ , $xy^4 z$ ..... must be in the language.

Then L is a context free language.

For example,

#### Example 1:

Consider the language $L = \{a^n | n \geq 1\}$. This language is regular.

The set of strings in this language are,

{a, aa, aaa, aaaa, aaaaa, .....}

Consider one string in this language, aaaa.

Let w=aaaa.

Break w into three parts, x, y and z.

Let w=aaaa=xyz

That is

w =

| a | aa | a |
|---|---|---|
| x | y | z |

Then,

$xy^iz$ is

Let i=1,

we have, $xy^1z$

w =

| a | aa | a |
|---|---|---|
| $x$ | $y^1$ | $z$ |

$xy^1z$=aaaa =$a^4$ is in the above language.

Let i=2,

we have, $xy^2z$

w =

| a | aaaa | a |
|---|---|---|
| $x$ | $y^2$ | $z$ |

$xy^2z$=aaaaaa =$a^6$ is in the above language.

Let i=3,

we have, $xy^3z$

w =

| a | aaaaaa | a |
|---|---|---|
| $x$ | $y^3$ | $z$ |

$xy^3z$=aaaaaaaa =$a^8$ is in the above language and so on.

Since all the strings of the form $xy^iz$ , are in the above language, the language $a^n$ is a regular language.

**Example 2:**

Consider the language $L = \{a^nb^m | n, m \geq 1\}$. This language is regular.

The set of strings in this language are,

{ab, abb, abbb, aab, aabb,aaaabbbbbbbb .....}

Consider one string in this language, aaabb.

Let w=aaabb.

Break w into three parts, x, y and z.

Let w=aaabb=xyz

That is

w =

| a | aa | bb |
|---|---|---|
| x | y | z |

Then,

$xy^iz$ is

Let i=1,

we have, $xy^1z$

w =

| a | aa | bb |
|---|---|---|
| $x$ | $y^1$ | $z$ |

$xy^1z$=aaabb =$a^3b^2$ is in the above language.

Let i=2,

we have, $xy^2z$

w = $\begin{array}{|c|c|c|}\hline \text{a} & \text{aaaa} & \text{bb} \\ \hline x & y^2 & z \\ \hline \end{array}$ $xy^2z$=aaaaabb =$a^5b^2$ is in the above language and so on

Since all the strings of the form $xy^iz$ , are in the above language, the language $a^nb^m$ is regular.

**Example 3:**

Consider the language $L = \{a^nb^n | n \geq 1\}$. This language is not regular.

The set of strings in this language are,

{ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, .....}

Consider one string in this language, aaabbb.

Let w=aaabbb.

Break w into three parts, x, y and z.

Let w=aaabbb=xyz

That is

w = $\begin{array}{|c|c|c|}\hline \text{a} & \text{aa} & \text{bbb} \\ \hline \text{x} & \text{y} & \text{z} \\ \hline \end{array}$

Then,

$xy^iz$ is

Let i=1,

we have, $xy^1z$

w = $\begin{array}{|c|c|c|}\hline \text{a} & \text{aa} & \text{bbb} \\ \hline x & y^1 & z \\ \hline \end{array}$ $xy^1z$=aaabbb =$a^3b^3$ is in the above language.

Let i=2,

we have, $xy^2z$

w = $\begin{array}{|c|c|c|}\hline \text{a} & \text{aaaa} & \text{bbb} \\ \hline x & y^2 & z \\ \hline \end{array}$ $xy^2z$=aaaaabbb =$a^5b^3$ is not in the above language.

Since some of the strings of the form $xy^iz$ , are not in the above language, the language $a^nb^n$ is not regular.

## Proof

Let L is regular. Then a DFA exists for L.

Let that DFA has n states.

Consider a string w of length n or more, let $w = a_1, a_2, a_3.....a_m$, where m≥n and each $a_i$ is an input symbol

By the Pigeonhole principle, it is not possible for the n+1 different states ($P_i$'s) for i=0,1,2,3....n to be distinct, since there are only n states.

Thus we can find two different integers i and j such that $P_i = P_j$.

Now we can break $w = xyz$ as follows:

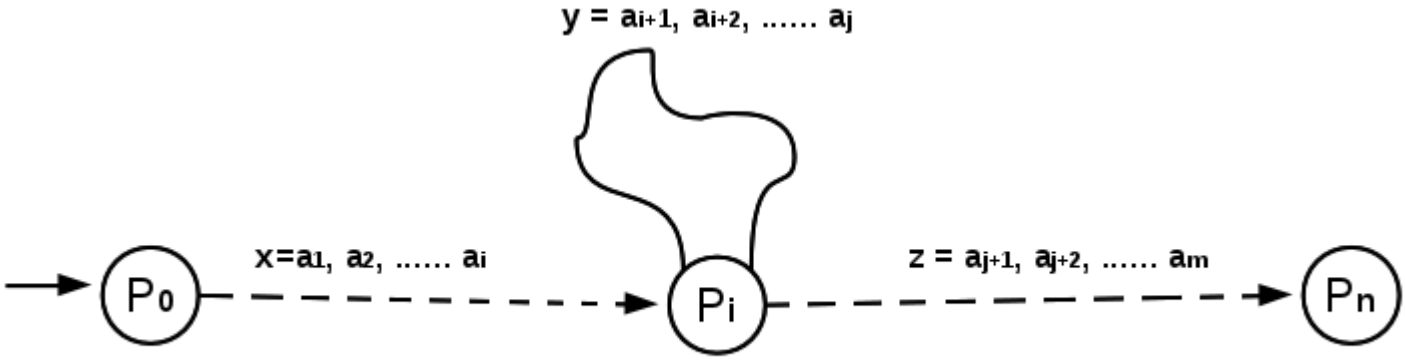
1. $x = a_1a_2, .....a_i$
2. $y = a_{i+1}a_{i+2}.....a_j$
3. $z = a_{j+1}a_{j+2}.....a_m$

That is x takes us to $P_i$ once,

y takes us from $P_i$ back to $P_i$ (since $P_i$ is also $P_j$), and

z is balance of w.

This is shown in the following diagram.




Note that x may be empty, when i=0.

Also, z may be empty, if j = n = m.

y cannot be empty, since i is strictly less than j.

Suppose the above automation receives $xy^iz$ for an $i \geq 0$.

If i = 0, then the string is $xz$,

automation goes from the start state $P_0$ to $P_i$ on input x.

Since $P_i$ is also $P_j$, it must be that M goes from $P_i$ to the accepting state on input z.

Thus $xz$ is accepted by the automation.

That is,

$\hat{\delta}(P_0, x) = \hat{\delta}(P_i, x) = P_i$

$\hat{\delta}(P_i, z) = P_n$

If i>0, then

automation goes from $P_0$ to $P_i$ on input string x,

circles from $P_i$ to $P_i$, i times on input $y_i$, and then

goes to the accepting state on input z.

Thus $xy^iz$ is accepted by the automation.

That is,

$\hat{\delta}(P_0, x) = P_i$

$\hat{\delta}(P_i, y) = P_i$

$\hat{\delta}(P_i, y^i) = P_i$

$\hat{\delta}(P_i, z) = P_n$

Pumping lemma is useful for proving that certain languages are not regular.

## Proving Non-regularity of certain Languages using Pumping Lemma

To prove that certain languages are not regular, following are the steps:

Step 1:

Assume that L is regular. Let n be the number of steps in the corresponding finite automation.

Step 2:

Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.

Step 3:

Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

The important part of proof is Step 3. There, we need to find i such that $xy^iz \notin L$.

In some cases, we prove $xy^iz \notin L$ by considering $|xy^iz|$.

In some cases, we may have to use the structure of strings in L.

**Example 1:**

Prove that language, $L = \{0^i1^i \text{ for } i \geq 1\}$ is not regular.

**Step 1: Assume that L is regular. Let n be the number of steps in the corresponding finite automation.**

**Step 2: Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.**

Let $w = 0^n1^n$.

Then $|w| = 2n > n$.

By pumping lemma, we can write $w = xyz$, with $|xy| \leq n$ and $|y| \neq 0$.

**Step 3: Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.**

The string y can be any one of the following forms:

Case 1: y has only 0's, ie. $y = 0^k$ for some $k \geq 1$.

In this case, we can take i=0. As $xyz = 0^n1^n$, $xz = 0^{n-k}1^n$.

As $k \geq 1$, $n - k \neq n$. So, $xz \notin L$.

Case 2: y has only 1's, ie. $y = 1^m$, for some $m \geq 1$.

In this case, take i=0.

As before, xz is $0^n1^{n-m}$ and $n \neq n - m$. So, $xz \notin L$.

Case 3: y has both 0's and 1's. ie. $y = 0^k1^j$, for some k, $j \geq 1$

In this case, take i=2.

As $xyz = 0^{n-k}0^k1^j1^{n-j}$,$xy^2z = 0^{n-k}0^k1^j0^k1^j1^{n-j}$,

As $xy^2z$ is not of the form $0^i1^i$, $xy^2z \notin L$.

In all three cases, we get a contradiction. So L is not regular.

**Example 2:**

Prove that $L = \{a^{i^2} | i \geq 1\}$ is not regular.

Proof:

**Step 1: Assume that L is regular. Let n be the number of steps in the corresponding finite automation.**

**Step 2: Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.**

Let $w = a^{n^2}$. Then $|w| = n^2 > n$.

By pumping lemma, we can write $w = xyz$ with $|xy| \leq n$ and $|y| > 0$.

**Step 3: Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.**

Let us choose, i=2.

Then consider $xy^2z$.

$|xy^2z| = |x| + 2|y| + |z| > |x| + |y| + |z|$ as $|y| > 0$.

This means $n^2 = |xyz| = |x| + |y| + |z| < |xy^2z|$.

As $|xy| \leq n$, we have $|y| \leq n$.

Therefore,

$|xy^2z| = |x| + 2|y| + |z| \leq n^2 + n$

That is,

$n^2 < |xy^2z| \leq n^2 + n < n^2 + n + n + 1$

That is,

$n^2 < |xy^2z| \leq (n + 1)^2$

Thus $|xy^2z|$ strictly lies between $n^2$ and $(n + 1)^2$, but is not equal to any one of them.

Thus $|xy^2z|$ is not a perfect square and so $xy^2z \notin L$.

But by pumping lemma, $xy^2z \in L$. This is a contradiction.

**Example 3:**

Prove that $L = \{a^p | p$ is a prime$\}$ is not regular.

Proof:

**Step 1: Assume that L is regular. Let n be the number of steps in the corresponding finite automation.**

**Step 2: Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.**

Let p be a prime number greater than n.

Let $w = a^p$.

By pumping lemma, we can write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.

x, y, z are simply strings of a's.

So, $y = a^m$, for some $m \geq 1$ (and $\leq n$).

**Step 3: Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.**

Let $i = p + 1$.

Then $|xy^iz| = |xyz| + |y^{i-1}| = p + (i-1)m = p + pm$.

By pumping lemma, $xy^iz \in L$.

But $|xy^iz| = p + pm = p(1 + m)$.

$p(1 + m)$ is not prime.

So $xy^iz \notin L$.

This is a contradiction.

Thus L is not regular.

**Example 4:**

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

Proof:

**Step 1: Assume that L is regular. Let n be the number of steps in the corresponding finite automation.**

**Step 2: Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.**

Let us consider $ww = a^nba^nb$ in L.

$|ww| = 2(n + 1) > n$

We can apply pumping lemma to write

$ww = xyz$ with $|y| \neq 0$, $|xy| \leq n$.

**Step 3: Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.**

We want to find i so that $xy^iz \notin L$ for getting a contradiction.

The string $y$ can be in only one of the following forms:

Case 1: $y$ has no $b$'s. ie. $y = a^k$ for some $k \geq 1$.

Case 2: $y$ has only one $b$.

We may note that y cannot have two b's. If so, $|y| \geq n + 2$. But $|y| \leq |xy| \leq n$.

In case 1, we can take $i = 0$. Then $xy^0z = xz$ is of the form $a^mba^nb$, where $m = n - k < n$. We cannot write $xz$ in the form $uu$ with $u \in \{a, b\}^*$, and so $xz \notin L$.

In case 2 also, we can take $i = 0$. Then $xy^0z = xz$ has only one b ( as one b is removed from $xyz$, b being in y). So $xz \notin L$ as any element in L should have an even number of a's and an even number of b's.

Thus in both cases, we have a contradiction. So, L is not regular.

**Questions (from Old syllabus of S7CS TOC)**

MGU/Nov2011

1. State the formal definition of an E-NFA (4marks).

2. Design a DFA that accepts the language:

   L = {w/w starts with 01} from the alphabet {0,1}. (4marks)

3a. Explain the different applications of finite automata.

OR

b. Prove that for every NFA there is an equivalent DFA (12marks).

MGU/April2011

1. Define regular expression (4marks).

2. Draw a DFA for the regular expression $[(a^* + b^*)^*]a^*$. (4marks)

3. Differentiate a DFA and NFA (4marks).

4a. i) If 'r' is a regular expression , then show that there is an NFA that accepts L(r).

   ii) Design a minimised FSA that recognize $(1110/100)^*0^*$.

MGU/Nov2010

1. Draw a DFA for the regular expression $(a^*b^*)^*$. (4marks)

2. Define pumping lemma for regular language (4marks).

3a. i) Construct a DFA to accept the language $L = [(a/b)^*/(ab)^*]^*$.

OR

b. Find a minimal DFA for the language $L = \{a^n b^n : n \geq 2, m \geq 1\}$. (12marks)

MGU/May2010

1. State the mathematical definition of DFA (4marks).

2. a. Give regular set for the following expressions : $1(01)^*(10)^*1$.

b. What is the difference between DFa and NFA. (4marks)

3a. Prove that for every non-deterministic finite automation there is an equivalent deterministic finite automation (12marks).

4a. Construct a DFA equivalent to non-deterministic automata given below:

b. Construct a DFA for the language given by:

$L = [(a + b)^*ab(a + b)^*] \cap L[(ab)^*]$. (12marks)

MGU/Nov2009

1. Find minimal DFA's for the language $L = \{a^n b^m, n \geq 2; m \geq 1\}$. (4marks)

2a. Prove the equivalence of NFA and DFA.

OR

b. Explain in detail with an example the conversion of NDFA to DFA (12marks).

MGU/Nov2008

1. Describe a finite automation (4marks).

2. Construct automata to accept $1(1 + 0)^* + a(a + b)^*$. (4marks)

3. Explain pumping lemma for regular languages (4marks).

4a. i) Design a minimum state FSA to recognize the expression $(111/000)^*0$.

ii. Construct automation that accepts language $S \longrightarrow aA, A \longrightarrow abB/b, B \longrightarrow aA/a$.

OR

b. Differentiate between deterministic and non-deterministic finite automation (12marks).

MGU/May2008

1. Differentiate deterministic and non-deterministic automata (4marks).

2. Construct automata to accept $1(1+0)^* + a(a+b)^*$. (4marks)

3a. explain the algorithm for the minimization of DFA.

OR

b. i) Construct DFA for the language given by

$L = [(a+b)^*ab(a+b)^*] \cap L^1[(ab)^*]$

ii) Design a minimum state FSA to recognize the expression $(111/000)^*0$. (12marks)

MGU/JDec2007

1. Write regular expression of set of strings with even number of a's followed by odd number of b's (4marks).

2. Show that the class of languages accepted by finite automata is clossed under union (4marks).

3. What are useless symbols and how they are removed (4marks)?

4a. i) Construct NFA for $(01^* + 1$. (12marks)

5a. i) Construct an automation accepting language generated by grammar $S \longrightarrow aA/a, A \longrightarrow abB$, and $B \longrightarrow bS$.

ii. State and prove pumping lemma for regular languages.

OR

b. Construct DFA for language L over the $\sum = \{0, 1\}$ and $\alpha$ is set of strings ending with "00". (12marks)

MGU/July2007

1. Define regular expression (4marks).

2. How will you use pumping lemma to show that certain languages are not regular? Give the general steps (4marks).

3a. i. Prove that $L = \{ww^R | w\epsilon\{a, b\}^*\}$ is not regular.

ii. Prove that $L = \{a^p | P$ is a prime number } is not regular.

OR

b. i. Give the regular expression for the language $L = \{a^n b^m | n \geq 4, m \leq 3\}$ and its component L.

ii. Define DFA and construct a DFA that accepts $L = \{w\epsilon\{a, b\}^*|$no. of a's in w is even and no. of b's in w is odd$\}$. (12marks)

MGU/Jan2007

1. Define regular expression. (4marks)

2. Construct a DFA (transition diagram) accepting the language $L = \{w\epsilon(a, b)^*|$w has abab as substring (4marks).

3a. i. Prove that $L = a^n b^n | n > 0\}$ is not regular.

OR

b. i. State and prove pumping lemma for regular languages.

ii. Construct a DFA accepting $L = \{w\epsilon(a, b)^*$

MGU/July2006

1. Construct a DFA for $(a/b)^*abb$. (4marks)

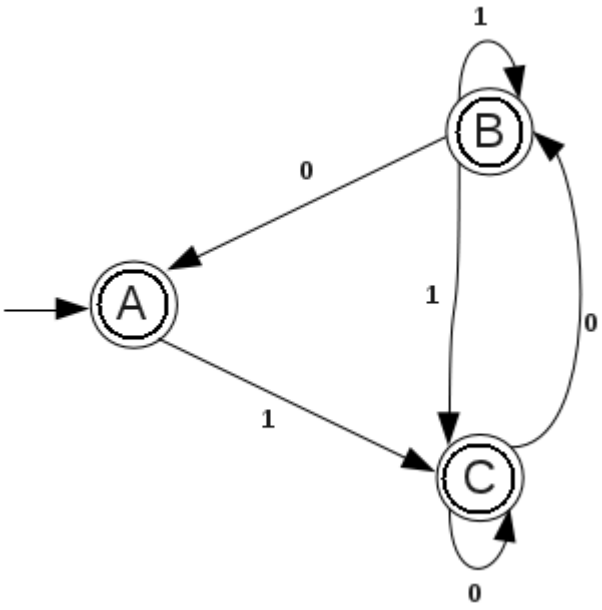
2. Define a NDFA. (4marks)

3a. For the regular expression, $a^*(a/b)^*b$, draw the NFA. Obtain DFA from NFA.

OR

b. i. Design an algorithm for minimising the states of DFA.

ii. Construct the regular expression equivalent to the state diagram given below:




MGU/Nov2005

1. What are the strings in the regular sets denoted by the regular expression $ab^*$ and $(ab)^*$? (4marks)

2. Define a NDFA. (4marks)

3a. If L is a language accepted by NDFA, then prove that there exists a DFA that accepts L.

OR

b. Discuss briefly any two applications of finite state automata (12marks).

**References**

.

Pandey, A, K (2006). An Introduction to automata Theory and Formal Languages. Kataria & Sons.

Mishra, K, L, P; Chandrasekaran, N (2009). Theory of Computer Science. PHI.

Nagpal, C, K (2011). Formal Languages and Automata Theory. Oxford University Press.

Murthy, B, N, S (2006). Formal Languages and Automata Theory. Sanguine.

Kavitha,S; Balasubramanian,V (2004). Theory of Computation. Charulatha Pub.

Linz, P (2006). An Introduction to Formal Languages and Automata. Narosa.

Hopcroft, J, E; Motwani, J; Ullman, J, D (2002). Introduction to Automata Theory, Languages and Computation. Pearson Education.