1.1 Introduction

The set theory concepts are widely used in this subject. So, let us learn some set concepts. Now, let us see **"What is a set? What is a power set?"**

Definition: A ***set*** is a collection of well-defined unique objects. All the elements of the set should be enclosed between '{' and '}' separated by commas. The name of the set is normally in capital letters. Normally, the sets are denoted by capital letters such as A, B, C,...etc. and the elements within the set are denoted by lower case letters such as a, b, c etc.

Ex: The set of positive integers greater than 0 but less than or equal to 25 and divisible by 5 can be represented as:

$S = \{5, 10, 15, 20, 25\}$

Now, let us see **"What is a power set?"**

Definition: Let A be the set. The set of all subsets of set A is called **power set of A** and is denoted by **P(A)** or **2^.**

Ex: Let $A = \{1, 2, 3\}$. The subsets of the set A are shown below:

$\{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}, \{\ \}$

The set of above subsets is called ***power set*** and is denoted by **2^**.

$2^A = \{\ \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}, \{\ \}\ \}$

Now, let us see **"What is Cartesian product (cross product) of two sets?"**

Definition: The Cartesian product of A and B is defined as shown below:

$A \times B = \{ (a, b) \mid a \in A \text{ and } b \in B\}$.

where

- (a, b) is an ordered pair
- 'a' is an element of set A
- 'b' is an element of set B.

For example, let $A = \{a, b, c\}$, $B = \{0, 1\}$. Then

$A \times B = \{ (a, 0), (a, 1), (b, 0), (b, 1), (c, 0), (c, 1) \}$

Now, let us see **"What is union of two sets?"**

Definition: The union of two sets A and B is defined as shown below:

$A \times B = \{x \mid x \in A \text{ or } x \in B\}$

which is a collection of elements both in set A and set B

For example, let $A = \{ a, b, c \}$ $B = \{ 0,1 \}$. Union of A and B is given by

$A \cup B = \{ a, b, c, 0, 1 \}$

Now, let us see **"What is intersection of two sets?"**

Definition: The intersection of two sets A and B is defined as shown below:

$A \cup B = \{ x \mid x \in A \text{ and } x \in B \}$

which is the collection of common elements in both the sets A and B.

For example, let A = { a, b, c } B = { c, d, e }. The intersection of A and B is given by:

$$A \cap B = \{ \{a, b, c\} \cap \{c, d, e\} \}$$
$$= \{ c \}$$

1.2 Languages and Strings

Let us have familiarity with basic notations and terminologies used in the subject.

1.2.1 Alphabet

First, let us see **"What is an alphabet? What is power of an alphabet?"**

Definition: An **alphabet** is defined as a finite non-empty set of symbols. An alphabet is denoted by the symbol Σ. The members of Σ are called **symbols** or **characters**.

For example,

- Σ = {0, 1} denotes the set of alphabets of machine language.
- Σ = {a, b,....z, A, B,...Z, 0, 1,....9, #, (,), {, }, <, >, !, [,],.....} represent the alphabets of C language.

Definition: The **power of an alphabet** denoted by Σ^i is the set of words of length i. For example, if Σ = {0, 1}, then

- $\Sigma^0 = \{\Sigma\}$ is set of words of length 0.
- $\Sigma^1 = \{0, 1\}$ is set of words of length 1.
- $\Sigma^2 = \{00,01,10,11\}$ is set of words of length 2.
- $\Sigma^3 = \{000,001,010,100,011,101,110,111\}$ is set of words of length 3.
- .....
- .....
- and so on.

Now, let us see **"What are the two variations of power of an alphabet?"**

The variations of power of an alphabet are shown below:

- Kleene closure (Kleene star)
- Kleene plus

Now, let us see **"What is Kleene closure(or Kleene star/star operator)? Give example"**

Definition: The Kleene closure Σ* is defined as follows:

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup ....$$

which is the set of words of any length (possibly ε i.e., the null string). Each string is made up of symbols only from Σ.

Ex : Let Σ = {0, 1}. Then Σ* is obtained as shown below:

$\Sigma^0 = \{\varepsilon\}$ set of words of length 0

$\Sigma^1 = \{0, 1\}$ set of words of length 1

$\Sigma^2 = \{00,01,10,11\}$ set of words of length 2

$\Sigma^3 = \{000,001,010,100,011,101,110,111\}$ set of words of length 3

.....

.....

So, $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup$.....

$\Sigma^* = \{\varepsilon , 0 ,1 ,00 ,01 ,10 ,11 ,000 ,001,.....\}$

$\{0,1\}^* = \{\varepsilon , 0 ,1 ,00 ,01 ,10 ,11 ,000 ,001,.....\}$

which is the set of strings of 0's and 1's of any length. Now, let us see **"What is Kleene plus? Give example"**

Definition: The **Kleene plus** is a variation of Kleene star operator. The Kleene plus denoted by Σ^+ is defined as follows.

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup .....$$

which is the set of words of any length except the null string i.e., ε (Σ^0) is not part of Σ^+ and hence $\varepsilon \notin \Sigma^+$.

For example, let $\Sigma = \{0, 1\}$. Then Σ^+ is shown below:

$$\Sigma^+ = \{0, 1, 00, 01, 10, 11, 000, 001, ...\}$$

The above set is defined as strings of 0's and 1's of any length except the null string.

Note: $\Sigma^* = \Sigma^+ \cup \varepsilon$. This can be written as $\Sigma^+ = \Sigma^* - \varepsilon$

1.2.2 String

Now, let us see **"What is a string?"**

Definition: A *string* is defined as a finite sequence of symbols from an alphabet Σ.

- The shortest string is empty string and it is denoted by the symbol ε (pronounced as epsilon)
- The set of all possible strings over an alphabet Σ is written as Σ^*

For example, $\Sigma = \{0, 1\}$ is set of binary alphabets. The various strings that can be obtained from Σ are:

$$\{\varepsilon, 0, 1, 00, 01, 10, 11, 010101, 1010.....\}$$

where Σ is an empty string. Note that an infinite number of strings can be generated from Σ and once the string is generated, it has finite number of symbols in it and has a definite sequence.

Note: $\varepsilon \notin \Sigma$ i.e., ε is not part of Σ.

1.2.3 Functions on Strings

In this section, let us see **"What are various string functions that are used for processing**

strings?" The various string functions that are used are:

- Length
- Concatenation
- Replication
- Reversal
- **Length:** The *length* of a string w denoted by |w| is the number of symbols in *w*. **For example:**

  $|\varepsilon| = 0$ // **Length of empty string is 0**

  |abcdef| = 6 // **Length of string "abcdef"**

A length function is defined as:

$\#_c(w)$ // **Returns the number of times the symbol *c***
// **appears in string *w***

For example:

#a(abababa) = 4 // **number of *a*'s in string "abababa" is 4**

#b(abababa) = 3 // **number of *b*'s in string "abababa" is 3**

#c(abababa) = 0 // **number of *c*'s in string "abababa" is 0**

- **Concatenation:** The **concatenation** of two strings *u* and *v* denoted by:

  uv

  is the string obtained by writing all the letters of string *u* followed by all the letters of string *v* (i.e., appending the symbols of *v* to the right of *u*)

For example,

- Let $u = a_1a_2a_3.....a_n$ and $v = b_1b_2b_3.....b_m$, then the concatenation of *u* and *v* is denoted by

  $uv = a_1a_2a_3.....a_n\ b_1b_2b_3.....b_m$

- Let u = "Computer" and v = "Science". The concatenation of *u* and *v* denoted by

  uv = ComputerScience

  Here, |uv| = |u| + |v| = |Computer| + |Science|

  = 8 + 7

  = 15

- **Replication:** The process of repeating a string pattern *w* by a specified number *i* is called replication and it is denoted by:

  w^i

For example,

- $w^0 = \varepsilon$
- $a^4 = aaaa$

- $b^2a^2 = bbaa$
- $(ba)^2 = baba$
- $(hello)^3 = hellohellohello$
- $a^0b^3 = bbb$
- $a^3b^0 = aaa$

♦ **Reversal:** The ***reversal*** of a string w denoted by w^R is obtained by writing the symbols of string w in reverse order i.e., if

$$w = a_1a_2a_3.....a_n$$

then the reverse of w is denoted by w^R and is given by

$$w_R = a_na_{n-1}a_{n-2}....a_3a_2a_1$$

For example,

♦ Let $w = \varepsilon$. Then, $w^R = \varepsilon$

♦ Let $w = a$. Then, $w^R = a$

♦ Let $w = aaabbb$. Then $w^R = bbbaaa$

Theorem 1.1: *If w and x are strings then prove that* $(wx)^R = x^Rw^R$

Let us prove this using mathematical induction of length of x i.e., |x|

Base case: $|x| = 0 \Rightarrow x = \varepsilon$

So, $(wx)^R = (w\varepsilon)^R = (w)^R = (\varepsilon w)^R = \varepsilon w^R = \varepsilon^R w^R = x^R w^R$

Hence, $(wx)^R = x^Rw^R$

Inductive hypotheses: For each $n \geq 0$, where $|x| = n$, let us assume that $(wx)^R = x^Rw^R$.

To prove: It is required to prove that the hypotheses is true for $|x| = n + 1$

Proof: Consider a string x where $|x| = n + 1$

Let $x = ua$ for some character a and $|u| = n$

Then,

$(wx)^R = (w(ua))^R$		replacing x by ua
$= ((wu)a)^R$		associativity of concatenation
$= a(wu)^R$		by definition of reversal
$= a(u^Rw^R)$		by induction hypotheses
$= (au^R)w^R$		associativity of concatenation
$= (ua)^Rw^R$		by definition of reversal
$= x^Rw^R$		replacing ua by x

Hence, the proof.

1.2.4 Relations on Strings

In this section, let us **"Define the terms: i) substring ii) proper substring iii) prefix iv) proper prefix v) suffix vi) proper suffix"**

- **Substring:** A string x is a substring of string *w* if and only if the string *x* occurs contiguously as part of *w*

  For example,

  - aaa is a substring in string bbaaaba
  - aba is a substring in string ababbb

- **Proper substring:** A string *x* is a proper substring of string *w* if and only if the string x occurs contiguously as part of *w* and $x \neq w$

For example,

- aaa is a proper substring in string bbaaaba
- aaa is not proper substring in string aaa

Note: Every string is a substring of itself. The symbol ε is a substring of every string.

Example 1.1: List all the substrings and proper substrings of string "HELLO"

The various substrings of string "HELLO" are:

ε, H, E, L, O, HE, EL, LL, LO, HEL, ELL, LLO, HELL, ELLO and HELLO

The various proper substrings of string "HELLO" are:

ε, H, E, L, O, HE, EL, LL, LO, HEL, ELL, LLO, HELL, ELLO

Note: "HELLO" is a substring of "HELLO". But, "HELLO" is not a proper substring of "HELLO"

- **Prefix:** A substring x of a string xy is called **prefix**. The prefix always occurs from the beginning of the string.

For example,

- ε is a prefix of every string
- Every string is a prefix of itself
- For a string "HELLO" the prefixes are ε, H, HE, HEL, HELL and HELLO

- **Proper Prefix:** A substring x of a string xy is **proper prefix** if and only if $y \neq \varepsilon$.

For example,

- ε is a proper prefix of every string
- For a string "HELLO" the proper prefixes are ε, H, HE, HEL and HELL. **Note**: "HELLO" is not the proper prefix of "HELLO"

- **Suffix:** A substring x of a string yx is called **suffix**. The suffix always occurs at the end of the string.

For example,

- ε is a suffix of every string
- Every string is a suffix of itself
- For a string "HELLO" the suffixes are ε, O, LO, LLO, ELLO and HELLO

♦ **Proper suffix:** A substring x of a string yx is **proper suffix** if and only if $y \neq \varepsilon$.

For example,

- ε is a proper suffix of every string
- For a string "HELLO" the proper suffixes are ε, O, LO, LLO, ELLO.

Note: "HELLO" is not the proper suffix of "HELLO"

1.3 Languages

Now, let us see **"What is a language? Give example"**

Definition: A ***language*** is a set of strings over finite alphabet Σ. Formally, a language L is subset of Σ^* denoted by $L \subseteq \Sigma^*$.

For example,

♦ A language of strings consisting of equal number of 0's and 1's can be represented as

{ε, 01, 10, 0011,1010,0101,0011,....}

♦ The language of strings consisting of n number of 0's followed by n number of 1's can be represented using the set as shown below:

{ε, 01, 0011, 000111,.....}

♦ A language containing empty string ε is denoted by {ε}

♦ An empty language is denoted by ϕ.

Example 1.1: For the following English language descriptions, give the formal language definition

Description in English	**Formal language description**
Language that contains string consisting of any number of a's and b's	$L = \{w \in \{a, b\}^* \}$
Language that contains strings of a's and b's where all a's precede b's	$L = \{w \in \{a, b\}^*$: all a's precede all b's in w} or $L = \{a^i b^j \mid i \geq 1, j \geq 1\}$
Language that contains strings of a's and b's ending with 'a'	$L = \{w : w = ya$ and every $y \in \{a, b\}^*$ or $L = \{w \in \{a, b\}^*$: last character of w is a}
Empty Language	$L = \{\} = \phi$. L is the language that contains no strings.

Language with empty string	L = {ε} Here, L is the language that contains empty string
A C program that halts on all inputs	L = { w : w is a C program that halts on all inputs}
Language consisting of a's and b's which do not start with b	L = { w ∈ {a, b}* : no prefix of w starts with b} or L = {w ∈ {a, b}* : first character of w is a} ∪ {ε}
Language consisting of a's and b's without b's or Language consisting of any number of a's	L = { w ∈ {a, b}* : no prefix of w contains b} or L = { w ∈ {a}* } or L = {an : n > 0} or L = {ε, a, aa, aaa, aaaa, ...}

1.4 Cardinality of a Language?

Now, let us see **"What is the cardinality of a language?"**

Definition: The length of a language is called cardinality of a language. That is, the number of members of a set is called the *cardinality*.

Ex1: L = { }. The cardinality of the empty set is zero.

Ex2: L = {ε}. The cardinality of the empty language is one.

Now, the question is **"If Σ = ϕ then what is the cardinality of a language?"** The cardinality of Σ* is countably infinite. Now, let us prove it.

Theorem: If Σ = ϕ then Σ* is countably infinite

Proof: By definition Σ* is set of strings of length 0, length 1, length 2 and so on. Since Σ* contains strings of any length, is it countably infinite. That is all the languages are either finite or countably infinite.

1.5 Why Theory of Computations

Now, let us see **"What are the applications of Theory of Computations"** The various applications of Theory of Computations are shown below:

- Used to design languages enabling machine/machine communication and man/machine communication. For example, Network communication protocols, Hyper Text Markup Language (HTML).
- Used to design and implement modern programming languages such as C/C++/Java/Python etc.

- Used to design vending machines (such as ATMs), communication protocols and also used in building security devices.
- Used to design interactive video games.
- DNA molecules can be analysed with the help of finite state machines and context-free grammars.
- Artificial Intelligence programs are used to solve various problems ranging from medical diagnosis to factory scheduling.
- Many natural structures such as organic molecules and computer networks can be modelled as graphs and the solution can be obtained using very efficient graph algorithms.

Exercises

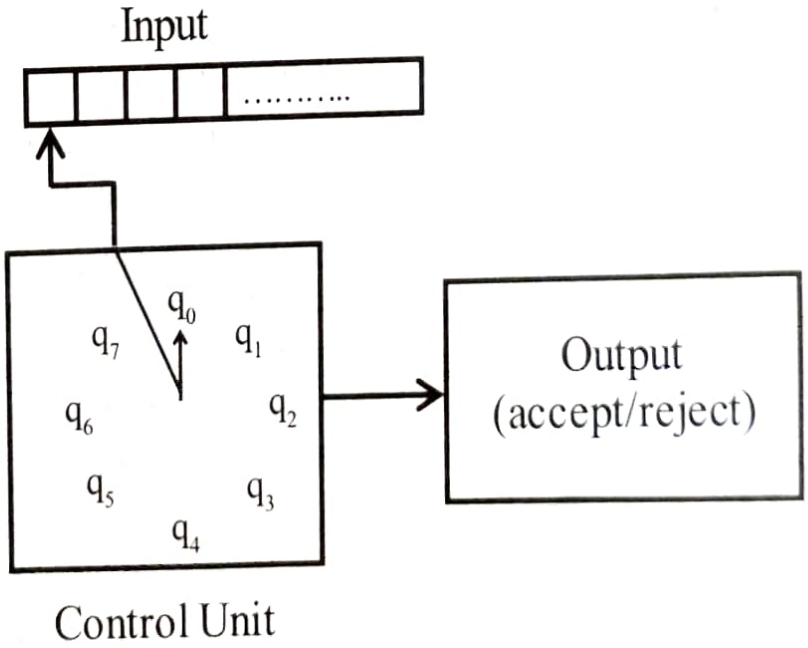
1) What is an alphabet? What is power of an alphabet? Give examples.
2) What are the two variations of power of an alphabet? Explain.
3) What is Kleene closure(or Kleene star/star operator)? Give example.
4) What is Kleene plus? Give example.
5) What is a string? What are various string functions that are used for processing strings?
6) Define the terms with example:
   a) Length b) Concatenation c) Replication d) Reversal
7) If w and x are strings then prove that $(wx)^R = x^R w^R$
8) Define the terms: each
   i) substring ii) proper substring iii) prefix iv) proper prefix
   v) suffix vi) proper suffix"
9) List all the substrings and proper substrings of string "HELLO".
10) What is a language? Give example.
11) What is the cardinality of a language?
12) What are the applications of Theory of Computations?

2.5 Finite State Machines

In this section, let us see **"What is a finite state machine? Explain the working of FSM"**

Definition: The **Finite State Machine** (in short **FSM**) is also called **Finite Automaton** (in short **FA**) is a computing device which accepts a string as input and produces the output ***accept*** or ***reject***. Automaton is a Greek word and it means a machine.

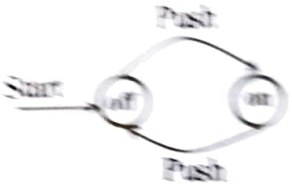
Working: The block diagram of FA is shown below:



- **Input:** The string to be processed is stored in input.

- **Control Unit:** It consists of a number of states. The states are identified by q_0, q_1, ... q_n. Initially, the machine will be in start state q_0 and the machine will have at least one final state. Based on the current input symbol and the current state of the machine, it may change its state.
- **Output:** When end of input is encountered and if the machine is in final state, the input string is accepted by the machine. But, when end of input is encountered and if the machine is not in final state, the input string is rejected by the machine.

For example, consider an electric switch which has only two states "OFF" and "ON". To start with, the switch will be in OFF state. When we push the button, it goes to ON state. If we push once again, it goes to OFF state. This can be represented as shown below:

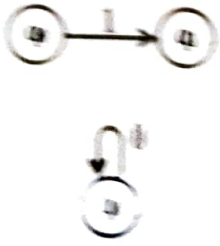


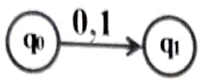
Note: The circles represent the states and the labels associated with arcs represent the input given to go to another state. The arrow with the label **Start** (not originating from any state) is considered as the **start state.**

Note: The various conventions used when we design finite state machine are shown below:

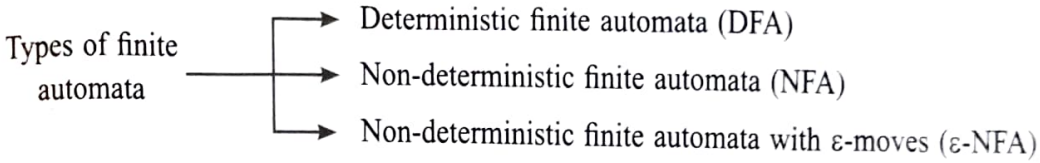
- The lower case letters near the beginning of the alphabets such as a, b, c, ... etc. are the elements of alphabets Σ. That is $[a, b, c, d, \ldots] \in \Sigma$
- The lower case letters near the end of the alphabets such as u, v, w, x, y, z represent input strings. That is $[u, v, w, x, y, z] \in \Sigma$

Now, let us see **"What are the various notations used during designing of FA?"**

Symbol	Meaning
	A circle with an arrow which is not originating from any node represents the **start state** of machine. Two circles are used to **represent a final state.** Here, q_2 is the final state.
	An arrow with label 1 goes from state q_1 to state q_2. This indicates that there is a transition from state q_1 on input symbol 1 to state q_2. This can be written as: $\delta(q_1, 1) = q_2$ An arrow with label 0 starts and ends in q_0. This indicates the machine in state q_0 on reading a 0, remains in same state q_0. This is represented as: $\delta(q_0, 0) = q_0$

	An arrow with label 0 or 1 goes from state q_0 to state q_1. This indicates that the machine in state q_0 on reading a 0 or 1 enters into state q_1. This is represented as: $\delta(q_0, 0) = q_1$ $\delta(q_0, 1) = q_1$

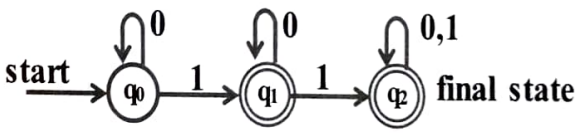
Now, let us see **"What are the different types of finite state machines?"** The different types of finite state machines are shown below:



Note: Deterministic Finite Automaton **(DFA)** is also called Deterministic Finite State Machine **(DFSM)**. Let us the term DFA instead of DFSM.

2.5.1 Deterministic Finite Automata (DFA)

Before worrying about the definition, let us consider the following pictorial representation of DFA.



From the above figure, observe following components of DFA:

- **States:** The above DFA has three states q_0, q_1 and q_2 and can be represented as

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

  The circles represent the states of DFA and they are classified as shown below:

  Start state: q_0 with an arrow labeled start is considered as start state.

  Final state: q_1, q_2 with two circles represent final states.

- **Input alphabets:** Each edge is labeled with 0 or 1 and represent the input alphabets which can be denoted as:

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

- **Transitions:** Transition is nothing but change of state after consuming an input symbol. If there is a change of state from q_i to q_j on an input symbol a, then we write

  $$\delta\ (q_i, a) = q_j$$

For example, in the above diagram, the various transitions shown are represented as shown below:

$\delta(q_0,0) = q_0$ There is a transition from state q_0 on 0 to state q_0

$\delta(q_0,1) = q_1$ There is a transition from state q_0 on 1 to state q_1

$\delta(q_1,0) = q_1$ There is a transition from state q_1 on 0 to state q_1

$\delta(q_1,1) = q_2$ There is a transition from state q_1 on 1 to state q_2

$\delta(q_2,0) = q_2$ There is a transition from state q_2 on 0 to state q_2

$\delta(q_2,1) = q_2$ There is a transition from state q_2 on 2 to state q_2

δ:(Q x Σ) to Q

which is the cross product of $Q = \{q_0, q_1, q_2\}$ and $\Sigma = \{0,1\}$

Now, let us see **"What is a transition function?"** The transition function δ is defined as:

$$\delta: Q \times \Sigma \text{ to } Q$$

which is read as **"δ is a transition function which maps Q X Σ to Q"**. For example, the change of state from state *q* on input symbol *a* to state *p* is denoted by

$$\delta(q, a) = p$$

where

- δ : is a function called **transition function.**
- **q** : is the **first parameter** representing the **current state of the machine.**
- **a** : is the **second parameter** representing the **current input symbol read**
- **p** : is the **next state of machine** which is returned by the transition function

Note: In other words, the transition function δ accepts two parameters namely state *q* and input symbol *a* as the parameters and returns a next state *p*

- **Start state (q_0):** q_0 with the label *start* is treated as the start state
- **Final state (q_2):** q_1 and q_2 with two circles are treated as the final or accept states.

Note: From this discussion, it is observed that the DFA has five components:

(states, input alphabets, transitions, start state, final states)

Q	Σ	δ	q_0	F

With this concept, now let us see **"What is a DFA/DFSM?"**

Definition: The **Deterministic Finite Automaton in short DFA or DFSM** is 5-tuple or quintuple indicating five components:

$$M = (Q, \Sigma, \delta, q_0, F)$$

where

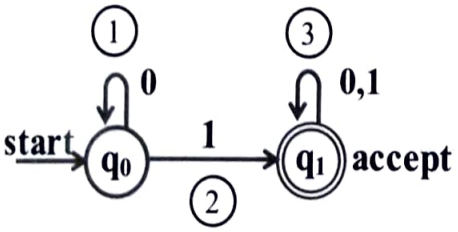
- **M** is the name of the machine. It can be called by any name

- **Q** is non-empty, finite set of states.
- Σ is non-empty, finite set of input alphabets.
- δ is a transition function which is a mapping from Q x Σ to Q which is denoted by δ: Q x Σ to Q.
- $q_0 \in Q$ is the start state.
- $F \subseteq Q$ is set of accepting or final states.

Thus, name **DFA** emerges from the following facts:

- **D** (**D**eterministic) – There is exactly one transition for every input symbol from the state. So, it is possible to **determine** exactly to which state the machine enters into after consuming the input symbol. So, the machine is **deterministic**.
- **F** (**F**inite) – Has finite number of states and edges. So, **it is deterministic and finite.**
- **A** (**A**utomaton) – Automaton is a machine which may accept the string or reject the string. So, it is deterministic finite automaton. It is also called **Deterministic Finite State Machine (DFSM).**

Note: The DFA can have only one transition from a state on an input symbol. Consider the following DFA and observe the various transitions:



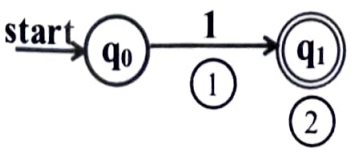
(1) ♦ From state q_0 on input 0, the machine will stay in q_0.

(2) ♦ From state q_0 on input 1, the machine enters into state q_1.

(3) ♦ From state q_1 on input 0, the machine stays in q_1.

(3) ♦ From state q_1 on input 1, the machine stays in q_1.

Observe from the above diagram that, **there is exactly one transition defined** from a state on an input symbol. Sometimes, there can be no transitions from a state as shown below:



(1) ♦ From state q_0 on input 1, the machine enters into state q_1.

(2) ♦ But, from state q_1, the transition is not defined on any of the input symbol. We say ***there is a zero transition*** from state q_1.

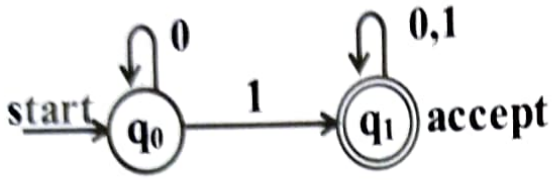
2.5.2 Simple Notations for DFAs

When we design a DFA, specifying a DFA as a 5-tuple such as **M = (Q, Σ, δ, q_0, F)** along with the detailed description of each transition is very tedious and difficult to understand. So, a DFA can be specified using simpler and more effective notations.

Now, let us see **"What are the various simpler notations used to specify a DFA?"** The two notations using which the DFA's can be easily represented are:

- Transition diagram (Transition graph)
- Transition table

For example, the transition diagram and its equivalent transition table are shown below:



Transition diagram

δ	0	1
→ q_0	q_0	q_1
*q_1	q_1	q_1

Transition table

- The transition diagram of DFA has two states q_0 and q_1
- There are two input symbols 0 and 1
- Start state is represented using an arrow mark not originating from any state and labeled *start*
- The final states are represented by two circles.
- The transition from state q on input symbol a to state p is represented by a directed edge originating from state q and ending at state p with label a.

- Represented using two rows q_0 and q_1
- The two input symbols 0 and 1 correspond to two columns
- The start state is identified by putting an arrow with direction towards right
- The final states are represented by putting stars (*'s) by the side of states
- The equivalent transition is represented by writing p in row q and column a.

Formal definition of DFA

M = (Q, Σ, δ, q_0, F) where

- Q = $\{q_0, q_1\}$
- Σ = {0, 1}
- q_0 is the start state
- F = $\{q_1\}$
- δ is shown using transition table.

Example 2.40: Obtain a DFA to accept L = { w : |w| mod 3 = 0} where ∈ = {a}

Solution: The language accepted by the DFA can be written as shown below:

L = {w : |w| mod 3 = 0} with q_0 as the start state and q_0 as final state

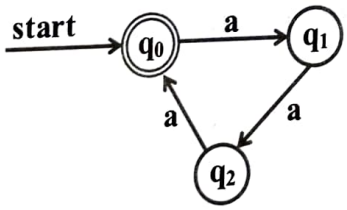
The transitions can be obtained using the relation " **$\delta(q_i, a) = q(i + 1 \bmod k)$**"

where k = 3(divisor) and i = 0, 1,2 (remainders after dividing by 3)

The transitions obtained using the above relation are shown below:

i	$\delta(q_i, a) = q(i + 1 \bmod 3)$
0	$\delta(q_0, a) = q(0 + 1 \bmod 3) = q_1$
1	$\delta(q_1, a) = q(1 + 1 \bmod 3) = q_2$
2	$\delta(q_2, a) = q(2 + 1 \bmod 3) = q_0$

So, the transition diagram to accept the given language is shown in figure:



The language accepted by above DFA can also be written as:

L = {w : |w| mod 3 = 0} where ∈ = {a}

or

L = {w: na(w) are divisible by 3 } where Σ={a}

or

$L = \{a^{3n} : n \geq 0\}$

Example 2.41: Obtain a DFA to accept the language:

L = { w : |w| mod 3 = 0} on Σ = {a, b}

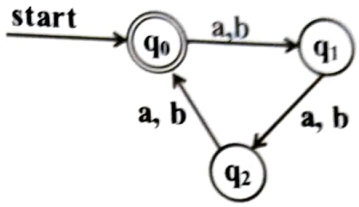
Solution: The language accepted by the DFA can be written as shown below:

$L = \{w : |w| \bmod 3 = 0\}$ with q_0 as the start state and q_0 as final state

The transitions can be obtained using the following relation:

$\delta(q_i, \{a,b\}) = q(i + 1 \bmod k)$ where $k = 3$ and $i = 0, 1, 2$ as shown below:

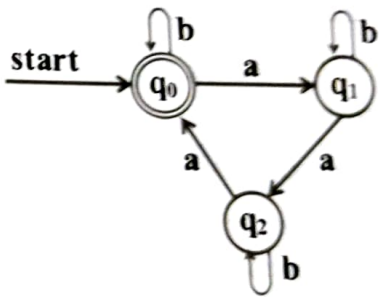
i	$\delta(q_i, a) = q(i + 1 \bmod 3)$
0	$\delta(q_0, \{a, b\}) = q(0 + 1 \bmod 3) = q_1$
1	$\delta(q_1, \{a, b\}) = q(1 + 1 \bmod 3) = q_2$
2	$\delta(q_2, \{a, b\}) = q(2 + 1 \bmod 3) = q_0$



So, the transition diagram to accept the given language is shown in figure:

Example 2.42: Obtain a DFA to accept $L = \{ w : n_a(w) \bmod 3 = 0\}$ on $\Sigma = \{a, b\}$

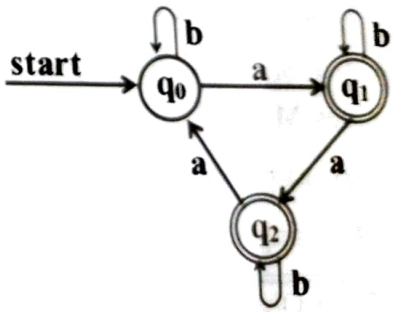
Solution: The transitions are obtained as in example 2.30 with q_0 as the start state and q_0 as the final state. Since there is no restriction on number of b's we can generate any number of b's at states q_0, q_1 and q_2. The final DFA can be written as shown here:



Example 2.43: Obtain a DFA to accept $L = \{ w : n_a(w) \bmod 3 \neq 0\}$ on $\Sigma = \{a, b\}$

Solution: The transitions are obtained as in example 2.30 with q_0 as the start state. Since $n_a(w) \bmod 3 \neq 0$, state q_0 should not be the

final state. So, all states except q_0 will be the final states. The final DFA can be written as shown here:



Example 2.44: Draw a DFA to accept language: $L = \{w : |w| \bmod 5 \neq 0\}$ on $\Sigma = \{a\}$.

Solution: The language accepted by the DFA can be written as shown below:

$L = \{ w \text{ where } |w| \bmod 5 \neq 0\}$

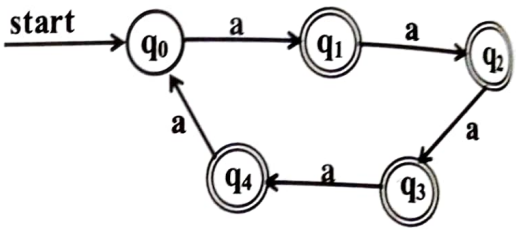
where

- q_0 is the start state.
- k = 5 (divisor)
- i = 0, 1, 2, 3, 4 are the remainders obtained after dividing by k
- **Note:** If |w| mod 5 = 0, then q_0 is the final state. But, in this case |w| mod 5 ≠ 0

and hence except q_0, all are final states. That is, q_1, q_2, q_3 and q_4 are final states. The transition along with DFA can be obtained using the following relation:

$\delta(q_i, a) = q(i + 1 \bmod k)$ where k = 5 and i = 0, 1, 2, 3, 4

i	$\delta(q_i, a) = q(i + 1 \bmod 5)$
0	$\delta(q_0, a) = q(0 + 1 \bmod 5) = q_1$
1	$\delta(q_1, a) = q(1 + 1 \bmod 5) = q_2$
2	$\delta(q_2, a) = q(2 + 1 \bmod 5) = q_3$
3	$\delta(q_3, a) = q(3 + 1 \bmod 5) = q_4$
4	$\delta(q_4, a) = q(4 + 1 \bmod 5) = q_0$



2.6.4 String Length Mod k Problems with Relational Operators

Now, let us see ***"What is the general method to solve "String length mod k problems with relational operators?"***

The general method is shown below:

- **Step 1:** Solve the string length mod k problems connected by a relational operator as two independent problems as in previous section.
- **Step 2:** Now, the states Q of combined machine M that accepts a given string w is obtained by taking the cross product of Q1 and Q2 as shown below:

  Q = Q1 x Q2

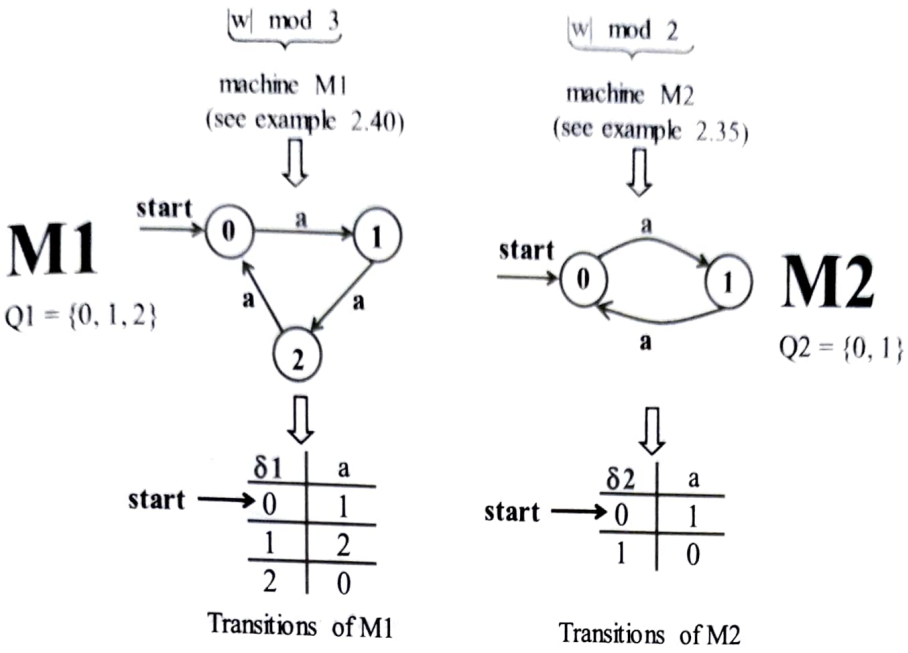
  where Q1 and Q2 are the states of machines M1 and M2 respectively.
- **Step 3:** Obtain the transitions δ of combined machine using the transitions δ_1 and δ_2 of machines M1 and M2 respectively. That is,

  $\delta(\{p, q\}, a) = (\delta_1(p, a), \delta_2(q, a))$
- **Step 4:** Using the relation identify the final states.

Example 2.45: Obtain a DFA to accept a string w satisfying the following condition:

(a) |w| mod 3 ≥ |w| mod 2 where w ∈Σ and Σ ={a}

Solution: In the above two problems we have both "|w| mod 3" and "|w| mod 2". So, we need to construct two machines M1 and M2 one with respect to "|w| mod 3" and another with respect to "|w| mod 2" as shown below:



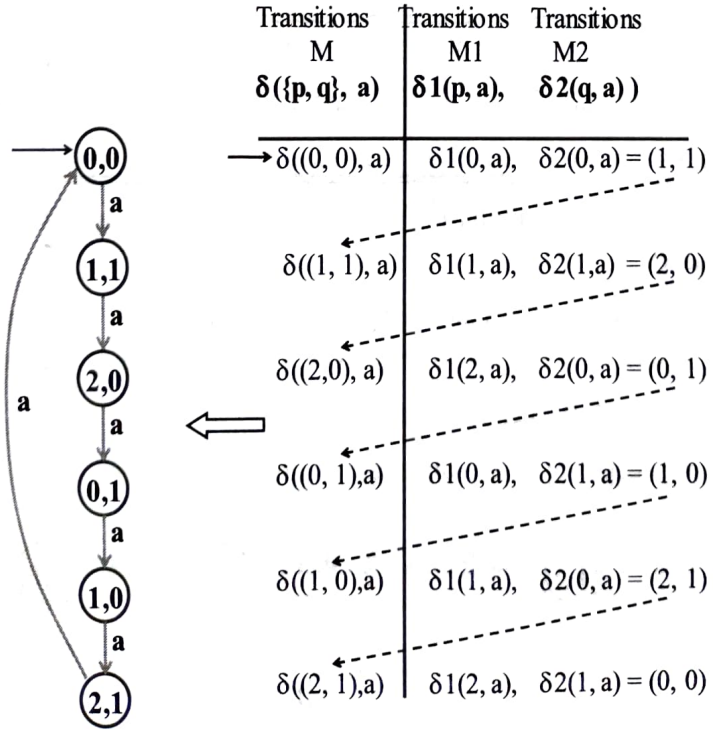
Observe that there are 2 machines:

- The machine M1 accepts a string w whose length is divided by 3 i.e., |w| mod 3
- The machine M2 accepts a string w whose length is divided by 2 i.e., |w| mod 2

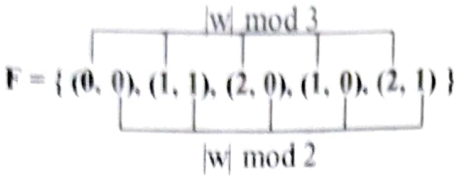
Now, the states Q of combined machine M that accepts a given string w can be obtained by taking the cross product of Q1 and Q2 as shown below:

Q = Q1 x Q2 = { (0,0), (0,1),(1,0),(1,1),(2,0),(2,1) }

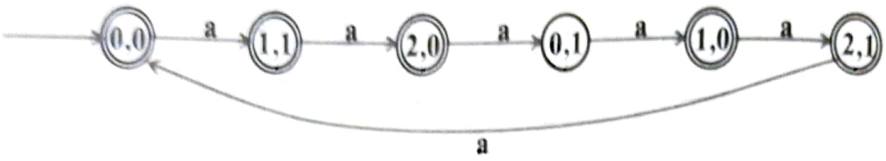
where **state (0,0) is the start state.** The transitions on each of the pair (x, y) can be obtained as shown below:



To accept strings of w such that **|w| mod 3 ≥ |w| mod 2**, the pairs (x, y) such that **x ≥ y** are final states. So, in the above DFA, the final states are:



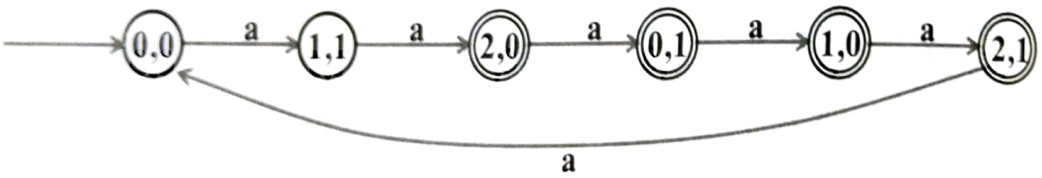
So, the above DFA to accept the given language can be written as shown below (by writing the states horizontally):



Note: To accept strings of w such that **|w| mod 3 ≠ |w| mod 2**, the above DFA can be written as shown below, with following final states.

F = { (2, 0), (0,1), (1, 0), (2, 1) }

So, the DFA to accept the given language is shown below:



Example 2.46: Obtain a DFA to accept the following language L = {w such that

(a) |w| mod 3 ≥ |w| mod 2 where w ∈Σ and Σ ={a, b}

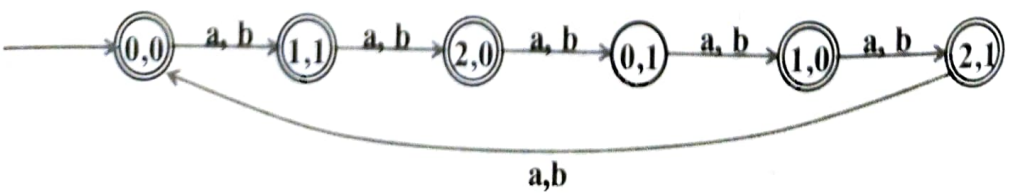
(b) |w| mod 3 ≠ |w| mod 2 where w ∈Σ and Σ ={a,b }

Solution: The solution is exactly similar to the above, but, the extra label *b* should be used along with *a* as shown below:

Case 1: To accept strings of w such that **|w| mod 3 ≥ |w| mod 2**, w ∈Σ and Σ ={a,b} the pairs (x, y) such that x >= y are final states. So, in the above DFA, the final states are:

F = { (0, 0), (1, 1), (2, 0), (1, 0), (2, 1) }

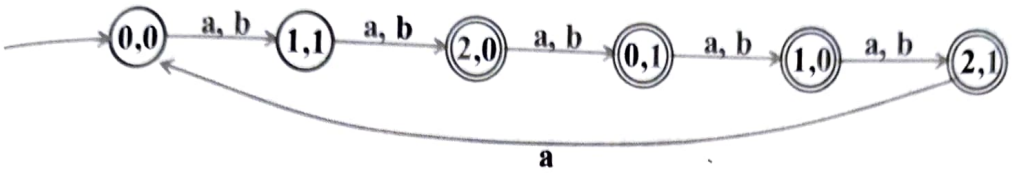
So, the DFA to accept the given language is shown below:



Case 2: To accept strings of w such that $|w| \bmod 3 \neq |w| \bmod 2$, $w \in \Sigma$ and $\Sigma = \{a,b\}$ the pairs (x, y) such that $x \neq y$ are final states. So, the final states in the DFA are:

$F = \{ (2, 0), (0,1), (1, 0), (2, 1) \}$

So, the DFA to accept the given language is shown below:



2.6.5 Number of Specified Symbols Mod k Problems

Number of specified symbol say a in a string w is denoted by $N_a(w)$. Now, let us see **"What is the general method to solve $N_a(w)$ mod k problems?"** The general method remains same as for "string length mod k problems". The procedure is repeated for convenience.

Step 1: Find the remainders obtained by dividing the number of specified symbols by *k*. The remainders obtained after dividing by k are shown below:

0, 1, 2....k-1

Step 2: Identify the transitions. The transitions can be obtained using the following relation:

$$\delta(q_i, a) = q(i + 1 \bmod k)$$

where

- *k* is the divisor
- *i* represent the remainders 0, 1, 2.....k-1 obtained after dividing by k

Step 3: Identify the start state and final state: q_0 is always the start state. Suppose, the language to be accepted is:

$L = \{ N_a(w) \bmod k = j \}$ then

↓

q_j is the final state.

Ex1: If $L = \{ N_a(w) \bmod k = 0$ for some string w$\}$, then q_0 is the final state.

Ex2: If $L = \{ N_a(w) \bmod k = 3$ for some string w$\}$, then q_3 is the final state.

Ex3: If $L = \{ N_a(w) \bmod k = 5$ for some string w$\}$, then q5 is the final state.

Step 4: Write the DFA. The DFA can be written using the transitions given in step 2 with *start state* and *final state* as obtained in step 3.

Example 2.47: Obtain a DFA to accept strings of even number of a's

Solution: The language accepted by the DFA can be written as shown below:

$L = \{ N_a(w) \bmod 2 = 0\}$ with q_0 as the start state and q_0 as final state

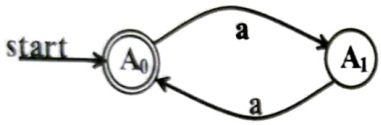
The transitions can be obtained using the following relation:

$\delta(q_i, a) = q(i + 1 \bmod k)$

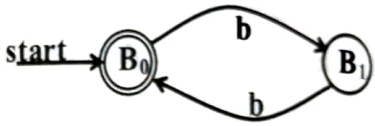
where k = 2, and i = 0, 1 (remainders after dividing by k) as shown below:

i	$\delta(q_i, a) = q(i + 1 \bmod 5)$
0	$\delta(q_0, a) = q(0 + 1 \bmod 2) = q_1$
1	$\delta(q_1, a) = q(1 + 1 \bmod 2) = q_0$

So, the transition diagram to accept even number of a's is shown below:



On similar lines, the transition diagram to accept even number of b's is shown below



Example 2.48: Draw a DFA to accept language: $L = \{N_a(w) \bmod 5 \neq 0 \}$

Solution: The language accepted by the DFA can be written as shown below:

$L = \{ N_a(w) \bmod 5 \neq 0\}$

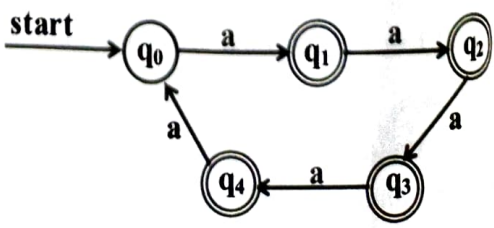
where

- q_0 is the start state.
- k = 5 (divisor)
- i = 0, 1, 2, 3, 4 are the remainders obtained after dividing by k
- **Note:** If $N_a(w) \bmod 5 = 0$, then q_0 is the final state. But, It is $N_a(w) \bmod 5 \neq 0$

and hence except q0, all are final states. That is, q_1, q_2, q_3 and q_4 are final states. The transitions can be obtained using the following relation:

$\delta(q_i, a) = q(i + 1 \bmod k)$ where k = 5 and i = 0, 1, 2, 3, 4

i	$\delta(q_i, a) = q(i + 1 \bmod 5)$
0	$\delta(q_0, a) = q(0 + 1 \bmod 5) = q_1$
1	$\delta(q_1, a) = q(1 + 1 \bmod 5) = q_2$
2	$\delta(q_2, a) = q(2 + 1 \bmod 5) = q_3$
3	$\delta(q_3, a) = q(3 + 1 \bmod 5) = q_4$
4	$\delta(q_4, a) = q(4 + 1 \bmod 5) = q_0$



So, the DFA $\mathbf{M = (Q, \Sigma, \delta, q_0, F)}$ is defined as:

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{a\}$
- δ is shown in transition diagram as shown above:
- q_0 = start state
- $F = \{q_1, q_2, q_3, q_4\}$

Example 2.49: Draw a DFA to accept language: $L = \{N_a(w) \bmod 3 = 0\}$

Solution: The language accepted by the DFA can be written as shown below:

$$L = \{ N_a(w) \bmod 3 = 0\}$$

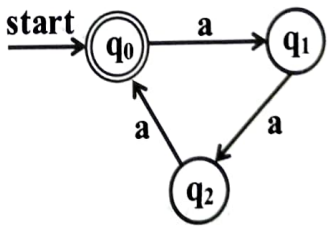
where

- q_0 is the start state.
- $k = 3$ (divisor)
- $i = 0, 1, 2$ are the remainders obtained after dividing by k
- **Note:** If $N_a(w) \bmod 3 = 0$, then q_0 is the final state.

The transitions can be obtained using the following relation:

$$\delta(q_i, a) = q_{(i + 1 \bmod k)} \text{ where } k = 3 \text{ and } i = 0, 1, 2$$

i	$\delta(q_i, a) = q(i + 1 \bmod 5)$
0	$\delta(q_0, a) = q(0 + 1 \bmod 5) = q_1$
1	$\delta(q_1, a) = q(1 + 1 \bmod 5) = q_2$
2	$\delta(q_2, a) = q(2 + 1 \bmod 5) = q_3$



So, the DFA $\mathbf{M = (Q, \Sigma, \delta, q_0, F)}$ is defined as:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a\}$
- δ is shown in transition diagram as shown above:
- q_0 = start state
- $F = \{q_0\}$

2.6.6 Number of Specified Symbol Mod k Problems (with logical operators)

Now, let us see **"What is the general method to solve "Number of symbols mod k problems with relational operators?"** The general method remains same as for "string length mod k problems". The procedure is repeated for convenience.

- **Step 1:** Solve "number of specified symbols mod k problems" connected by a logical operator as two independent problems as in previous section.
- **Step 2:** Now, the states Q of combined machine M that accepts a given string w is obtained by taking the cross product of Q1 and Q2 as shown below:

$$\mathbf{Q = Q1 \times Q2}$$

where Q1 and Q2 are the states of machines M1 and M2 respectively.

- **Step 3:** Obtain the transitions δ of combined machine using the transitions $\delta 1$ and $\delta 2$ machines M1 and M2 respectively. (See examples below)
- **Step 4:** Identify the final states by looking at the problem statement.

Example 2.50: Obtain a DFA to accept strings of a's and b's having even number of a's and even number of b's

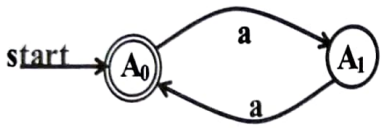
Solution: The given language can be written as shown below:

$$L = \{ N_a(w) \bmod 2 = 0 \text{ and } N_b(w) \bmod 2 = 0\}$$

In the above language, we have both "$N_a(w) \bmod 2 = 0$" and "$N_b(w) \bmod 2 = 0$". So, we need to construct 2 machines M1 and M2 one with respect to "$N_a(w) \bmod 2 = 0$" and another with respect to "$N_b(w) \bmod 2 = 0$" as shown below:

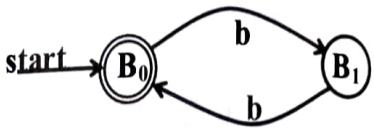
$N_a(w) \bmod 2 = 0$

machine M1
(see example 2.35)



$N_b(w) \bmod 2 = 0$

machine M2
(see example 2.35)



The transition table to accept even number of a's and even number of b's are shown below:

start	δ1	a
→	A_0	A_1
	A_1	A_0

Transitions of M1

start	δ2	b
→	B_0	B_1
	B_1	B_0

Transitions of M2

Observe that there are 2 machines:

- The machine M1 accepts a string w such that $N_a(w) \bmod 2 = 0$
- The machine M2 accepts a string w such that $N_b(w) \bmod 2 = 0$

Now, the states Q of combined machine M that accepts a given string w can be obtained by taking the cross product of Q1 and Q2 as shown below:

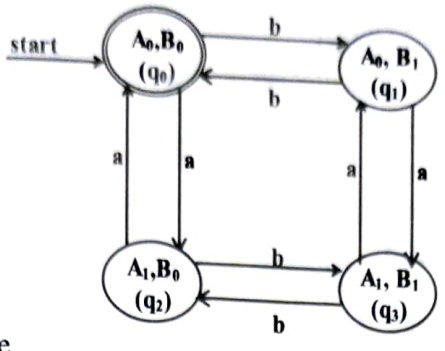
$$Q = Q1 \times Q2 = \{ (A_0,B_0), (A_0,B_1), (A_1,B_0), (A_1,B_1) \}$$

⇨// horizontal transition for input symbol b

⇧// vertical transition for input symbol *a*.

The horizontal transitions are written for input symbol *b* whereas the vertical transitions are written for input symbol *a*. So, the DFA M = (Q, Σ, δ, q_0, F) is defined as:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{a, b\}$
- δ = shown in transition diagram



DFA to accept even number of a's and even number of b's

- q_0 = start state
- $F = \{q_0\}$

Note: The language accepted by above DFA can be written as:

$L = \{w : w$ has even number of a's and even number of b's.$\}$

or

$L = \{w :$ Both $N_a(w)$ and $N_b(w)$ are divisible by $2\}$

or

$L = \{w :$ Both $N_a(w)$ and $N_b(w)$ are multiples of $2\}$

or

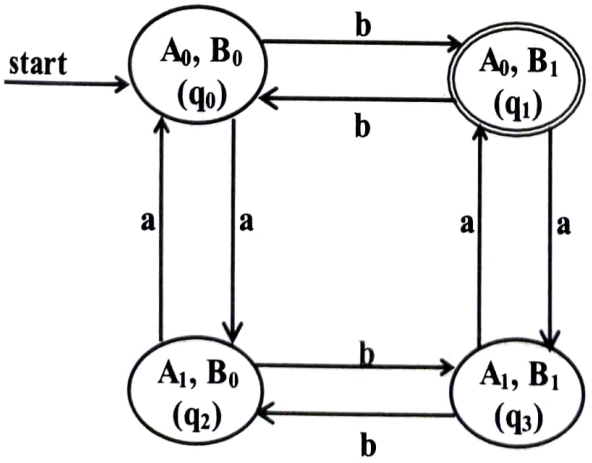
$L = \{w \mid N_a(w) \bmod 2 = 0$ and $N_b(w) \bmod 2 = 0\}$

Note: $N_a(w)$ is the total number of a's in the string w and $N_b(w)$ is the total number of b's in the string w.

Note: The DFA to accept even number of a's and odd number of b's can be obtained by making:

$$A_0, B_1$$
$$(q_1)$$

as the only final state as shown below:



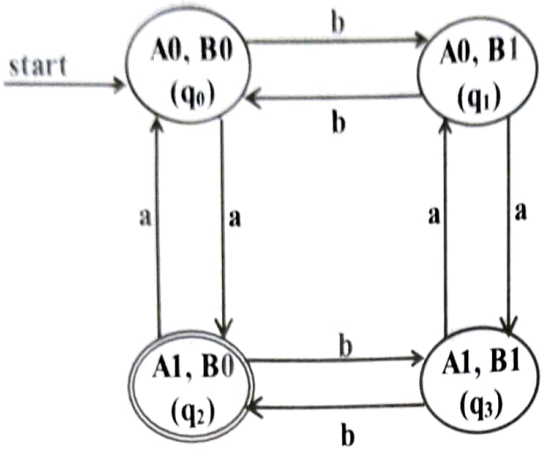
DFA to accept even number of a's and odd number of b's

Note: The DFA to accept odd number of a's and even number of b's can be obtained by making:

$$A_1, B_0$$

$$(q_2)$$

as the only final state as shown below:



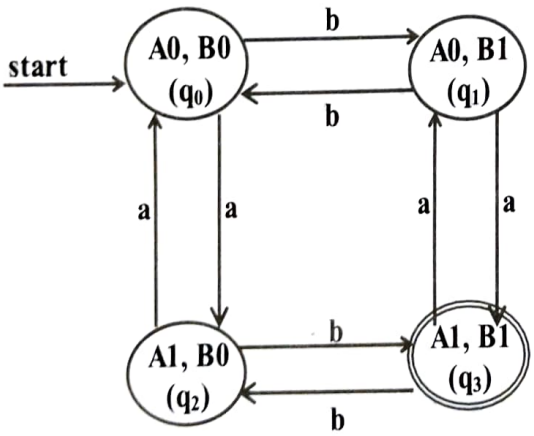
DFA to accept odd number of a's and even number of b's

Note: The DFA to accept odd number of a's and odd number of b's can be obtained by making:

$$(A_1, B_1)$$

$$(q_3)$$

as the only final state as shown below:



DFA to accept odd number of a's and odd number of b's

2.6.7 Disadvantages of DFA

Now, let us see ***"What are the various disadvantages of DFA?"*** The various disadvantages of DFA are listed as shown below:

- Constructing some of the DFA's is very difficult
- The DFA cannot guess about its input
- The DFA is not very powerful

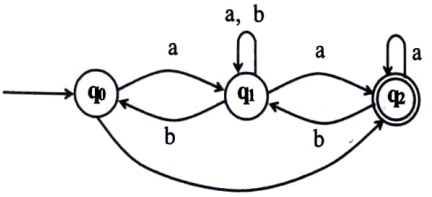
- At any point of time, the DFA is in only one state. So, a DFA does not have the power to be in several states at once.

2.7 Why NFA?

Computers are completely deterministic machines. The state of the computer can be predicted from the input and initial state. We cannot find a computer which is non-deterministic. In such case, the question is ***"Why non-deterministic finite automaton?"*** All the disadvantages of DFA mentioned earlier can be overcome using NFA. The various advantages of NFA are:

- Constructing most of the NFA's is very easy.
- A "non-deterministic" finite automaton has the ability to guess something about its input.
- It is more powerful than DFA
- It has the power to be in several states at once.
- An NFA is an efficient mechanism to describe some complicated languages concisely.

Before defining NFA, let us consider the following transition diagram:



Let us see ***"What is the specialty of the above finite automaton?"*** Observe that from a given state there is possibility of zero, one or more edges leaving that state after consuming the input symbol.

Ex 1: From state q_2, there are zero edges (or no edges) for the input symbol *b*

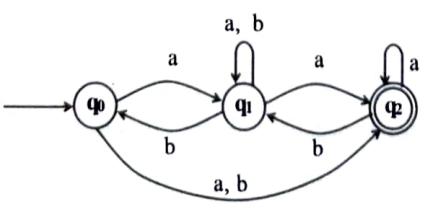
Ex 2: From state q_2, there is one edge from q_2 to q_2 labeled *a*

Ex 3: From state q_0, there are two edges (q_0, q_1) and (q_0, q_2) labelled *a*.

Note: In the FA shown above, there can be zero, one or more transitions on an input symbol. Such machines are called non-deterministic finite state machines or non-determinstic finite automaton.

2.7.1 Non-deterministic Finite Automata (NFA)

Before worrying about the definition, let us consider the pictorial representation of NFA.



In the above figure, observe the following components of NFA:

- **States:** The NFA has three states q_0, q_1 and q_2 and can be represented as

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

  Note: The power set of Q is denoted by 2^Q which is set of subset of set Q. This is de as shown below:

  $2^Q = \{ \{\}, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\} \}$

- **Input alphabets:** Each edge is labeled with *a* or *b*. Here, *a* and *b* represent the alphabets and denoted as:

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

- **Transitions:** Transition is nothing but change of state from current state after consu an input symbol. If there is change of state from q_i to q_j on input symbol *a*, then write

  $$\delta(q_i, a) = q_j$$

  If there is a change of state from q_i to q_j and q_i to q_k, then we write

  $$\delta(q_i, a) = \{q_j, q_k\}$$

The transition diagram of above NFA is shown below:

Current State	Input	Next state	Representation	
q_0	a	q_1,q_2	$\delta(q_0, a) =$	$\{q_1,q_2\}$
q_0	b	q_2	$\delta(q_0, b) =$	$\{q_2\}$
q_1	a	q_1,q_2	$\delta(q_1, a) =$	$\{q_1,q_2\}$
q_1	b	q_0,q_1	$\delta(q_1, b) =$	$\{q_0,q_1\}$
q_2	a	q_2	$\delta(q_2, a) =$	$\{q_2\}$
q_2	b	q_1	$\delta(q_2, b) =$	$\{q_1\}$

δ: Q x Σ to 2^Q [power set]

Now, let us see ***"What is a transition function?"*** The transition function δ is defined as:

$$\delta: Q \times \Sigma \text{ to } 2^Q$$

which is read as "δ ***is a transition function which maps Q x Σ to 2^Q***". For example, the chan of state from state *q* on input symbol *a* to states $p_1,p_2,p_3,\ldots.p_n$ is denoted by:

$$\delta(q, a) = \{ p_1, p_2, p_3,\ldots.p_n \}$$

where

- **δ** : is called transition function
- **q** : is the first parameter representing the current state of the machine
- ***a*** : is the second parameter representing the current input symbol
- **$p_1, p_2, p_3,\ldots.p_n$** : represent possible states of the machine.

Note: In other words, the transition function δ accepts two parameters namely state q and input symbol a as the parameters and returns set of states the machine enters into.

- **Start state (q_0):** The state q_0 with the label start is treated as the start state
- **Final state (q_2):** The state q_2 with two circles is treated as the final state

Note: From the above discussion, it is observed that the NFA has five components: (states, input alphabets, transitions, start state, final states)

$$\underset{Q}{\downarrow} \quad \underset{\Sigma}{\downarrow} \quad \underset{\delta}{\downarrow} \quad \underset{q_0}{\downarrow} \quad \underset{F}{\downarrow}$$

With this discussion, now let us see ***"What is a non-deterministic finite automaton (NFA)?***

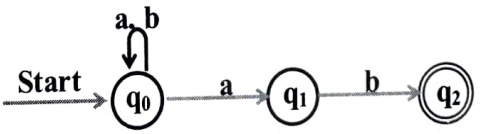
Definition: A non-deterministic finite automaton in short an NFA is a 5-tupe or quintuple:

$$\mathbf{M = (Q, \Sigma, \delta, q_0, F)}$$

where

- Q is non-empty, finite set of states
- Σ is non-empty, finite set of input alphabets
- δ: Q x Σ to 2^Q i.e., δ is transition function which is a mapping from Q x Σ to 2^Q. Based on the current state and input symbol, the machine enters into one or more states.
- $q_0 \in Q$ is the start state.
- $F \subseteq Q$ is set of final states.

Suppose δ(q, a) is in P where P is in 2^Q and a is in Σ. This indicates that there is a transition from state *q* on input symbol *a* to set of states P. **Note: In an NFA there can be zero, one or more transitions on an input symbol. For example,** an NFA to accept strings of a's and b's ending with *ab* is shown below:



The above NFA can be specified formally as M = (Q, Σ, δ, q_0, F) where

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- q_0 is the start state
- $F = \{ q_2 \}$
- δ is shown using the transition table

δ	a	b
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$
*q_2	ϕ	ϕ

Note: In the transition table of NFA, each entry in the table should be denoted by a set. Also, when there is no transition from a given state on an input symbol, make an entry ϕ (or -) indicating an empty set.

Now, let us see **"What is a configuration of a NFA?"** The configuration of NFA is nothing but the snapshot which gives the following information:

2.6 DFA Design Techniques

Now, let us see **"What are the various problem types for which we can construct DFA?"**

There various types of problems for which we can construct a DFA are:

- Pattern recognition problems
- Divisible by k problems
- |w| mod k problems
- |w| mod k problems with relational operators
- $N_a(w)$ mod k problems
- $N_a(w)$ mod k problems with logical operators

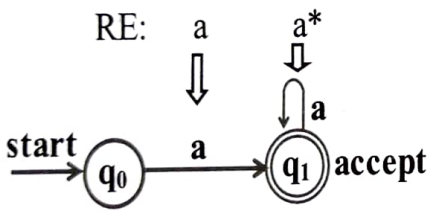
2.6.1 Pattern Recognition Problems

Now, let us see how to construct DFA's from simple machines to complex machines. In this section, let us concentrate on pattern recognition problems.

Example 2.4: Draw a DFA to accept string consisting of at least one 'a'"

Solution: The string consisting of at least one 'a' can be **stated as string consisting of one 'a' followed by any number of a's**. It is denoted by **aa** and its equivalent DFA can be written as shown below:

Transition diagram



Formal definition of DFA

M = (Q, Σ, δ, q_0, F) where

- Q = {q_0, q_1}
- Σ = {a}
- q_0 is the start state
- F = {q_1}
- δ is shown using transition table.

δ	a
→ q_0	q_1
*q_1	q_1

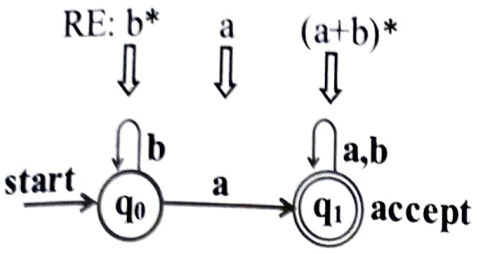
Example 2.5: Draw a DFA to accept strings of a's and b's with at least one 'a'"

Solution: The strings of a's and b's with at least one 'a' can be stated as

- string consisting of any number of b's denoted by b*
- followed by one 'a'
- followed by any number of a's and b's denoted by (a + b)*.

So, the given language can be represented as b* a (a+b)* and its equivalent transition diagram along with formal definition can be written as shown below:

Transition diagram



Formal definition of DFA

M = (Q, Σ, δ, q_0, F) where

- Q = {q_0, q_1}
- Σ = {a, b}
- q_0 is the start state
- F = {q_1}
- δ is shown using transition table.

δ	a	b
→ q_0	q_1	q_0
*q_1	q_1	q_1

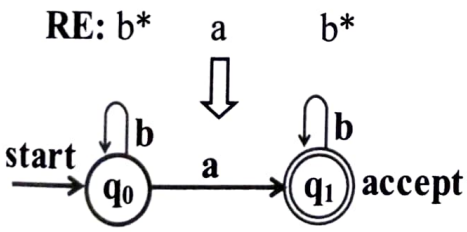
Example 2.6: Draw a DFA to accept strings of a's and b's having exactly one *a*

Solution: The strings of a's and b's having exactly one 'a' can be stated as

- string consisting of any number of b's denoted by b*
- followed by one 'a'
- followed by any number of b's denoted by b*.

So, the given language can be represented as b* a b* and its equivalent transition diagram along with formal definition can be written as shown below:

Transition diagram



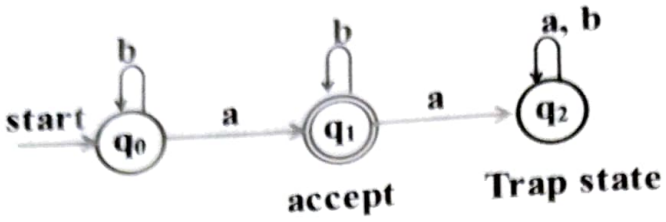
Formal definition of DFA

M = (Q, Σ, δ, q_0, F) where

- Q = {q_0, q_1}
- Σ = {a, b}
- q_0 is the start state
- F = {q_1}
- δ is shown using transition table.

δ	a	b
→ q_0	q_1	q_0
*q_1	-	q_1

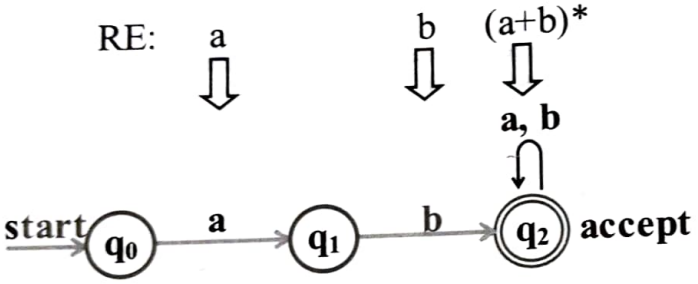
Note: In the above DFA, there is no transition from state q_1 for the input symbol a. But, we can include a transition from state q_1 for the input symbol a to *a* non-final reject state q_2 as shown below:



Here, state q_2 is called a *trap state*. Now, let us see **"What is a trap state?"** A state for which there exists transitions to itself for all the input symbols chosen from Σ and there is no way to reach the final state is called a **trap state.** It is also called ***dead state.***

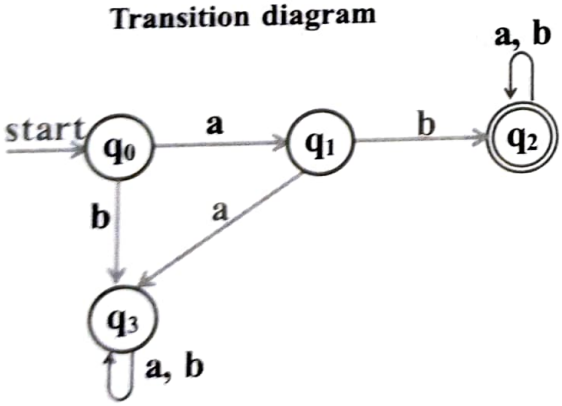
Example 2.7: Obtain a DFA to accept strings of a's and b's starting with the string ab

Solution: The strings of a's and b's starting with the string *ab* can be stated as "String *ab* followed by any combination of a's and b's". So, the given language can be represented as **ab (a+b)*** and its equivalent transition diagram along with formal definition can be written as shown below:



Since the transitions are not defined on q_0 and q_1 for the input symbol *b* and *a* respectively, we can have transitions to the trap state. So, the above DFA can also be written as shown below.

Transition diagram



Formal definition of DFA

M = (Q, Σ, δ, q_0, F) where

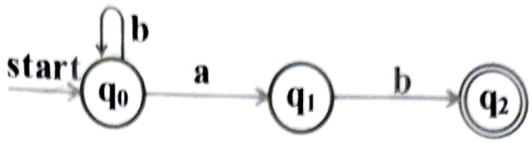
- Q = {q_0, q_1}
- Σ = {a, b}
- q_0 is the start state
- F = {q_2}
- δ is shown using transition table.

δ	a	b
→ q_0	q_1	q_3
* q_1	q_3	q_2
q_2	q_2	q_2
q_3	q_3	q_3

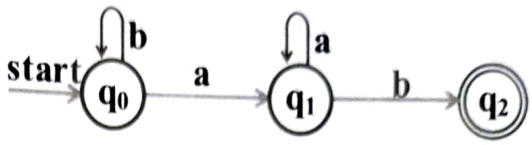
Example 2.8: Obtain a DFA to accept strings of a's and b's ending with the string ab

Solution: The minimum string that can be accepted is "ab" and the equivalent DFA can be written as:

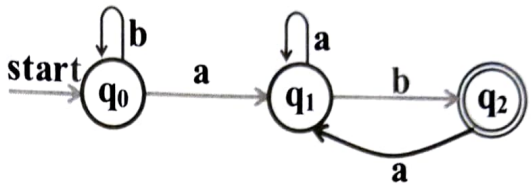
- **$\delta(q_0, b)$:** In state q_0, **any number of b's can be accepted** and still the string ends with ab and the equivalent DFA can be written as:



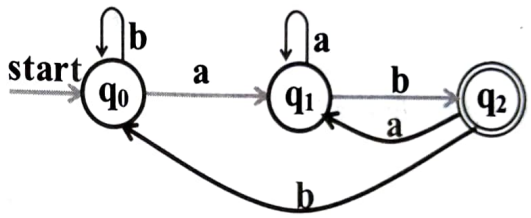
- **$\delta(q_1, a)$:** In state q_1, **any number of a's can be accepted** and still the string ends with *ab* and the equivalent DFA can be written as:



- **$\delta(q_2, a)$:** In state q_2 on 'a' the machine can goto q_1 accepting **any number of a's** and still the string ends with ab and the equivalent DFA can be written as:



- **$\delta(q_2, b)$:** In state q_2 on 'b', the machine can goto q_0 accepting **any number of b's** and from q_0, the machine can goto q_2 accepting the string ending with *ab*. The equivalent DFA can be written as:



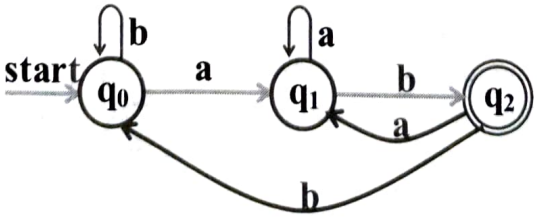
So, formally, the DFA can be written as: **M = (Q, Σ, δ, q_0, F)** where

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- q_0 is the start state
- $F = \{q_2\}$
- δ is shown using the transition table:

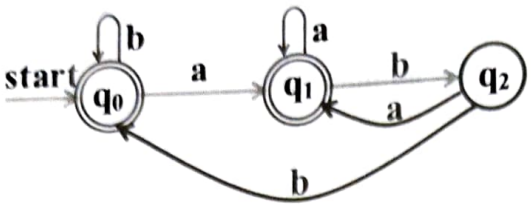
δ	a	b
→ q_0	q_1	q_0
q_1	q_1	q_2
*q_2	q_1	q_0

Example 2.9: Obtain a DFA to accept strings of a's and b's which do not end with the string ab. Show the sequence of moves made by the DFA for the strings "abaaba" and "abab"

Solution: First, let us construct the DFA to accept strings of a's and b's ending with *ab* as shown in previous problem:



Now, **change the final states to non-final states and non-final states** to final states. The resulting DFA accepts strings of a's and b's which do not end with *ab* and can be written as:



So, formally, the DFA can be written as: **M = (Q, Σ, δ, q_0, F)** where

- Q = {q_0, q_1, q_2}
- Σ = {a, b}
- q_0 is the start state
- F = {q_0, q_1}
- δ is shown using the transition table:

δ	a	b
→*q_0	q_1	q_0
*q_1	q_1	q_2
q_2	q_1	q_0

Input string "abaaba": The sequence of moves made by the DFA for the string "abaaba" is shown below:

(Initial configuration)

(q_0, abaaba) ⊢ (q_1, baaba)
⊢ (q_2, aaba)
⊢ (q_1, aba)
⊢ (q_1, ba)
⊢ (q_2, a)
⊢ (q_1, ε)

(Final configuration)

In the final configuration, q_1 is final state and so the string "abaaba" is accepted.

Input string "abab": The sequence of moves made by the DFA for the string "abab" is shown below:

(Initial configuration)

(q_0, abab) ⊢ (q_1, bab)
⊢ (q_2, ab)
⊢ (q_1, b)
⊢ (q_2, ε)

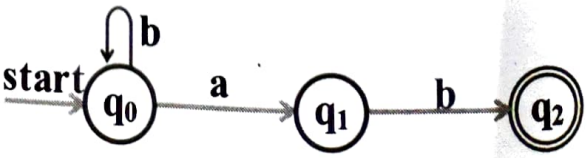
(Final configuration)

In the final configuration, q_2 is not final state. So, the string "abab" is rejected by DFA.

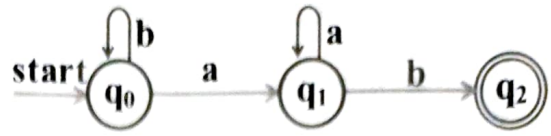
Example 2.10: Obtain a DFA to accept strings of a's and b's having a sub string ab

Solution: The minimum string that can be accepted is "ab" and the equivalent DFA can be written as:

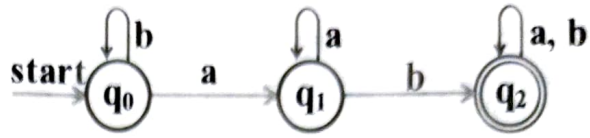
- **δ(q_0, b):** In state q_0, **any number of b's can be accepted** and the string ends with *ab*. The resulting string will have the substring *ab* and the equivalent DFA can be written as:



- **$\delta(q_1, a)$:** In state q_1, any number of a's can be accepted and still the string ends with *ab*. The resulting string will have the sub string *ab* and the equivalent DFA can be written as:



- **$\delta(q_2, a)$:** The moment the machine enters into state q_2, the string will have the substring ab. So, **in state q_2, accept any number of a's and b's and the string will have substring *ab***.



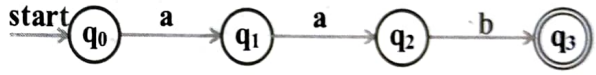
So, formally, the DFA can be written as: **$M = (Q, \Sigma, \delta, q_0, F)$** where

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- q_0 is the start state
- $F = \{q_2\}$
- δ is shown using the transition table:

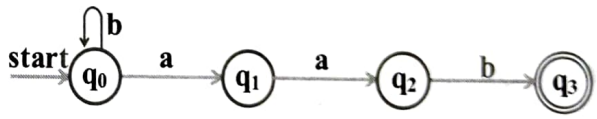
δ	a	b
→ q_0	q_1	q_0
q_1	q_1	q_2
*q_2	q_2	q_2

Example 2.11: Obtain a DFA to accept strings of a's and b's having a sub string aab

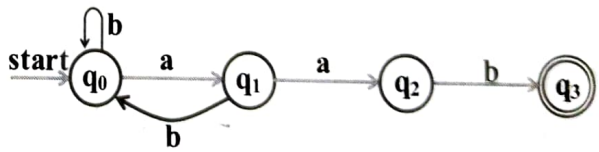
Solution: The minimum string that can be accepted is "aab" and the equivalent DFA can be written as:



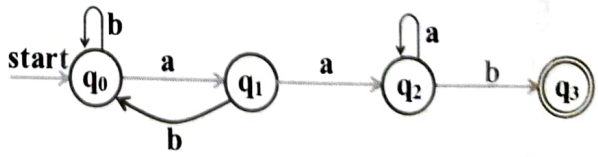
- **$\delta(q_0, b)$:** In state q_0, **any number of b's can be accepted** and the string ends with *aab*. The resulting string will have the substring *aab* and the equivalent DFA is:



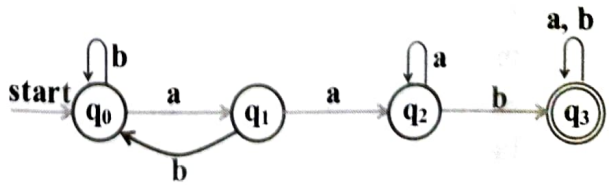
- **$\delta(q_1, b)$:** In state q_1, if the input symbol is *b*, the sequence *aa* is broken and so go back to q_0 and the equivalent DFA is:



- **$\delta(q_2, a)$:** When the machine enters into state q_2, it will have the substring *aa*. So, in state q_2, the machine can accept any number of a's and the string will have substring *aa*.



- **$\delta(q_3, \{a, b\})$:** The moment the machine enters into state q_3, it will have the substring *aab*. So, in state q_3, the machine can accept any number of a's and b's and the string will have substring *aab*. The DFA is:



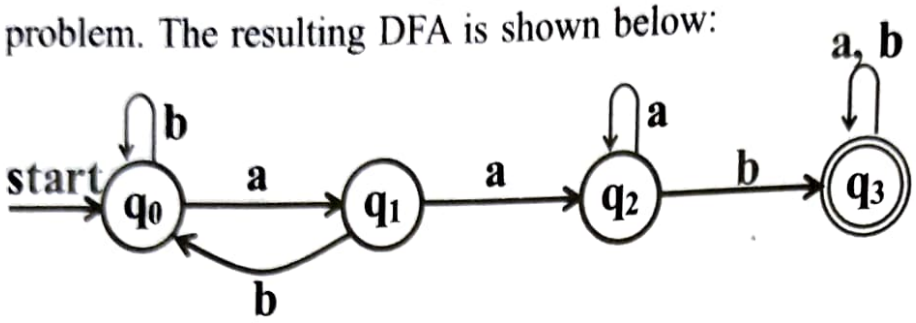
So, formally, the DFA can be written as: $M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{a, b\}$
- q_0 is the start state
- $F = \{q_3\}$
- δ is shown using the transition table:

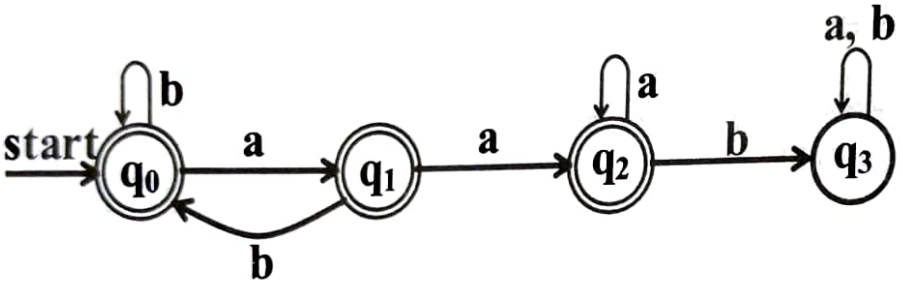
δ	a	b
→ q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_3
*q_3	q_3	q_3

Example 2.12: Obtain a DFA to accept strings of a's and b's having no sub-string aab

Solution: First, let us construct a DFA to accept strings of a's and b's having a sub-string aab as shown in previous problem. The resulting DFA is shown below:



Now, to accept strings of a's and b's without having a substring make all non-final states to final states and final states to non-final states. The resulting DFA is shown below:



So, formally, the DFA can be written as: $M = (Q, \Sigma, \delta, q_0, F)$ where

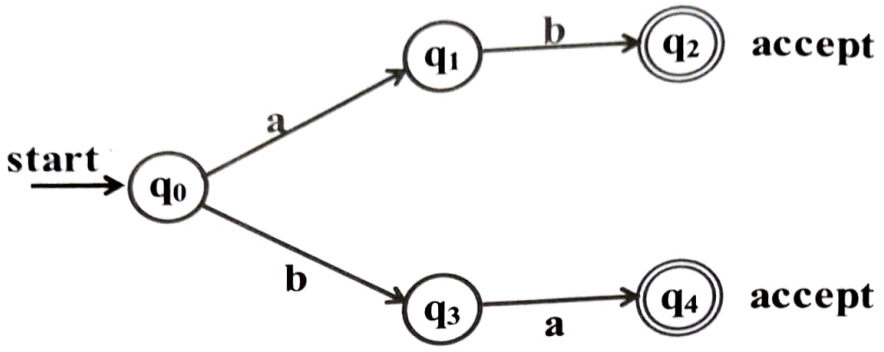
- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{a, b\}$
- q_0 is the start state
- $F = \{q_0, q_1, q_2\}$
- δ is shown using the transition table:

δ	a	b
→*q_0	q_1	q_0
*q_1	q_2	q_0
*q_2	q_2	q_3
q_3	q_3	q_3

- If q_3 and q_6 are the only final states, the DFA accepts strings of a's and b's ending with *aa* or *bb*

Example 2.14: Draw a DFA to accept string of a's and b's ending with *ab* or *ba*.

The **minimized DFA** for the previous problem can be designed as shown below. The minimum string that can be accepted is either *ab* or *ba*. The equivalent DFA can be written as shown below:



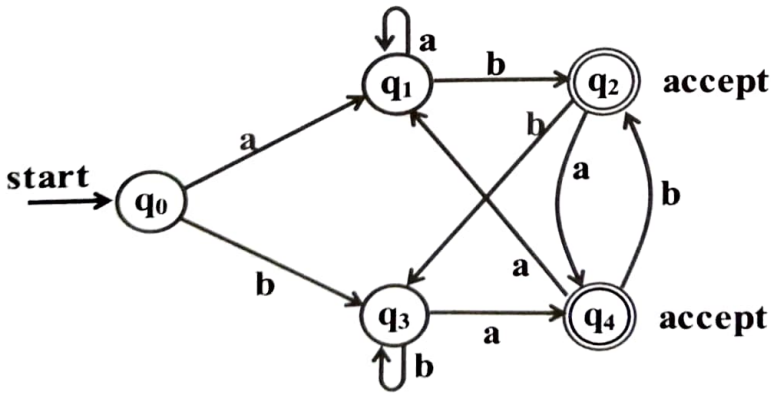
State q_1: This state can accept any number of a's and hence there can be self loop on input symbol *a*. So, $\delta(q_1, a) = q_1$

State q_3: This state can accept any number of b's and hence there can be self loop on input symbol *b*. So, $\delta(q_3, b) = q_3$

State q_2: In this state, if the input symbol is *a*, the string ends with *ba* and hence it should goto state q_4. So, $\delta(q_2, a) = q_4$. If the input symbol is *b*, the string ends with *bb* and hence goto state q_3 where it can accept any number of b's. So, $\delta(q_2, b) = q_3$

State q_4: In this state, if the input symbol is a, the string ends with *aa* and hence it should goto state q_1 where it can accept any number of *a*'s. So, $\delta(q_4, a) = q_1$. If the input symbol is *b*, the string ends with *ab* and hence goto state q_2. So, $\delta(q_4, b) = q_2$

Now, the transition diagram of DFA along with formal definition can be written as shown below:



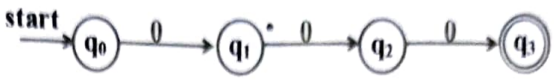
M = (Q, Σ, δ, q_0, F) where

- Q = $\{q_0, q_1, q_2, q_3, q_4\}$
- Σ = {a, b}
- q_0 is the start state
- F = $\{q_2, q_4\}$
- δ is shown below using the transition table:

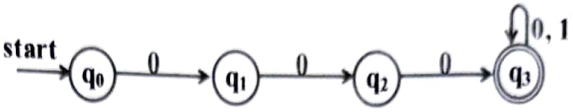
δ	a	b
→ q_0	q_1	q_3
q_1	q_1	q_2
*q_2	q_4	q_3
q_3	q_4	q_3
*q_4	q_1	q_2

Example 2.15: Draw a DFA to accept string of 0's and 1's having three consecutive 0's

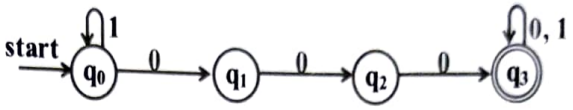
Solution: The minimum string that can be accepted is "000" and it requires four states as shown:



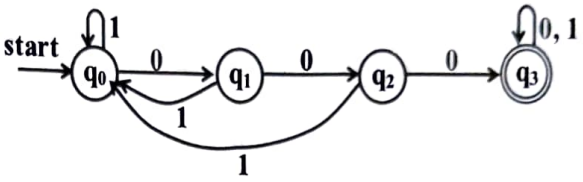
After accepting "000" it can accept any number of 0's and 1's thus having a self loop as shown:



Before accepting a zero it can accept any number of 1's. Thus, there is a self loop at q0 on 1 as shown:



In state q_1 and q_2, if the input symbol is 1, the sequence "000" is broken and hence go back to start state as shown:



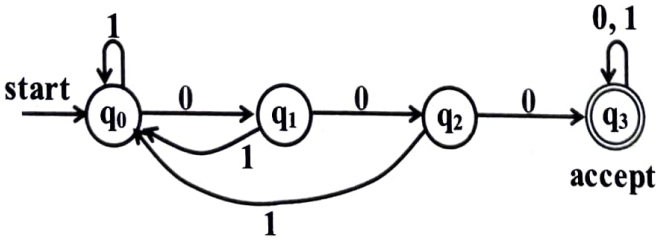
Formally, the above DFA can be written as $\mathbf{M = (Q, \Sigma, \delta, q_0, F)}$ where

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- q_0 is the start state
- $F = \{q_3\}$
- δ is shown below using the transition table:

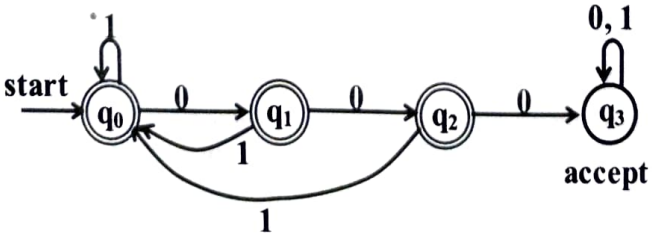
δ	0	1
→ q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_3	q_0
*q_3	q_3	q_3

Example 2.16: Draw a DFA to accept string of 0's and 1's having no 3 consecutive 0's

Solution: First, let us construct a DFA to accept strings of 0's and 1's having three consecutive zeros as in previous example. The DFA is re-written for the sake of convenience as shown below:

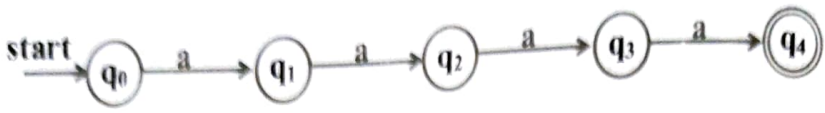


Now, changing the final state to non-final state and non-final states to final state, we get a DFA which accepts strings of 0's and 1's having no three consecutive 0's as shown below:

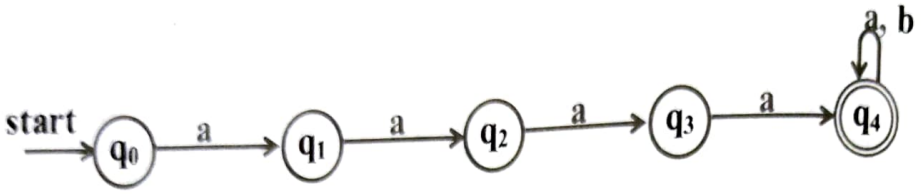


Example 2.17: Draw a DFA to accept string of a's and b's having at least four a's

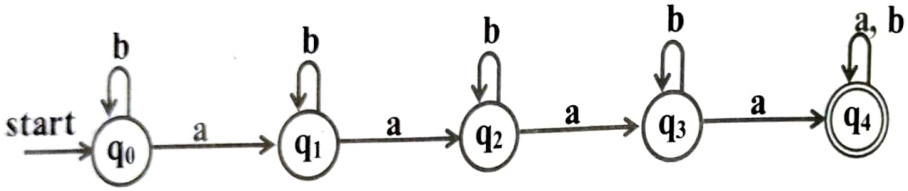
Solution: The minimum string that can be accepted is "aaaa" and require five states. The equivalent DFA can be written as shown below:



After accepting four a's at state q_4, the machine can accept any number of a's and b's and so it should have self loop at q_4 on input symbols *a* and *b* as shown below:



Also just before each four *a*'s the machine can accept any number of b's. So, there should be a self loop at all the states q_0, q_1, q_2 and q_3 on input symbol b as shown below:



Transition Diagram

Formally, the above DFA can be written as M = (Q, Σ, δ, q_0, F) where

- Q = {q_0, q_1, q_2, q_3, q_4}
- Σ = {a, b}
- q_0 is the start state
- F = {q_4}
- δ is shown below using the transition table:

δ	a	b
→ q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_4	q_3
*q_4	q_4	q_4

Note: The language accepted by machine can be represented as:

$$L = \{w : n_a(w) \geq 4, w \in \{a, b\}\}$$

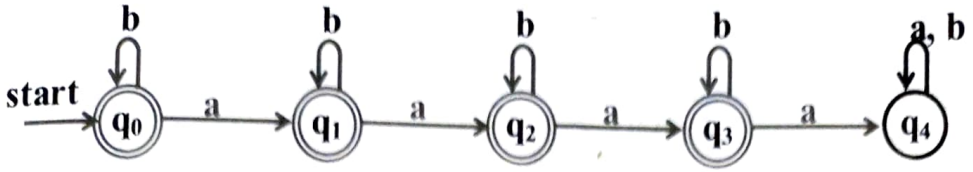
Example 2.18: Draw a DFA to accept string of a's and b's having not more than 3 a's

Solution: Strings of a's and b's having not more than 3 a's can be formally written as:

$$L = \{w : n_a(w) \leq 3, w \in \{a, b\}\}$$

The given language can also be stated as **"Strings of a's and b's with at most three a's.**

It is very easy to construct the machine to accept a string consisting of a's and b's with at least four a's (see previous problem). By taking the complement of this DFA we can construct the machine to accept the given language. The final DFA obtained after complementing (reversing the states of DFA) is shown below:



The language indicates that the machine can accept three a's, two a's, one *a* and even empty string denoted by ε.

Formally, the above DFA can be written as **M = (Q, Σ, δ, q_0, F)** where

- Q = {q_0, q_1, q_2, q_3, q_4}
- Σ = {0, 1}
- q_0 is the start state
- F = {q_0, q_1, q_2, q_3}
- δ is shown below using the transition table:

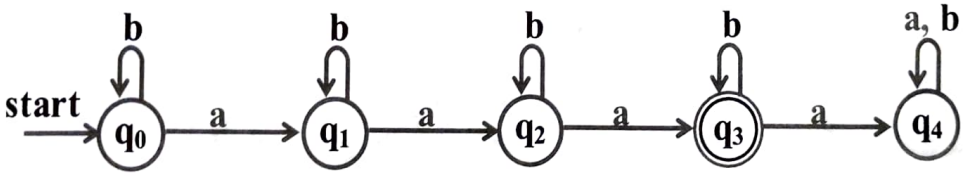
δ	a	b
→*q_0	q_1	q_0
*q_1	q_2	q_1
*q_2	q_3	q_2
*q_3	q_4	q_3
q_4	q_4	q_4

Example 2.19: Draw a DFA to accept string of a's and b's having exactly three a's

Solution: Strings of a's and b's having exactly 3 a's can be formally written as:

$$L = \{w : n_a(w) = 3, w \in \{a, b\}\}$$

The given language can also be obtained by making q_3 as the final state and remaining as non-final states as shown below:



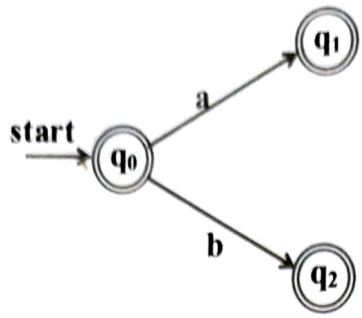
Formally, the above DFA can be written as **M = (Q, Σ, δ, q_0, F)** where

- Q = {q_0, q_1, q_2, q_3, q_4}
- Σ = {0, 1}
- q_0 is the start state
- F = {q_3}
- δ is shown below using the transition table:

δ	a	b
→*q_0	q_1	q_0
*q_1	q_2	q_1
*q_2	q_3	q_2
*q_3	q_4	q_3
q_4	q_4	q_4

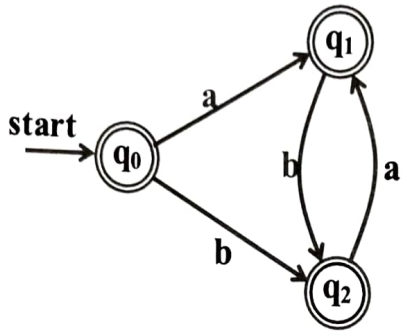
Example 2.20: Draw a DFA to accept string of a's and b's such that no two consecutive characters are same.

Solution: The minimum strings that can be accepted are: ε, a, b. The DFA to accept the above strings can be written as shown in fig:



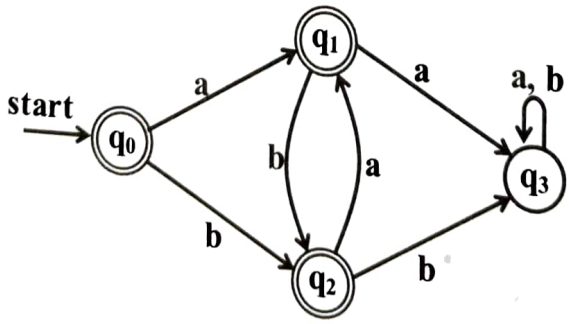
State q_1: Here, if the input symbol is *b*, it has to be accepted changing the state to q_2. The transition is: $\delta(q_1, b) = q_2$

State q_2: Here, if the input symbol is *a*, it has to be accepted changing the state to q_1. The transition is: $\delta(q_2, a) = q_1$.



Now, the DFA can be written as shown in fig:

The transition from state q_1 on input symbol *a* and transition from state q_2 on input symbol *b* is not defined. It indicates that there will be transitions to the trap state as shown below:



Transition Diagram

Formally, the above DFA can be written as **M = (Q, Σ, δ, q_0, F)** where

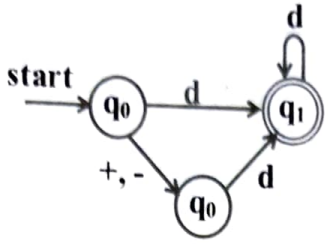
- Q = {q_0, q_1, q_2, q_3}
- Σ = {a, b}
- q_0 is the start state
- F = {q_0, q_1, q_2}
- δ is shown below using the transition table:

δ	a	b
→*q_0	q_1	q_2
*q_1	q_3	q_2
*q_2	q_1	q_3
q_3	q_3	q_3

Note: In this DFA, state q3 is called the **dead state** or **trap state**

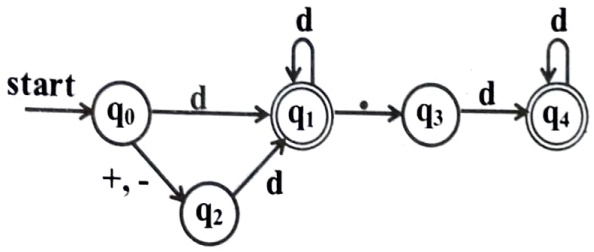
Example 2.21: Draw a DFA to accept an integer with optional sign

Solution: An integer is a string consisting of at least one digit and the equivalent DFA can be written as shown below:

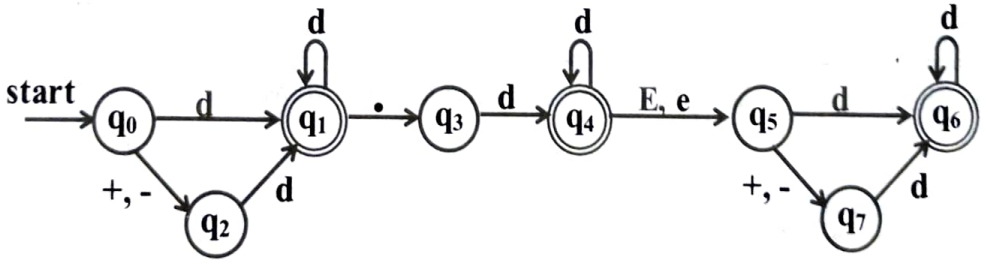


Example 2.22: Draw a DFA to accept a floating point number with optional sign

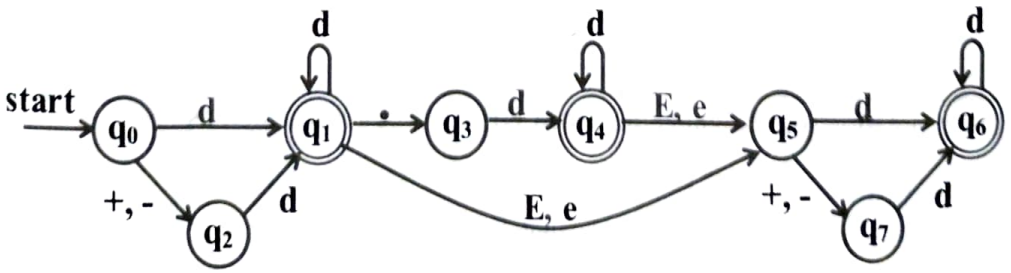
Solution: An integer is a string consisting of at least one digit and the equivalent DFA can be written as shown below:



Note: The DFA accepts all floating-point numbers of the form 123.456



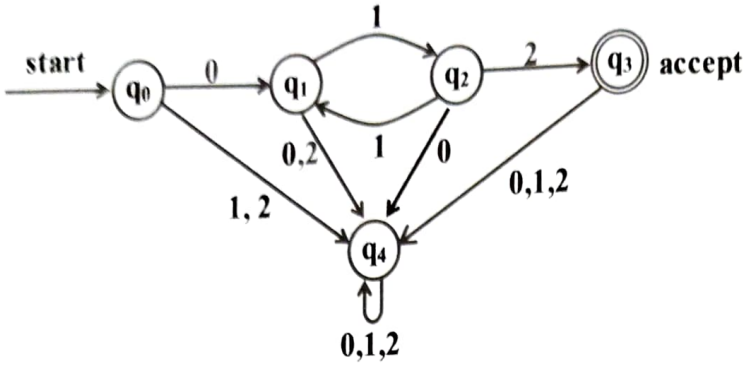
The above floating point number may be followed by an exponent E/e in turn followed by an integer. The equivalent DFA can be written as shown below:



But, an integer followed by E/e which in turn followed by an integer is also a floating-point number. The equivalent DFA can be obtained by having an edge from state q_1 to q_5 as shown below:

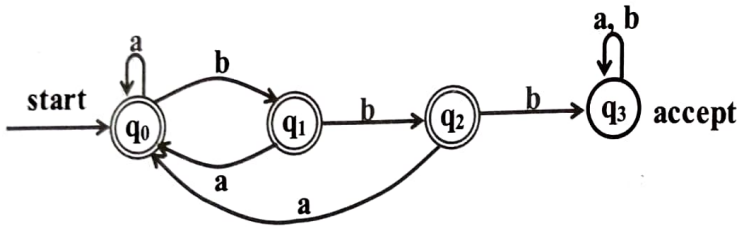
Example 2.23: Obtain a DFA to accept strings of 0's, 1's and 2's beginning with a '0' followed by odd number of 1's and ending with a '2'

Solution: The machine to accept the corresponding string is shown below:



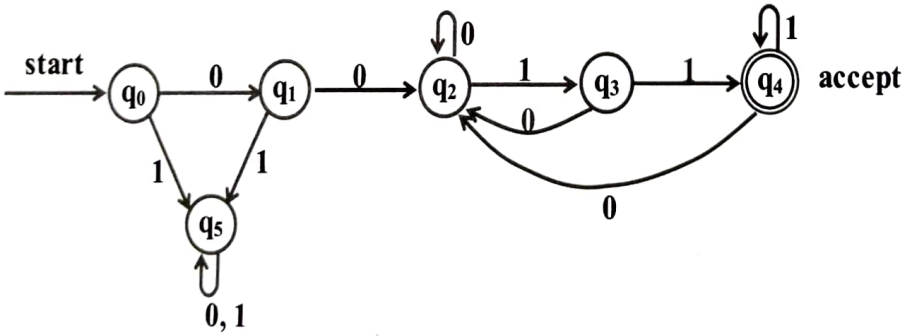
Example 2.24: Obtain a DFA to accept strings of a's and b's with at most two consecutive b's

Solution: The machine to accept strings of a's and b's with at most two consecutive b's is shown below:



Example 2.25: Obtain a DFA to accept strings of 0's and 1's starting with at least two 0's and ending with at least two 1's.

Solution: The machine to accept the corresponding string is shown below:



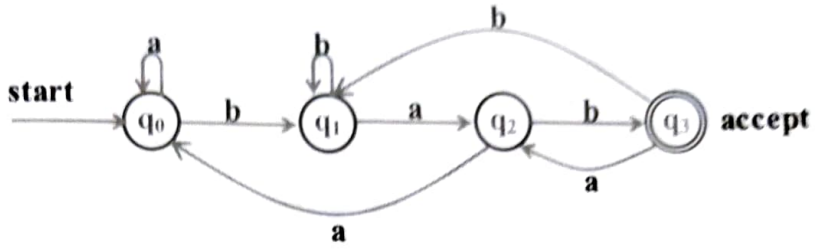
Note: In set notation, the language accepted DFA can be represented as

$$L = \{ w \mid w \in 00(0+1)^*111^* \}$$

which is the language formed by words that begin with at least two 0's and ending with at least two 1's.

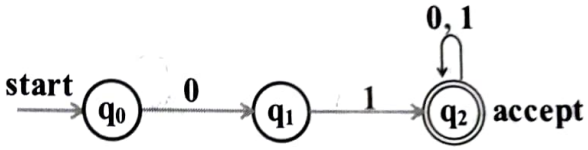
Example 2.26: Obtain a DFA to accept the language $L = \{wbab \mid w \in \{a, b\}^*\}$

Solution: The DFA to accept the language L = {wbab | w ∈ {a, b}*} is shown below:

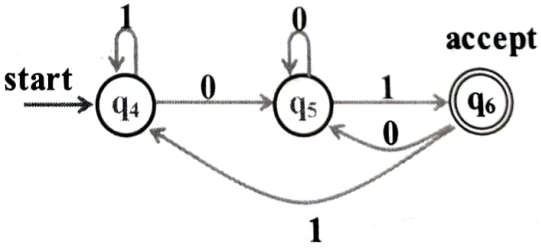


Example 2.27: Draw a DFA to accept set of all strings on the alphabet Σ ={0, 1} that either begins or ends or both with the substring 01.

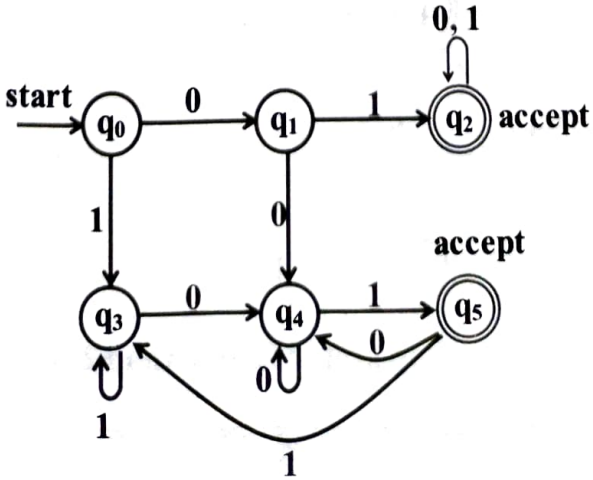
Solution: The DFA to accept strings of 0's and 1's starting with string 01 can be written as shown below: **(Procedure remains same as example 2.7)**



The DFA to accept strings of 0's and 1's ending with string 01 can be written as shown below: **(Procedure remains same as example 2.8)**



The two DFA's can be joined **to accept strings of 0's and 1's beginning with 01 and ending with 01 or both** can be written as shown below:



δ	0	1
→ q0	q1	q3
q1	q4	q2
*q2	q2	q2
q3	q4	q3
q4	q4	q5
*q5	q4	q3

So, formally a DFA can be written as **M = (Q, Σ, δ, q_0, F)** where

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$
- $\Sigma = \{0, 1\}$
- δ is shown above using the transition diagram and transition table
- q_0 is the start state
- $F = \{q_2, q_5\}$

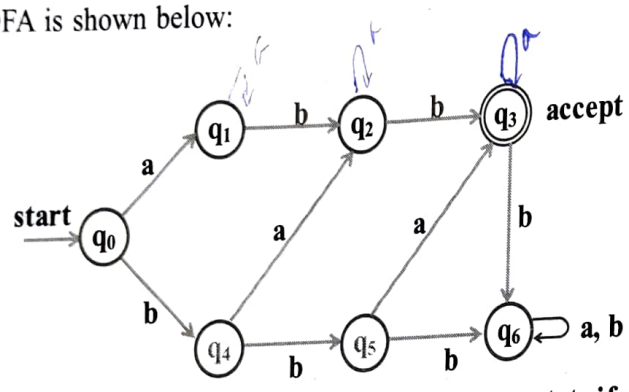
Example 2.28 : Draw a DFA to accept the language $L = \{w: n_a(w) \geq 1, n_b(w) = 2\}$

Solution: Note that the string should have exactly two *b*'s and at least one *a*. So, the minimum string that can be accepted by the DFA may be:

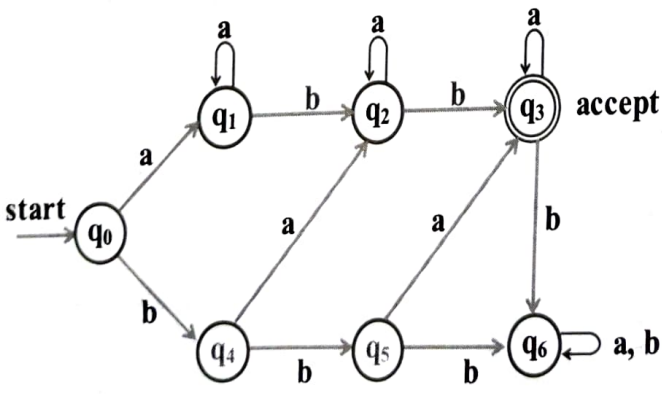
- abb
- bab
- bba

Note: The language can also be interpreted as **"set of all strings with at least one 'a' and exactly two b's**

The initial DFA is shown below:



But, one or more a's can be present in the string. So, the above DFA can be written (using the same steps of designing technique) as shown below:

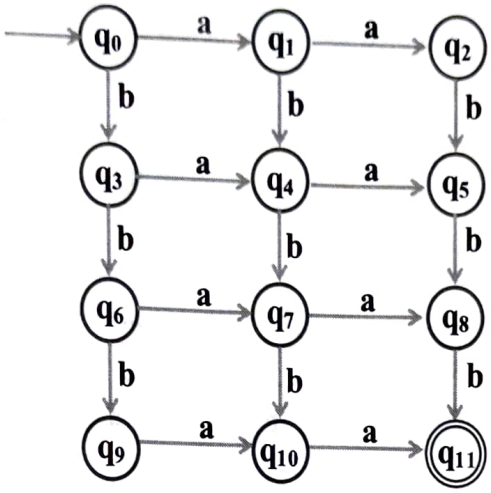


Example 2.29 : Draw a DFA to accept the language $L = \{w: n_a(w) = 2, n_b(w) >= 3\}$

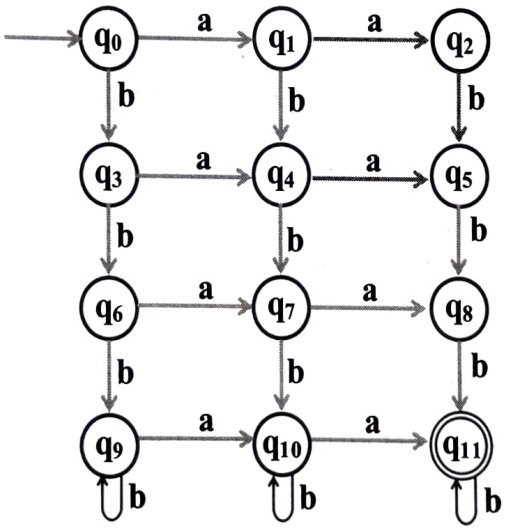
Solution: It is observed from the given language that the string that is accepted by DFA should have exactly two a's and at least three b's. So, the minimum strings that can be accepted are:

- aabbb
- baabb
- bbaab
- bbbaa
- ababb
- babab
- bbaba
- abbab
- babba
- abbba

The initial DFA that accepts all the above strings is shown below:



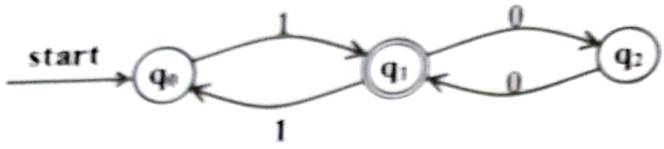
Since, minimum three b's should be there, after three b's we can consume any number of b's. So, the above DFA can be written as shown below:



Note: From states q_2, q_5, q_8 and q_{11} we can have transitions from input symbol *a* to trap state.

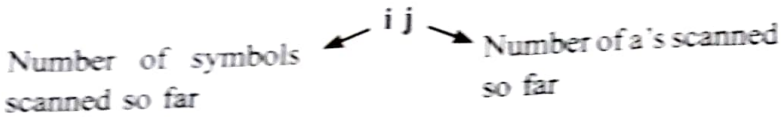
Example 2.30: Draw a DFA to accept the language: L = {w: w has odd number of 1's and followed by even number of 0's)

Solution: The DFA to accept strings of 0's and 1's such that the string has odd number of 1's and followed by even number of 0's is shown below:

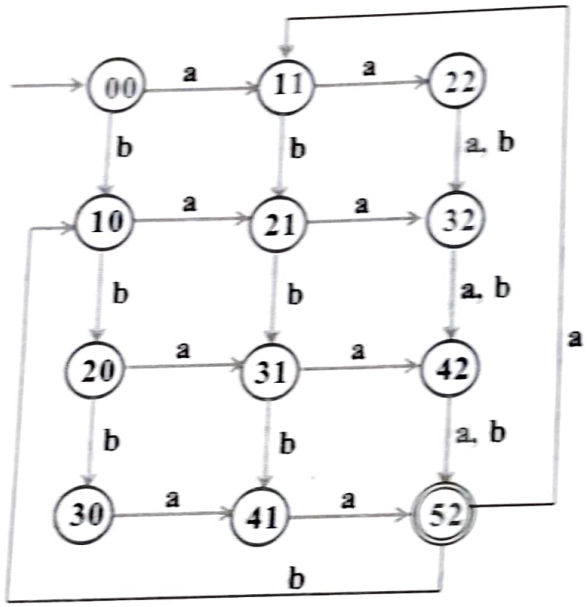


Example 2.31: Obtain a DFA to accepts strings of a's and b's such that each block of 3 consecutive symbols have at least two a's

Solution: This DFA should keep track of number of symbols scanned for and number of a's. Hence, each state of DFA is represented as shown below:



The complete DFA is shown below:



Note: The transitions from states 30 and 41 on input symbol *b* are not defined and hence those transitions indicate that they are the transitions to the trap state. The trap state is not shown in the diagram.

2.6.2 Divisible by k Problems

In this section, let us discuss a method of constructing DFA that divide a number by *k*. For these types of problems, the transitions can be obtained using the following relation:

$$\delta(q_i, a) = q_j \text{ where } j = (r \; i + d) \bmod k$$

- *r* is the radix of input. For binary r = 2
- *i* is the remainder obtained after dividing by k
- *d* represent digits. For binary d = { 0, 1}
- *k* is divisor

The steps to be followed to find FA using these types of problems is shown below:

Step 1: Identify the radix, input alphabets and the divider *k*

Step 2: Compute the possible remainders: These remainders represent the states of DFA

Step 3: Find the transitions using $\delta(q_i, a) = q_j$ where $j = (r\ i + d) \bmod k$

Step 4: Construct the DFA using the transitions obtained in step 3.

Example 2.32: Construct a DFA which accepts strings of 0's and 1's where the value of each string is represented as a binary number. Only the strings representing zero modulo five should be accepted. For example, 0000,0101,1010,1111 etc. should be accepted.

Solution: The DFA can be obtained as shown below:

Step 1: Identify the radix, input alphabets and the divisor k: In this case, r = 2, d = {0, 1}, k = 5

Step 2: Compute the possible remainders: After dividing by k, the possible remainders are: i = 0, 1, 2, 3, 4

Step 3: Compute transitions: The transitions can be computed using the following relation:

$\delta(q_i, a) = q_j$ where $j = (r\ i + d) \bmod k$ with r = 2 and k = 5

So, $j = (2i + d) \bmod 5$

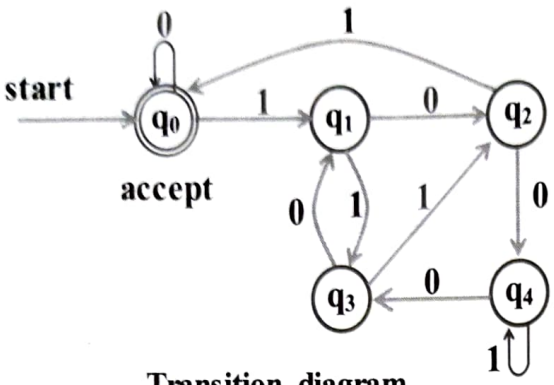
remainder	d	$(2*i+d) \bmod 5 = j$	$\delta(q_i,d) = q_j$
i = 0	0	$(2*0+0) \bmod 5 = 0$	$\delta(q_0,0) = q_0$
	1	$(2*0+1) \bmod 5 = 1$	$\delta(q_0,1) = q_1$
i = 1	0	$(2*1+0) \bmod 5 = 2$	$\delta(q_1,0) = q_2$
	1	$(2*1+1) \bmod 5 = 3$	$\delta(q_1,1) = q_3$
i = 2	0	$(2*2+0) \bmod 5 = 4$	$\delta(q_2,0) = q_4$
	1	$(2*2+1) \bmod 5 = 0$	$\delta(q_2,1) = q_0$
i = 3	0	$(2*3+0) \bmod 5 = 1$	$\delta(q_3,0) = q_1$
	1	$(2*3+1) \bmod 5 = 2$	$\delta(q_3,1) = q_2$
i = 4	0	$(2*4+0) \bmod 5 = 3$	$\delta(q_4,0) = q_3$
	1	$(2*4+1) \bmod 5 = 4$	$\delta(q_4,1) = q_4$

Transitions of resulting DFA

Step 4: The DFA can be defined as $M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$

- q_0 is the start state
- $F = \{q_0\}$
- δ is shown below using the transition diagram and transition table as shown below:



Transition diagram

δ	0	1
→*q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_0
q_3	q_1	q_2
q_4	q_3	q_4

Transition Table

Note: Observe that for modulo 5, number of states of DFA will be 5.

In general, for modulo k, number of states of DFA will be k.

Example 2.33: Draw a DFA to accept decimal strings divisible by 3 **(VTU July 2007)**

Solution: The DFA can be obtained as shown below:

Step 1: Identify the radix, input alphabets and the divisor k: In this case, r = 10,

d = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, k = 3

Step 2: Compute the possible remainders: After dividing any decimal number by 3, results in following remainders:

i = 0, 1, 2 which implies q_0, q_1 and q_2 are the states of DFA.

Step 3: Compute transitions: The transitions can be computed using the following relation: $\delta(q_i, a) = q_j$ where $j = (r\,i + d) \bmod k$

with r = 10 and k = 3

So, $j = (2i + d) \bmod 3$

Note: For the sake of convenience, let us group the digits from 0 to 9 based on the remainders we get after dividing by 3 as shown below:

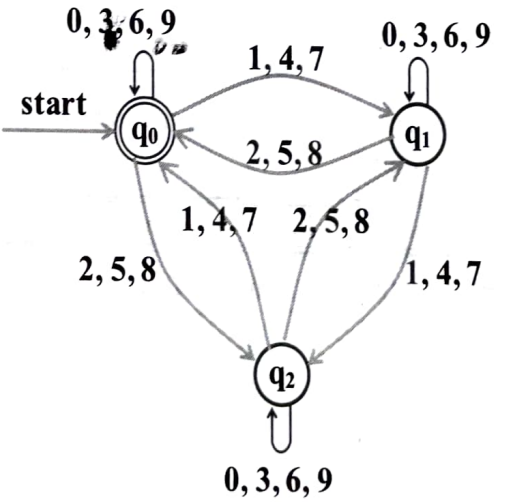
- {0, 3, 6, 9} with 0 as the remainder. So, **δ from {0, 3, 6, 9} ⇒ δ from {0}**
- {1, 4, 7} with 1 as the remainder. So, **δ from {1, 4, 7} ⇒ δ from {1}**
- {2, 5, 8} with 2 as the remainder. So, **δ from {2, 5, 8} ⇒ δ from {2}**

remainder	d	(2* i +d) mod 3 = j	$\delta(q_i,d) = q_j$		
i = 0	0	(10*0 + 0) mod 3 = 0	$\delta(q_0,0) = q_0$	⇒	$\delta(q_0,\{0,3,6,9\}) = q_0$
	1	(10*0 + 1) mod 3 = 1	$\delta(q_0,1) = q_1$	⇒	$\delta(q_0,\{1,4,7\}) = q_1$
	2	(10*0 + 2) mod 3 = 2	$\delta(q_0,2) = q_2$	⇒	$\delta(q_0,\{2,5,8\}) = q_2$
i = 1	0	(10*1 + 0) mod 3 = 1	$\delta(q_1,0) = q_1$	⇒	$\delta(q_1,\{0,3,6,9\}) = q_1$
	1	(10*1 + 1) mod 3 = 2	$\delta(q_1,1) = q_2$	⇒	$\delta(q_1,\{1,4,7\}) = q_2$
	2	(10*1 + 2) mod 3 = 0	$\delta(q_1,2) = q_0$	⇒	$\delta(q_1,\{2,5,8\}) = q_0$
i = 2	0	(10*2 + 0) mod 3 = 2	$\delta(q_2,0) = q_2$	⇒	$\delta(q_2,\{0,3,6,9\}) = q_2$
	1	(10*2 + 1) mod 3 = 0	$\delta(q_2,1) = q_0$	⇒	$\delta(q_2,\{1,4,7\}) = q_0$
	2	(10*2 + 2) mod 3 = 1	$\delta(q_2,2) = q_1$	⇒	$\delta(q_2,\{2,5,8\}) = q_1$

Transitions of DFA

Step 4: The DFA can be defined as **M = (Q, Σ, δ, q_0, F)** where

- **Q** = $\{q_0, q_1, q_2\}$
- **Σ** = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- q_0 is the start state
- **F** = $\{q_0\}$
- **δ** is shown below using the transition diagram as shown below:



2.6.3 String Length Mod k problems

Now, let us see **"What is the general method to solve "String length mod k problems?"** The general method is shown below:

Step 1: Find the remainders obtained by dividing the string length by k. The remainders obtained after dividing by k are shown below:

0, 1, 2....k-1

Ex1: when the divisor is k = 2 the remainders are: i = 0, 1

Ex2: when the divisor is k = 3 the remainders are: i = 0, 1, 2

Ex3: when the divisor is k = 4 the remainders are: i = 0, 1, 2, 3

Step 2: Identify the transitions. The transitions can be obtained using the following relation:

$$\delta(q_i, a) = q_{(i + 1 \bmod k)}$$

where

- *k* is the divisor
- *i* represent the remainders 0, 1, 2.....k-1 obtained after dividing by k

Step 3: Identify the start state and final state: q_0 is always the start state. Suppose, the language to be accepted is:

$L = \{ w : |w| \bmod k = j\}$ then

↓

q_j is the final state.

Ex1: If $L = \{ w : |w| \bmod k = 0$ for some string $w\}$, then q_0 is the final state.

Ex2: If $L = \{ w : |w| \bmod k = 3$ for some string $w\}$, then q_3 is the final state.

Ex3: If $L = \{ w : |w| \bmod k = 5$ for some string $w\}$, then q_5 is the final state.

Step 4: Write the DFA. The DFA can be written using the transitions given in step 2 with *start state* and *final state* as obtained in step 3.

Example 2.34: Obtain a DFA to accept strings of even number of a's

Solution: The language accepted by the DFA can be written as shown below:

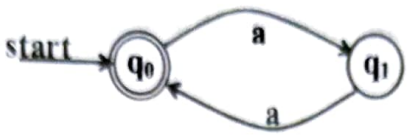
$L = \{w : |w| \bmod 2 = 0\}$ with q_0 as the start state and q_0 as final state

The transitions can be obtained using the following relation:

$\delta(q_i, a) = q_{(i + 1 \bmod k)}$ where k = 2, and i = 0, 1 (remainders after dividing by k) as shown below:

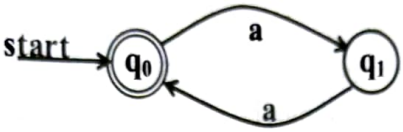
i	$\delta(q_i, a) = q(i + 1 \bmod 2)$
0	$\delta(q_0, a) = q(0 + 1 \bmod 2) = q_1$
1	$\delta(q_1, a) = q_{(1 + 1 \bmod 2)} = q_0$

So, the transition diagram to accept even number of a's is shown below:

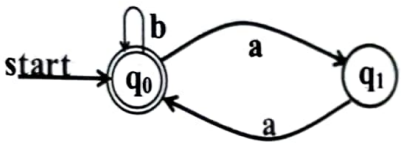


Example 2.35: Obtain a DFA to accept L = {w ∈ {a, b} : every 'a' region in *w* is of even length

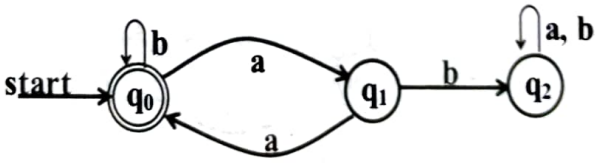
Solution: The language accepted by the DFA can have even number of a's as shown below:



But, immediately after accepting even number of a's we can accept any number of b's. The state q0 is the place where even number of a's are accepted. So, that is the place where it can accept any number of b's by having a self-loop on input symbol b as shown below:



Since there is no transition defined from state q_1 on input symbol *b*, it indicates that there is a transition to the trap state as shown below:



Formally, the above DFA can be defined as M = (Q, Σ, δ, q_0, F) where

- Q = {q_0, q_1, q_2}
- Σ = {a, b}
- q_0 is the start state
- F = {q_0}
- δ is shown below using the transition table:

δ	a	b
→*q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_2	q_2

Example 2.36: Obtain a DFA to accept strings of a's and b's having even number of symbols.

Solution: The language accepted by the DFA can be written as shown below:

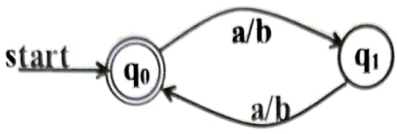
L = {w : |w| mod 2 = 0} with q_0 as the start state and q_0 as final state

The transitions can be obtained using the following relation:

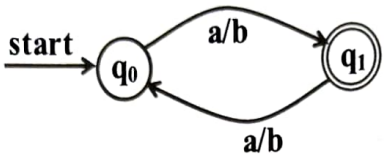
$\delta(q_i, a) = q(i + 1 \bmod k)$ where k = 2, and i = 0, 1 (remainders after dividing by k) as shown below:

i	$\delta(q_i, a) = q(i + 1 \bmod 2)$
0	$\delta(q_0, a) = q(0 + 1 \bmod 2) = q_1$
1	$\delta(q_1, a) = q(1 + 1 \bmod 2) = q_0$

So, the transition diagram to accept even number of symbols is shown below:

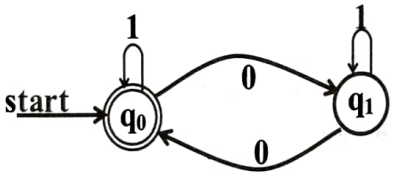


Note: By changing the final state to non-final state and non-final state to final state, we get strings of a's and b's having odd number of symbols



Example 2.37: Obtain a DFA to accept strings of even number of 0's where $\Sigma = \{0, 1\}$

Solution: The transitions remains same as previous problem with q_0 as the start state and q_0 as the final state. Since there is no restriction on number of 1's we can generate any number of 1's both at q_0 and q_1 states. The final DFA can be written as shown below:



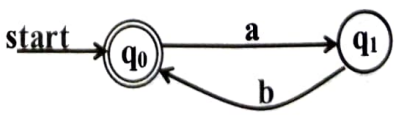
For example, the following strings of 0's and 1's are accepted by the DFA:

010, 0110, 10101, 111011011, 110111011, etc.

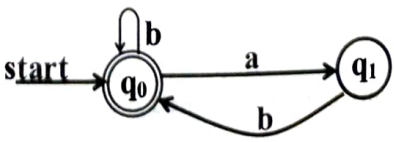
Note that all the above strings have only even number of 0's, but there is no restriction number of 1's:

Example 2.38: Obtain a DFA to accept the language L = {w ∈{a, b} : every *a* is immediately followed by a *b*.

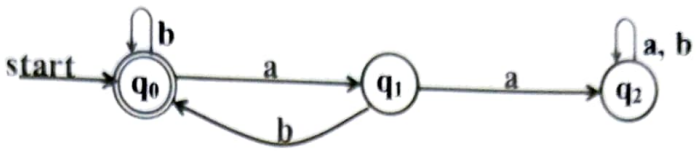
Solution: The language consists of strings of a's and b's such that every a is immediately followed by a *b*. The transition diagram can be written as shown:



Observe that every *a* followed by a *b* is accepted. But, at state q_0, it can accept any number of *b*'s resulting in a self-loop and the DFA can be written as shown:



In state q_1 the transition is not defined for input symbol *a* and it means that there is a transition to the trap state. The final transition diagram can be written as shown below:



Formally, the above DFA can be defined as **M = (Q, Σ, δ, q_0, F)** where

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- q_0 is the start state
- $F = \{q_0\}$
- δ is shown below using the transition table:

δ	a	b
→*q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_2

Example 2.39: Obtain a DFA to accept L = { w ∈ {0, 1} : w has odd parity }

Solution: Odd parity indicates that odd number of 1's should be there. The language accepted by the DFA can be written as shown below:

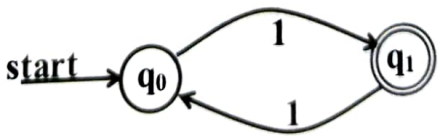
L = {w : |w| mod 2 = 1} with q_0 as the start state and q_1 as final state

The transitions can be obtained using the following relation:

$\delta(q_i, a) = q(i + 1 \bmod k)$ where k = 2, and i = 0, 1 (remainders after dividing by k) as shown below:

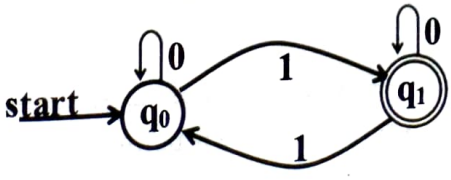
i	**$\delta(q_i, a) = q(i + 1 \bmod 2)$**
0	$\delta(q_0, a) = q(0 + 1 \bmod 2) = q_1$
1	$\delta(q_1, a) = q(1 + 1 \bmod 2) = q_0$

So, the transition diagram to accept odd parity is shown below:



But, in state q_0 and q_1, any number of 0's can be generated thus having self loop at both states. The resulting DFA can be written as shown below:

Transition diagram

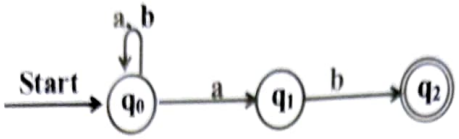


δ	0	1
→*q_0	q_0	q_1
q_1	q_1	q_0

Example 2.51: Obtain an NFA to accept strings of a's and b's ending with *ab*.

Design: The minimum string that can be accepted by NFA is the string *ab*. Since, it has two symbols it requires three states as shown below:

Before accepting *ab*, it can accept any number of a's and b's in state q_0. This is possible by having a self-loop on *a* and *b* at state q_0 as shown below:



The above NFA accepts strings of a's and b's ending with *ab* and it is formally defined as:

$$M = (Q, \Sigma, \delta, q_0, F)$$

where

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- q_0 is the start state
- $F = \{q2\}$
- δ is shown using the transition table:

δ	a	b
→ q_0	$\{q_0, q_1\}$	q_0
q_1	ϕ	q_2
*q_2	ϕ	ϕ

Now, let us **"Show the sequence of moves made by the DFA for the string aaab and aaa"**

Input string aaab: The sequence of moves made by the NFA for the string "aaab" is shown below:

(Initial configuration)

$(q_0, aaab) \vdash (\{q_0, q_1\}, aab)$
$\vdash (\{q_0, q_1\}, ab)$
$\vdash (\{q_0, q_1\}, b)$
$\vdash (\{q_0, q_2\}, \varepsilon)$

(Final configuration)

Observe that in the final configuration, the set $\{q_0, q_2\}$ has a final state and so the string is accepted.

Input string aaa: The sequence of moves made by the NFA for the string "aaa" is shown below:

(Initial configuration)

$(q_0, aaa) \vdash (\{q_0, q_1\}, aa)$
$\vdash (\{q_0, q_1\}, a)$
$\vdash (\{q_0, q_1\}, \varepsilon)$

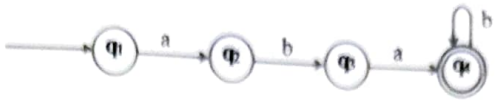
(Final configuration)

Observe that in the final configuration, the $\{q_0, q_1\}$ does not have a final state and so the string is rejected by the machine.

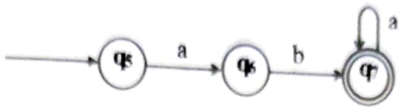
Example 2.52: Obtain an NFA to accept the language

$$L = \{w \mid w \in abab^n \text{ or } aba^n \text{ where } n \geq 0\}$$

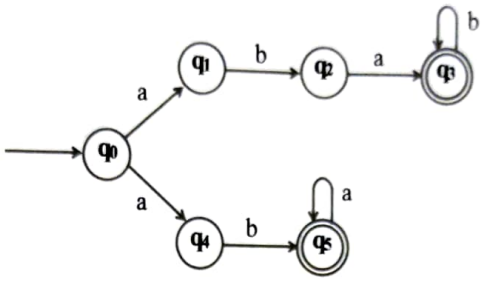
Solution: The machine to accept $abab^n$ where $n \geq 0$ is shown below:



The machine to accept aba^n where $n \geq 0$ is shown below:

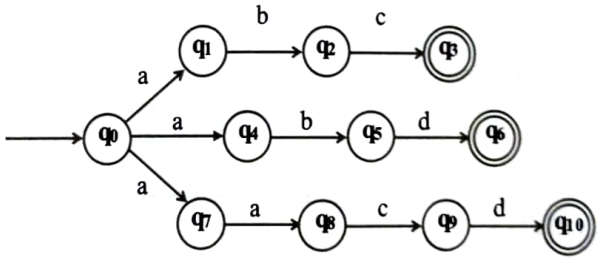


Since both the machines accept a as the first input symbol, the states q1 and q5 can be merged into a single state and the machine to accept either $abab^n$ or aba^n where $n \geq 0$ is shown below:



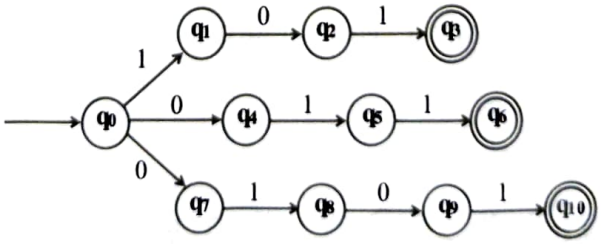
Example 2.53: Design an NFA to recognize the following keywords *abc*, *abd* and *aacd*.

Solution: An NFA to recognize the keywords *abc, abd* and *aacd* is shown below:



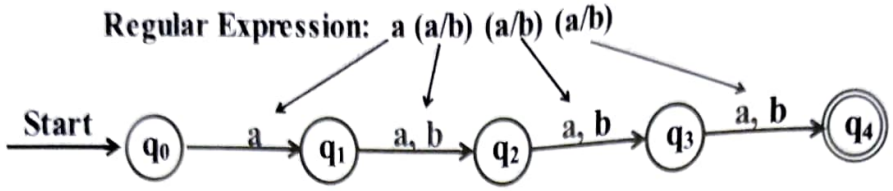
Example 2.54: Design an NFA to recognize the following set of strings *101* and *011* and 0101

Solution: An NFA to recognize the strings *101, 011* and *0101* is shown below:

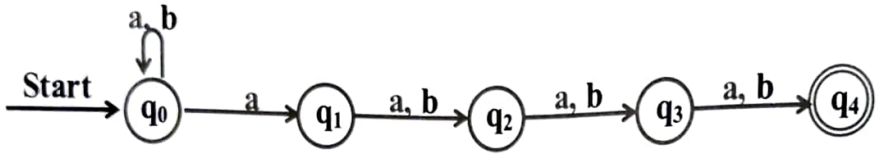


Example 2.55: Obtain an NFA to accept strings of a's and b's such that fourth symbol from the right side (last side) is *a*

Solution: The minimum string that can be accepted by the NFA such that fourth symbol from the right end is *a* can be written as "a (a/b) (a/b) (a/b)" and the equivalent NFA can be written as shown below:



Just before the string "a (a/b) (a/b) (a/b)" the machine can accept any number of b's in state q_0. This can be done by having a self-loop at q_0 as shown below:



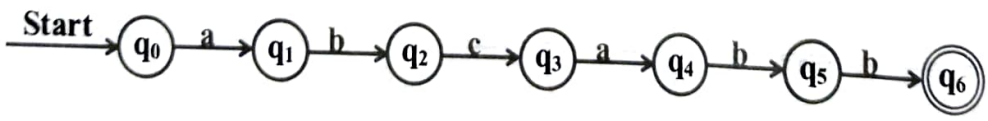
Formally, the machine can be written as: $\mathbf{M = (Q, \Sigma, \delta, q_0, F)}$ where

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{a, b\}$
- q_0 is the start state
- $F = \{q_4\}$
- δ is shown using the transition table:

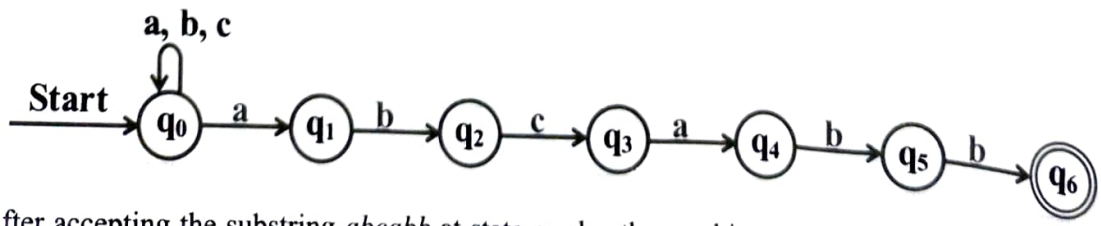
δ	a	b
→ q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	q_2
q_2	q_3	q_3
q_3	q_4	q_4
*q_4	ϕ	ϕ

Example 2.56: Obtain an NFA to accept L = { w ∈ {a, b, c}* : x and y ∈ {a, b, c}* and w = x abcabb y

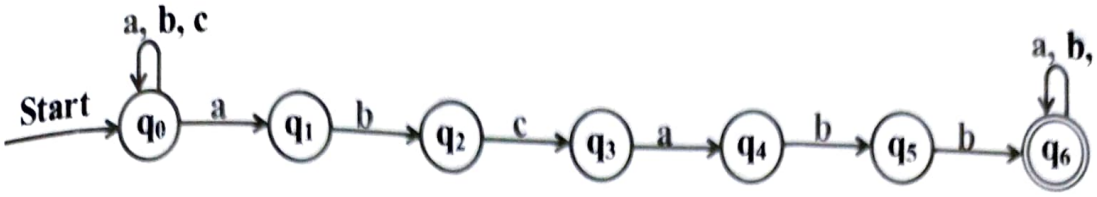
Solution: The minimum string that can be accepted by NFA is the substring "abcabb" and has 6 symbols. So, we require 7 states to accept the string and the finite automaton can be written as shown below:



Before accepting the substring *abcabb* the machine can accept any combination of a's, b's and c's. This can be achieved by having a self-loop at q_0 on input symbols *a*, *b* and *c* as shown below:



After accepting the substring *abcabb* at state q_6 also the machine can accept any combination of a's, b's and c's as shown below:



NFA to accept strings of a's, b's and c's having a substring *abcabb*

NFA to accept strings of a's, b's and c's having a substring abcabb

2.8 Conversion from NFA to DFA

Constructing an NFA is an efficient mechanism to describe some complicated languages concisely. Practically, non-deterministic machines will not exist. So, we convert an NFA into a DFA. Now, let us ***"Describe the method to convert an NFA into DFA"*** The procedure is shown below:

Procedure NFA_DFA: Let $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ be an NFA and accepts the language $L(M_N)$. There should be an equivalent DFA $M_D = (Q_D, \Sigma, \delta_D, q_0, F_D)$ such that $L(M_D) = L(M_N)$. The procedure to convert an NFA to its equivalent DFA is shown below:

Step 1: Identify the start state of DFA: Since q_0 is the start state of NFA, $\{q_0\}$ is the start of DFA.

Step 2: Identify the alphabets of DFA: The input alphabets of NFA are the input alphabets of DFA. For example, if $\Sigma = \{a, b\}$ are the alphabets of NFA, then they are the alphabets of DFA.

Step 3: Identify the transitions δ_D of DFA: For each state $\{q_i, q_j, \ldots q_k\}$ in Q_D and for each input symbol *a* in Σ, the transition can be obtained as shown below:

$$\delta_D(\{q_i, q_j, \ldots q_k\}, a) = \delta_N(q_i, a) \cup \delta_N(q_j, a) \cup \ldots\ldots\ldots \delta_N(q_k, a)$$

$$= \{q_l, q_m, \ldots\ldots q_n\}$$

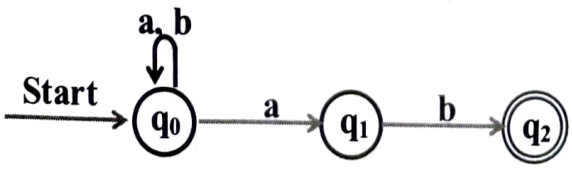
- Add the state $\{q_l, q_m, \ldots\ldots q_n\}$ to Q_D, if it is not already in Q_D
- Add the transition from $\{q_i, q_j, \ldots q_k\}$ to $\{q_l, q_m, \ldots\ldots q_n\}$ on the input symbol a.

Note: The step 3 has to be repeated for each state that is added to Q_D

Step 4: Identify the final states of DFA: If $\{q_i, q_j, \ldots q_k\}$ is a state in Q_D and if one of $q_i, q_j, \ldots q_k$ is the final state of NFA, then $\{q_i, q_j, \ldots q_k\}$ will be the final state of DFA.

Thus, a DFA can be obtained using NFA.

Example 2.57: Convert the following NFA to its equivalent DFA



Solution: The transition table for the above NFA can be written shown below:

δ	a	b
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$
$*q_2$	ϕ	ϕ

Step1: *Identify the start state of DFA*: Since q_0 is the start state of NFA, $[q_0]$ is the start state of DFA.

Step 2: *Identify the alphabets of DFA*: The input alphabets of NFA are the input alphabets of DFA. So, $\Sigma = \{a, b\}$

Step 3: *Identify QD which are the sates of DFA*: Start from the start state q_0 and find the transitions as shown below:

For state q_0:

Input symbol = a

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

Input symbol = b

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

For state $\{q_0, q_1\}$:

Input symbol = a

$\delta_D(\{q_0,q_1\}, a) = \delta_N(\{q_0,q_1\}, a)$

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

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

$= \{q_0, q_1\}$

Input symbol = b

$\delta_D(\{q_0,q_1\}, b) = \delta_N(\{q_0,q_1\}, b)$

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

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

$= \{q_0, q_2\}$

For state $\{q_0, q_2\}$:

Input symbol = a

$\delta_D(\{q_0,q_2\}, a) = \delta_N(\{q_0,q_2\}, a)$

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

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

$= \{q_0, q_2\}$

Input symbol = b

$\delta_D(\{q_0,q_2\}, b) = \delta_N(\{q_0,q_2\}, b)$

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

$= \{q_0\} \cup \phi$

$= \{q_0\}$

Since, no new state is generated the procedure is terminated.

Step 4: *Identify the final states of DFA:* Since q_2 is the final state of NFA in the above set, wherever q_2 is present as an element, the corresponding set is the final state of DFA. So,

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

Now, all the above transitions can be represented using transition table as shown below:

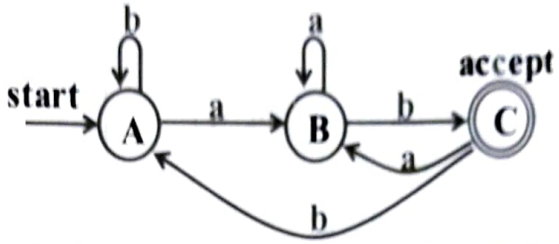
δ	a	b
→ {q0}	{q0,q1}	{q0}
{q0, q1}	{q0,q1}	{q0, q2}
*{q0, q2}	{q0, q1}	{q0}

⇨ by renaming the states of DFA as A, B, C

δ	a	b
→ A	B	A
B	B	C
*C	B	A

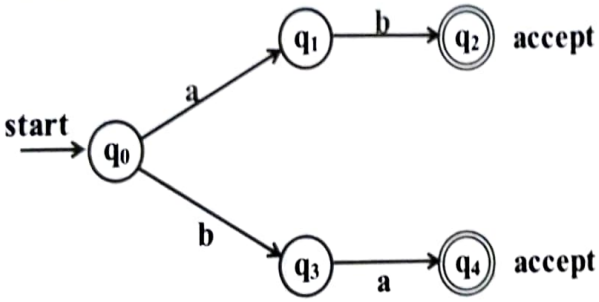
So, the final DFA is given by M = (Q, Σ, δ, q_0, F) where

- Q = {A, B, C}
- Σ = {a, b}
- q_0 = A is the start state
- F = {C} is final state
- δ is shown using transition diagram:

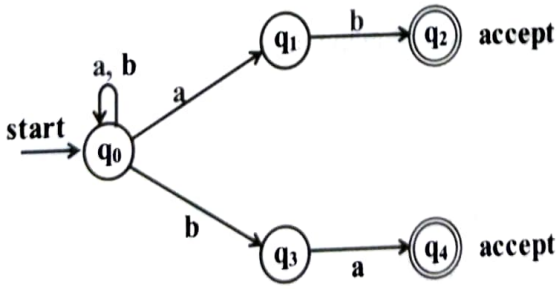


Example 2.58: Obtain an NFA to accept strings of a's and b's ending with ab or ba. Convert this NFA to its equivalent DFA.

Solution: The minimum string that can be accepted by an NFA is string consisting of either ab or *ba*. The equivalent machine is shown:



The string *ab* or *ba* can be preceded by any number of a's and b's. This is possible by having a self-loop at q_0 on input symbols *a* and *b*. The equivalent NFA shown below:



Now, the NFA is formally defined as: **M = (Q, Σ, δ, q_0, F)** where

- Q = {q_0, q_1, q_2, q_3, q_4}
- Σ = {a, b}
- q_0 is the start state
- F = {q_4}
- δ is shown using the transition table:

δ	a	b
→ q_0	q_0,q_1	q_0,q_3
q_1	ϕ	q_2
*q_2	ϕ	ϕ
q_3	q_4	ϕ
*q_4	ϕ	ϕ

Conversion from NFA to DFA: Now, let us see the conversion of above NFA into its equivalent DFA.

Step 1: Identify the start state of DFA: Since q_0 is the start state of NFA, $\{q_0\}$ is the start st of DFA. So, $Q_D = \{q_0\}$

Step 2: Identify the alphabets of DFA: The input alphabets of NFA are the input alphabets DFA. So, $\Sigma = \{a, b\}$

Step 3: Identify the transitions of DFA: Start from the start state $\{q_0\}$ and find the transitio as shown below:

For state q_0:

Input symbol = a

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

Input symbol = b

$\delta_D(\{q_0\}, b) = \{q_0, q_3\}$

For state $\{q_0, q_1\}$:

Input symbol = a

$$\delta_D(\{q_0,q_1\}, a) = \delta_N(\{q_0,q_1\}, a)$$
$$= \delta_N(\{q_0\}, a) \cup \delta_N(\{q_1\}, a)$$
$$= \{q_0, q_1\} \cup \phi$$
$$= \{q_0, q_1\}$$

Input symbol = b

$$\delta_D(\{q_0,q_1\}, b) = \delta_N(\{q_0,q_1\}, b)$$
$$= \delta_N(\{q_0\}, b) \cup \delta_N(\{q_1\}, b)$$
$$= \{q_0, q_3\} \cup \{q_2\}$$
$$= \{q_0, q_2, q_3\}$$

For state $\{q_0, q_3\}$:

Input symbol = a

$$\delta_D(\{q_0,q_3\}, a) = \delta_N(\{q_0,q_3\}, a)$$
$$= \delta_N(\{q_0\}, a) \cup \delta_N(\{q_3\}, a)$$
$$= \{q_0, q_2\} \cup \{q_4\}$$
$$= \{q_0, q_1, q_4\}$$

Input symbol = b

$$\delta_D(\{q_0,q_3\}, b) = \delta_N(\{q_0,q_3\}, b)$$
$$= \delta_N(\{q_0\}, b) \cup \delta_N(\{q_3\}, b)$$
$$= \{q_0, q_3\} \cup \phi$$
$$= \{q_0, q_3\}$$

On similar lines, the reader is supposed to find the transitions for other states specified in Q_D. The reader is advised to verify the following answers:

$\delta_D(\{q_0, q_2, q_3\}, a) = \{q_0, q_1, q_4\}$

$\delta_D(\{q_0, q_2, q_3\}, b) = \{q_0, q_3\}$

$\delta_D(\{q_0, q_1, q_4\}, a) = \{q_0, q_1\}$

$\delta_D(\{q_0, q_1, q_4\}, b) = \{q_0, q_2, q_3\}$

The final DFA obtained along with transition diagram and transition table is shown below:

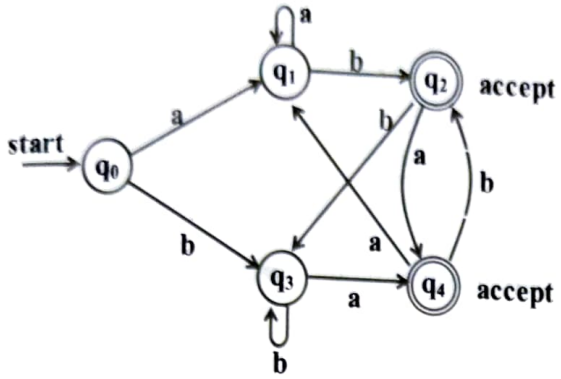
δ	a	b
→ $\{q_0\}$	$\{q_0,q_1\}$	$\{q_0, q_3\}$
$\{q_0, q_1\}$	$\{q_0,q_1\}$	$\{q_0, q_2, q_3\}$
* $\{q_0, q_3\}$	$\{q_0, q_1,q_4\}$	$\{q_0, q_3\}$
* $\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_4\}$	$\{q_0, q_3\}$
$\{q_0, q_1, q_4\}$	$\{q_0, q_1\}$	$\{q_0, q_2, q_3\}$

The states of the above DFA are:

$$\{q_0\}, \{q_0,q_1\}, \{q_0, q_2, q_3\}, \{q_0, q_3\}, \{q_0, q_1,q_4\}$$

By renaming: q_0, q_1, q_2, q_3, q_4

The final transition diagram and transition table are shown below:



δ	a	b
→ q_0	q_1	q_3
q_1	q_1	q_2
* q_2	q_4	q_3
q_3	q_4	q_3
* q_4	q_1	q_2

Now, it is observed that for every NFA there exists some DFA that accepts the same language as accepted by NFA.

Now, let us ***"Formally prove that every NFA MN can be converted into DFA M_D such that $L(M_D) = L(M_N)$"***

Theorem: If there exists NFA $MN = (Q_N, \Sigma, \delta_N, q_0, F_N)$ which accepts the language $L(M_N)$, there exists an equivalent DFA $MD = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that $L(M_D) = L(M_N)$.

Proof: It is required to prove that

$$\delta_D (q_0, w) = \delta_N (q_0, w)$$

We know that if Q_N represent states of NFA then power set of Q_N which contains the set of subsets of set Q_N are the states of DFA denoted Q_D. The DFA interprets each set as a single state in DFA.

Basis: Consider a string $w = \varepsilon$ where $|w| = 0$

$$\delta_D(\{q_0\}, \varepsilon) = \{q_0\} \text{ by definition}$$

$$\delta_N (\{q_0\}, \varepsilon) = \{q_0\} \text{ by definition}$$

Hence $\delta_D(q_0,w) = \delta_N(q_0, w)$ is proved when $w = \varepsilon$

Induction hypothesis: Now, let us assume that

$$\delta_D(q_0, w) = \delta_N(q_0, w) \text{ for some } w \text{ where } |w| = n + 1$$

Induction proof: Let $w = xa$ where a is the last symbol of w and x is the remaining string of w. So,

$$|x| = n \text{ and } |xa| = n + 1$$

By definition δ of NFA we know that

$$\delta_N(q_0, w) = \delta_N(q_0, xa)$$

$$= \delta_N(\delta_N(q_0, x), a) \qquad ...(1)$$

Now, x is the string to be processed and after consuming the string x, let the states of the machine be $\{p_i, p_j, \ldots\ldots p_k\}$

i.e., $\delta_N(q_0, x) = \{p_i, p_j, \ldots\ldots p_k\}$

Substituting the above relation in equation (1) we have

$$\delta_N(q_0, w) = \delta_N(\{p_i, p_j, \ldots\ldots p_k\}, a)$$
$$= \delta_N(p_i, a) \cup \delta_N(p_j, a) \cup \ldots\ldots\ldots \delta_N(p_k, a) \quad \ldots(2)$$

By definition δ of DFA we know that

$$\delta_D(q_0, w) = \delta_D(q_0, xa)$$
$$= \delta_D(\delta_D(q_0, x), a) \quad \ldots(3)$$

Now, x is the string to be processed and after consuming the string x, let the states of the machine be $\{p_i, p_j, \ldots\ldots p_k\}$

i.e., $\delta_D(q_0, x) = \{p_i, p_j, \ldots\ldots p_k\}$

Substituting the above in equation (3) we have

$$\delta_D(q_0, w) = \delta_D(\{p_i, p_j, \ldots p_k\}, a) = \delta_N(p_i, a) \cup \delta_N(p_j, a) \cup \ldots \delta_N(p_k, a) \quad \ldots(4)$$

By comparing relations (2) and (4) we have

$$\delta_D(\{q_0\}, w) = \delta_N(\{q_0\}, w)$$

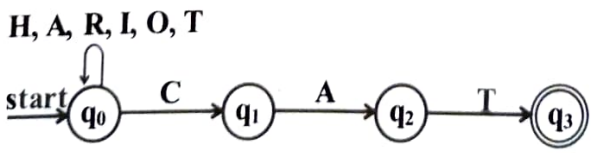
So, if $\delta_D(\{q_0\}, w)$ is in FD and $\delta_N(\{q_0\}, w)$ is in FN, then both enters into final state accepting the same language. Thus, $L(M_N) = L(M_D)$. Hence, the proof.

Example 2.59: Design a DFA to read strings made up of the letters "CHARIOT" and recognize those strings that contains the word "CAT" as a substring

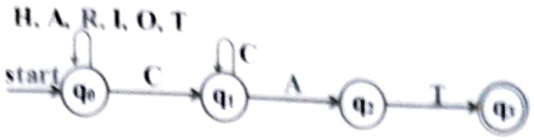
Solution: It is similar to the problem that we have discussed in example 2.11. The minimum string that can be accepted is "CAT" and the equivalent DFA can be written as shown in the diagram. Now, let us identify other transitions:

$\Sigma = \{C, H, A, R, I, O, T\}$

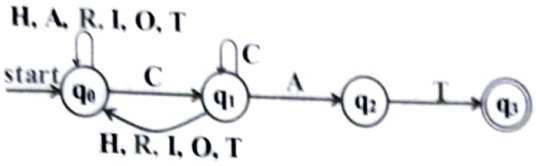
- **$\delta(q_0, Z)$:** In state q_0, **any number of Z's can be accepted** and the string ends with *CAT* where $Z = \{H, A, R, I, O, T\}$. The resulting string will have the substring *CAT* and the equivalent DFA is shown in the diagram:



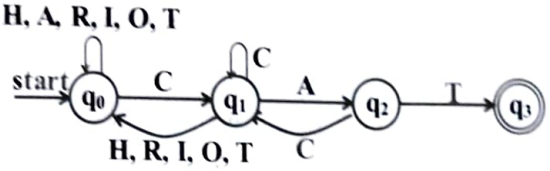
- **$\delta(q_1, C)$:** In state q_1, the machine can accept any number of C's and still the string has CAT as the substring. So, there should be a self-loop on C in state q_1. The equivalent DFA is shown:



- **$\delta(q_1, Z)$:** In state q_1, if the input symbol is not 'C' or not 'A' there will not be a substring CAT and hence to get the string CAT we have to go back to state q_0. Here, Z = {H, R, I, O, T}

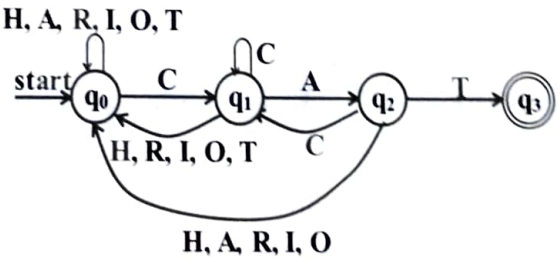


- **$\delta(q_2, C)$:** From state q_2 on C we have to go back to q_1 and the DFA can be written as shown below:

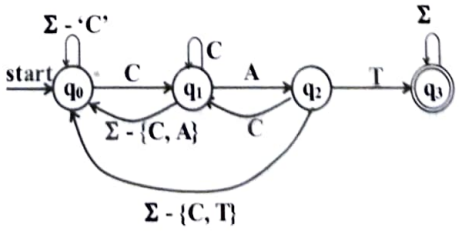


- **$\delta(q_2, Z)$:** In state q_2, if the input symbol is not 'C' or not 'T' there will not be a substring CAT and hence to get the string CAT we have to go back to state q0.

  Here, Z = {H, A, R, I, O}



- **$\delta(q_3, Z)$:** In state q_3, we have the substring "CAT" and so we can accept any number of symbols in Σ any number of times. Thus, there will be a self-loop on symbols in Σ from state q_3. The DFA can be written as:



So, formally, the DFA can be written as: **M = (Q, Σ, δ, q_0, F)** where

- Q = $\{q_0, q_1, q_2, q_3\}$
- Σ = {C, H, A, R, I, O, T}
- q_0 is the start state
- F = $\{q_3\}$
- δ is shown using the transition diagram:

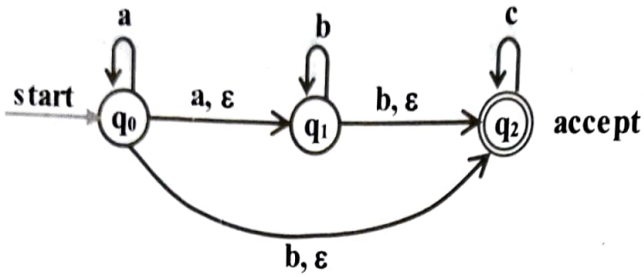
Exercises

1. What is DFA? Explain with an example.
2. When we say that a language is accepted by the DFA? Explain with example.
3. When a given language is not accepted by DFA? Explain with example.
4. How DFA's can be represented? Explain with example.
5. What is a transition diagram/graph?
6. Obtain a DFA to accept strings of a's and b's starting with the string ab.
7. Draw a DFA to accept string of 0's and 1's ending with the string 011.
8. Obtain a DFA to accept strings of a's and b's having a sub string aa.
9. Obtain a DFA to accept strings of a's and b's except those containing the substring aab.
10. Obtain DFAs to accept strings of a's and b's having exactly one a, atleast one a, not more than three a's.
11. Obtain a DFA to accept the language $L = \{awa \mid w \in (a+b)\}$.
12. Obtain a DFA to accept even number of a's, odd number of a's.
13. Obtain a DFA to accept strings of a's and b's having even number of a's and b's.
14. Obtain a DFA to accept odd number of a's and even number of b's.
15. Obtain a DFA to accept even number of a's and odd number of b's.
16. Obtain a DFA to accept strings of a's and b's having odd number of a's and b's.
17. Obtain a DFA to accept strings of 0's, 1's and 2's beginning with a '0' followed by odd number of 1's and ending with a '2'.
18. Obtain a DFA to accept binary odd numbers.
19. Obtain a DFA to accept strings of a's and b's starting with at least two a's and ending with at least two b's.
20. Obtain a DFA to accept strings of a's and b's with at most two consecutive b's.
21. Obtain a DFA to accept the language $L = \{ w : |w| \bmod 3 = 0 \text{ on } \Sigma = \{a, b\}$.
22. Obtain a DFA to accept the language $L = \{ w : |w| \bmod 5 \neq 0 \text{ on } \Sigma = \{a, b\}$.
23. What is a regular language? What are the applications of finite automaton?
24. What is an NFA? Explain with example.
25. What is the need for an NFA?
26. What is the difference between DFA and NFA?
27. Give a general procedure to convert an NFA to DFA.

3.1 ε-NFA (Finite Automata with Epsilon Transitions)

In this section, let us see the extended model of NFA called ε-NFA. An NFA with zero or more ε-transitions is called an ε-NFA. Now, let us see **"What is an ε-transition?"**

Definition: A transition with an empty input string is called an ε-transition (read as epsilon transition). That is, if there is a transition from one state to another state without any input (i.e no input implies empty string denoted by ε) is called **ε-transition**. For example, consider the finite automaton shown below:



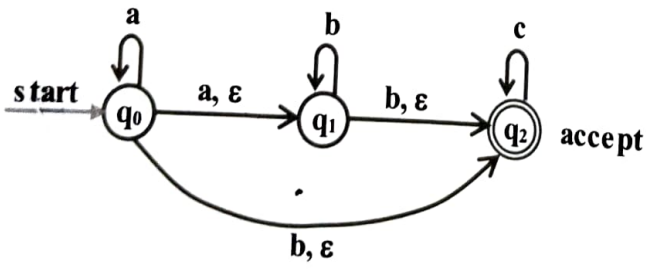
From above FA, observe the following points:

- In the above FA, we have more than two transitions from state q_0 with input symbol *a*. **So, it is an NFA.**
- There is a transition from state q_0 to q_1 with ε as the input. There is another transition from state q_1 to q_2 with ε as the input. So, the above FA is an NFA with ε-transitions and hence it is an ε-NFA. **Note:** If zero or more ε-transitions are present in a FA then it is an ε-NFA.

Note: If there is an ε-transition, then NFA makes transition without receiving any input symbol.

ε-NFA

Before worrying about the definition, let us consider the following pictorial representation of ε-NFA.



From the above figure, observe following components of ε-NFA:

- **States:** The ε-NFA has three states q_0, q_1 and q_2 and can be represented as

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

Note: The power set of Q is denoted by 2^Q which is set of subsets of set Q. This is denoted as shown below:

$$2^Q = \{ \{\}, \{q_0\}, \{q_1\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\} \}$$

- **Input alphabets:** Each edge is labeled with *a* or *b* and represent the input alphabets which can be denoted as:

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

- **Transitions:** Transition is nothing but change of state after consuming an input symbol. If there is a change of state from q_i to q_j on an input symbol a, then we write

$$\delta(q_i, a) = q_j$$

If there is a change of state from q_i to q_j and q_i to q_k on an input symbol a, then we write

$$\delta(q_i, a) = \{q_j, q_j\}$$

If there is a change of state from q_i to q_j on ε (no input is read), then we write:

$$\delta(q_i, \varepsilon) = \{q_j\}$$

The transitions for above ε-NFA is shown below:

Current State	Input	Next state	Representation
q_0	a	$\{q_1,q_2\}$	$\delta(q_0, a) = \{q_1,q_2\}$
q_0	b	$\{q_2\}$	$\delta(q_0, b) = q_2$
q_0	c	ϕ	$\delta(q_0, c) = \phi$
q_0	ε	$\{q_1\}$	$\delta(q_0, \varepsilon) = \{q_1\}$
q_1	a	ϕ	$\delta(q_1, a) = \phi$
q_1	b	$\{q_1,q_2\}$	$\delta(q_1, b) = \{q_1,q_2\}$
q_1	c	ϕ	$\delta(q_1, c) = \phi$
q_1	ε	$\{q_2\}$	$\delta(q_1, \varepsilon) = \{q_2\}$
q_2	a	ϕ	$\delta(q_2, a) = \phi$
q_2	b	ϕ	$\delta(q_2, b) = \phi$
q_2	c	$\{q_2\}$	$\delta(q_2, c) = \{q_2\}$
q_2	ε	ϕ	$\delta(q_2, \varepsilon) = \phi$

$\delta: Q \times (\Sigma \cup \varepsilon)$ to 2^Q [power set]

Now, let us see **"What is a transition function for ε-NFA?"** The transition function δ is defined as:

$$\delta: Q \times (\Sigma \cup \varepsilon) \text{ to } 2^Q$$

which is read as **"δ is a transition function which maps Q x (Σ U ε) to 2^Q"** i.e., the function accepts:

- state q as the first parameter
- input symbol in Σ or ε as the second parameter and
- returns a set of states the machine enters into.

- **Start state (q_0):** q_0 with the label **start** is treated as the start state
- **Final state (q_2):** q_2 with two circles is treated as the final or accept state.

Note: From this discussion, it is observed that the ε-NFA has five components:

(states, input alphabets, transitions, start state, final states)

$\downarrow$ Q $\downarrow$ Σ $\downarrow$ δ $\downarrow$ q_0 $\downarrow$ F

With this concept, now let us see **"What is an ε-NFA?"**

Definition: The **ε-NFA** is 5-tuple or quintuple indicating five components:

$$M = (Q, \Sigma, \delta, q_0, F)$$

where

- **Q** is non-empty, finite set of states.
- **Σ** is non-empty, finite set of input alphabets.
- **δ** : Q x (Σ U ε) to 2^Q i.e., δ is transition function which is a mapping from Q x (Σ U ε) to 2^Q. Based on the current state there can be a transition to other states with or without any input symbols.
- $q_0 \in Q$ is the start state.
- $F \subseteq Q$ is set of accepting or final states.

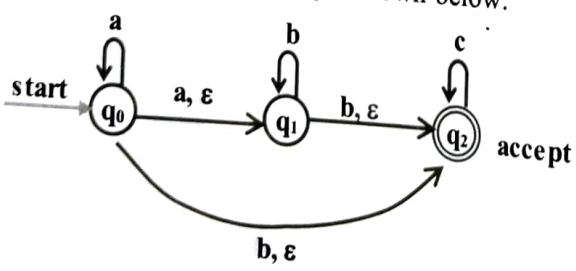
Note: In an ε-NFA there can be zero, one or more transitions with or without any input symbol.

Before proceeding further, let us see **"What is ε-CLOSURE?"**

Definition: The ε-CLOSURE of q denoted by ECLOSE(q) is the set of all states which are reachable from q on ε-transitions only. It is recursively defined using recursion as shown below:

- State q is in ECLOSE(q) i.e., ESLOSE(q) = q
- If ECLOSE(q) contains p and if there is a transition from state *p* to state *r* labeled ε, then state *r* is also in ECLOSE(q).

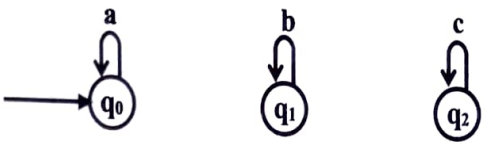
For example, the ε-CLOSURE(q) for each $q \in Q$ is shown below:



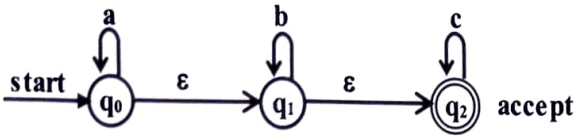
- The states q_0, q_1 and q_2 are reachable from q0 without giving any input. So, $ECLOSE(q_0) = \{q_0, q_1, q_2\}$
- The states q_1 and q_2 are reachable from q1 without giving any input. So, $ECLOSE(q_1) = \{q_1, q_2\}$
- The state q_2 is reachable from q_2 without giving any input. So, $ECLOSE(q_2) = \{q_2\}$

Example 3.1: Obtain an ε-NFA which accepts strings consisting of zero or more a's followed by zero or more b's followed by zero or more c's"

Solution: The ε-NFA that accepts strings of zero or more a's, zero or more b's and zero or more c's can be represented as shown below:



But, it is given that zero or more a's should be followed by zero or more b's followed by zero or more c's. So, there should be a transition from state q_0 to q_1 on ε and there should be a transition from state q_1 to q_2 on ε. Now, the corresponding ε-NFA can be written as shown below:



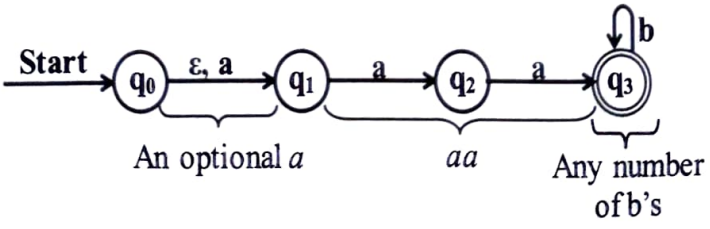
Thus, an ε-NFA is given by **M = (Q, Σ, δ, q_0, F)** where

- Q = {q_0, q_1, q_2}
- Σ = {a, b, c}
- q_0 is start state
- F = {q_2}
- δ is shown below using the transition table:

δ	a	b	c	ε
→ q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
*q_2	ϕ	ϕ	q_2	ϕ

Example 3.2: Obtain an NFA to accept L { w : w ∈ {a, b}* and w is made up of an optional *a* followed by *aa*.

Solution: The given language can be expressed as: **an optional *a* followed by *aa* followed by any number of b's.** The equivalent NFA can be written as shown below:



So, formally, an NFA can be defined as **M = (Q, Σ, δ, q_0, F)** where

- Q = {q_0, q_1, q_2, q_3}

- $\Sigma = \{a, b\}$
- q0 is the start state
- $F = \{q_3\}$
- δ is shown using the transition table

δ	a	b	ε
→ q_0	q_1	-	q_1
q_1	q_2	-	-
q_2	q_3	-	-
q_3	-	q_3	-

3.2 Conversion from ε-NFA to DFA (Algorithm to Convert ε-NFA to DFA)

We have seen in the previous section that an NFA can be converted into DFA. On similar lines, it is possible to convert an ε-NFA to DFA. Now, let us see **"What is the procedure (or algorithm) to obtain a DFA from an ε-NFA?"** The procedure (or algorithm) is shown below:

Procedure (Algorithm): Let $\mathbf{M_E = (Q_E, \Sigma, \delta_E, q_0, F_E)}$ be an ε-NFA where Q_E is set of finite states, Σ is set of input alphabets, δ_E is transition from Q x $\{\Sigma \cup \varepsilon\}$ to 2^Q, q_{0E} is the start state and F_E is the set of final states. The equivalent DFA

$$\mathbf{M_D = (Q_D, \Sigma, \delta_D, q_{0D}, F_D)}$$

can be obtained as shown below:

Step1: If q_0 is the start state of ε-NFA, then ECLOSE(q_0) is the start state of DFA

i.e., $\mathbf{q_{0D} = ECLOSE(q_0)}$

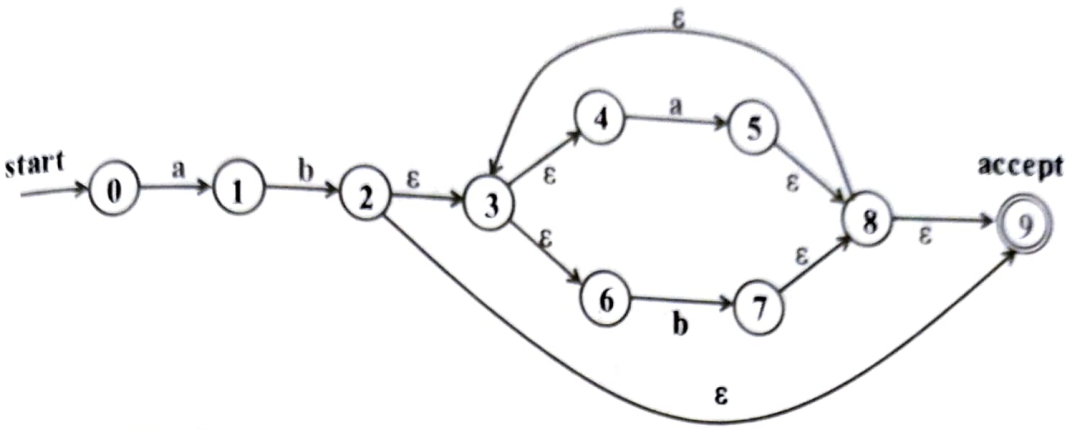
Step2: Compute the transitions for DFA. Let $\{q_i, q_j, \ldots..q_k\}$ is a state in DFA. Then, the transitions from this state on a i.e., $\delta_D(\{q_i, q_j, \ldots..q_k\}, a)$ is computed as shown below:

- Let $\delta_E(\{q_i, q_j, \ldots..q_k\}, a) = \{p_1, p_2, \ldots..p_m\}$
- Then take ECLOSE($\{p_1, p_2, \ldots..p_m\}$)

Thus, $\delta_D(\{q_i, q_j, \ldots..q_k\}, a)$ = ECLOSE($\{p_1, p_2, \ldots..p_m\}$)

Step 3: If $\{q_i, q_j, \ldots..q_k\}$ is a state in DFA and if this set contains at least one final state of ε-NFA, then $\{q_i, q_j, \ldots..q_k\}$ is a final state of DFA.

Example 3.3: Convert the following ε-NFA to its equivalent DFA.



Note: δ_E represent the **transition for ε-NFA**

Step 1: Identify the start state of DFA: Since 0 is the start state of ε-NFA, ECLOSE(0) is the start state of DFA i.e., ECLOSE (0) = **{0}** **(A)**

Consider the state A:

When input is *a*:

$\delta(A, a) = ECLOSE(\delta_E(A, a))$

$= ECLOSE(\delta_E(0, a))$

$= \{1\}$ (B)

When input is *b*:

$\delta(A, b) = ECLOSE(\delta_E(A, b))$

$= ECLOSE(\delta_E(0, b))$

$= \{\phi\}$

Consider the state B:

When input is *a*:

$\delta(B, a) = ECLOSE(\delta_E(B, a))$

$= ECLOSE(\delta_E(1, a))$

When input is *b*:

$\delta(B, b) = ECLOSE(\delta_E(B, b))$

$= ECLOSE(\delta_E(1, b))$

$= ECLOSE(\{2\})$

$= \{2,3,4,6,9\}$ (C)

Consider the state C:

When input is *a*:

$\delta(C, a) = ECLOSE(\delta_E(C, a))$

$= ECLOSE(\delta_E(\{2,3,4,6,9\}, a))$

$= ECLOSE\ \{5\}$

= {5, 8, 9, 3, 4, 6}

= {3, 4, 5, 6, 8, 9} (ascending order) (D)

When input is *b*:

δ(C, b) = ECLOSE (δ_E(C, b))

= ECLOSE (δ_E({2, 3, 4, 6, 9}, b)

= ECLOSE ({7})

= {7, 8, 9, 3, 4, 6}

={3, 4, 6, 7, 8, 9}(ascending order) (E)

Consider the state D:

When input is a:

δ(D, a) = ECLOSE (δ(D, a))

= ECLOSE (δ_E({3,4,5,6,8,9}, a))

= ECLOSE ({5})

= {5, 8, 9, 3, 4, 6}

= {3, 4, 5, 6, 8, 9} (ascending order) (D)

When input is *b*:

δ(D, b) = ECLOSE (δ(D, b))

= ECLOSE (δ_E({3,4,5,6,8,9}, b))

= ECLOSE ({7})

= {7, 8, 9, 3, 4, 6}

= {3, 4, 6, 7, 8, 9} (ascending order) (E)

Consider the state E:

When input is *a*:

δ(E, a) = ECLOSE (δ(E, a))

= ECLOSE (δ_E({3,4,6,7,8,9}, a))

= ECLOSE ({5})

= {5, 8, 9, 3, 4, 6}

= {3, 4, 5, 6, 8, 9} (ascending order) (D)

When input is *b*:

δ(E, b) = ECLOSE (δ(E, b))

= ECLOSE (δ_E({3,4,6,7,8,9}, b))

= ECLOSE ({7})

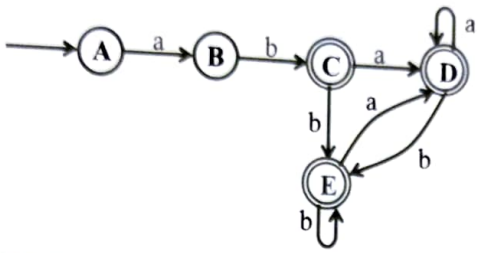
= {7, 8, 9, 3, 4, 6}

= {3, 4, 6, 7, 8, 9} (ascending order) (E)

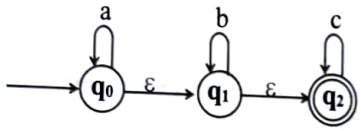
Since there are no new states, we can draw the transition table for the DFA as shown below:

δ	a	b
→ A	B	ϕ
B	ϕ	C
*C	D	E
*D	D	E
*E	D	E

Note: Since state 9 is final state of ε-NFA and state 9 is present states C, D and E we say that C, D and E are final states of DFA. The final transition diagram of DFA is shown below:



Example 3.4: Convert the following NFA to its equivalent DFA.



Note: δ_E represent the transition for ε-NFA

Note: Identify the start state of DFA: Since q_0 is the start state of ε-NFA, ECLOSE(q_0) is the start state of DFA i.e., **ECLOSE (q_0) = {q_0, q_1, q_2}..... (A)**

Consider the state {q_0, q_1, q_2}:

On input *a*:

$$\delta(\{q_0, q_1, q_2\}, a) = \text{ECLOSE}(\{q_0\})$$
$$= \{q_0, q_1, q_2\} \quad \text{(A)}$$

On input *b*:

$$\delta(\{q_0, q_1, q_2\}, b) = \text{ECLOSE}(\{q_1\}).$$
$$= \{q_1, q_2\} \quad \text{(B)}$$

On input *c*:

$$\delta(\{q_0, q_1, q_2\}, c) = \text{ECLOSE}(\{q_2\})$$
$$= q_2 \quad \text{(C)}$$

Consider the state {q_1, q_2}:

On input *a*:

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

On input *b*:

$$\delta(\{q_1, q_2\}, b) = \text{ECLOSE}(\{q_1\}) = \{q_1, q_2\} \quad (B)$$

On input *c*:

$$\delta(\{q_1, q_2\}, c) = \text{ECLOSE}(\{q_2\}) = q_2 \quad (C)$$

Consider the state $\{q_2\}$:

On input *a*:

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

On input *b*:

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

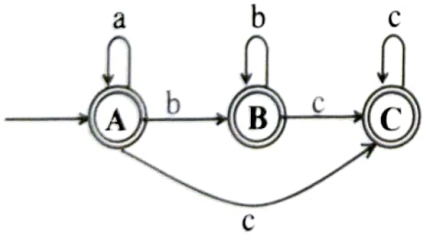
On input *c*:

$$\delta(q_2, c) = \text{ECLOSE}(q_2) = q_2 \quad (C)$$

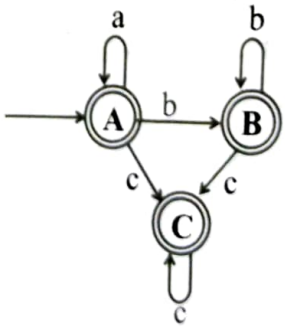
The transition table along with transition diagram is shown below:

δ	a	b	c
→*A	A	B	C
*B	ϕ	B	C
*C	ϕ	ϕ	C

The DFA can be written as shown below:



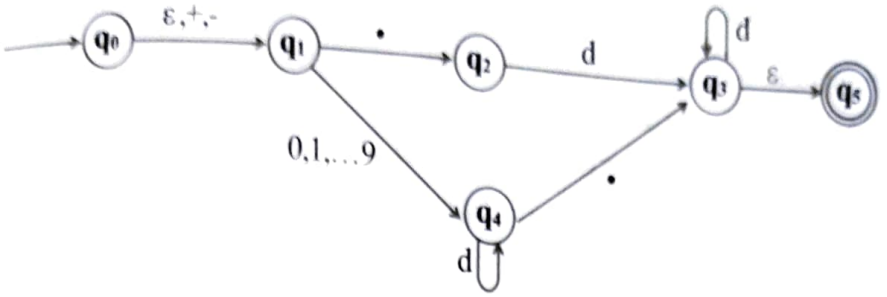
OR



Example 3.5: Obtain an NFA with ε-transitions (ε-NFA) to accept decimal numbers and obtain $\delta(q_0, 4.7)$

Note: A decimal number has: An optional + or – sign followed by '.' followed by one or more digits *d* **OR** an optional + or – sign followed by a string of digits followed by a '.' and in turn followed by string of digits.

The ε-NFA is shown in figure below:



Here, the machine **M = (Q, Σ, δ, q_0, F)** where

- Q = {q_0, q_1, q_2, q_3, q_4, q_5}
- Σ = { +, –, ., d} where d = {0, 1, 2,.....9}
- q_0 is the start state
- F = { q_5 } is the final state
- δ is shown below using the transition table

δ	ε	+	–	.	d
→ q_0	q_1	q_1	q_1	ϕ	ϕ
q_1	ϕ	ϕ	ϕ	q_2	q_4
q_2	ϕ	ϕ	ϕ	ϕ	q_3
q_3	q_5	ϕ	ϕ	ϕ	q_3
q_4	ϕ	ϕ	ϕ	q_3	q_4
*q_5	ϕ	ϕ	ϕ	ϕ	ϕ

To obtain $\delta(q_0, 4.7)$: Since q_0 is the start state of ε-NFA, ECLOSE(q_0) is the start state of DFA i.e., ECLOSE (q_0) = {q_0, q_1}

$\delta(q_0, 4) = ECLOSE(\delta_E(\{q_0, q_1\}, 4))$

$= ECLOSE (\delta_E(q_0, 4) \cup \delta_E(q_1, 4))$

$= ECLOSE (\phi \cup q4)$

$= \{q_4\}$

$\delta(\{q_4\}, .) = ECLOSE(\delta_E(\{q_4\}, .))$

$= ECLOSE (q_3)$

$= \{q_3, q_5\}$

$\delta(\{q_3, q_5\}, 7) = ECLOSE (\delta_E(q_3, 7) \cup \delta_E(q_5, 7))$

$= ECLOSE (q_3 \cup \phi)$

$= ECLOSE (q_3)$

$= \{q_3, q_5\}$

Thus, $\delta(q_0, 4.7) = \{q_3, q_5\}$ Since q_5 is the final state, the string 4.7 is accepted.

Example 3.6: Obtain the equivalent DFA for the above ε-NFA

Note: Identify the start state of DFA: Since q_0 is the start state of ε-NFA, ECLOSE(q_0) is the start state of DFA i.e., **ECLOSE (q_0) = {q_0, q_1}**..... (A)

Consider the state [A]:

When input is ± :

$\delta(A, \pm)$ $= ECLOSE(\delta_E(A, \pm))$

$= ECLOSE(\delta_E(\{q_0,q_1\}, \pm))$

$= ECLOSE(q_1) = \{q_1\}$ (B)

When input is .

$\delta(A, .)$ $= ECLOSE(\delta_E(A, .))$

$= ECLOSE(\delta_E(\{q_0,q_1\}, .))$

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

When input is d

$\delta(A, d)$ $= ECLOSE(\delta_E(A, d))$

$= ECLOSE(\delta_E(\{q_0,q_1\}, d))$

$= ECLOSE(q_4) = \{q_4\}$ (D)

Consider the state [B]:

When input is ± :

$\delta(B, \pm)$ $= ECLOSE(\delta_E(B, \pm))$

$= ECLOSE(\delta_E(q_1, \pm))$

$= ECLOSE(\phi) = \phi$

When input is .

$\delta(B, .)$ $= ECLOSE(\delta_E(B, .))$

$= ECLOSE(\delta_E(q_1, .))$

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

When input is d

$\delta(B, d)$ $= ECLOSE(\delta_E(B, d))$

$= ECLOSE(\delta_E(q_1, d))$

$= ECLOSE(q_4) = \{q_4\}$ (D)

Consider the state [C]:

When input is ± :

$\delta(C, \pm)$ $= ECLOSE(\delta_E(C, \pm))$

$= ECLOSE(\delta_E(q_2, \pm))$

$= ECLOSE(\phi) = \phi$

When input is .

$\delta_E(C, .)$ $= ECLOSE(\delta_E(C, .))$

$= ECLOSE(\delta_E(q_2, .))$

$= \text{ECLOSE}(\phi) = \phi$

When input is d

$\delta(C, d) = \text{ECLOSE}(\delta_E(C, d))$

$= \text{ECLOSE}(\delta_E(q_2, d))$

$= \text{ECLOSE}(q_3)$

$= \{q_3, q_5\}$

Consider the state [D]: (E)

When input is ± :

$\delta(D, \pm) = \text{ECLOSE}(\delta_E(D, \pm))$

$= \text{ECLOSE}(\delta_E(q_4, \pm))$

$= \text{ECLOSE}(\phi) = \phi$

When input is .

$\delta(D, .) = \text{ECLOSE}(\delta_E(D, .))$

$= \text{ECLOSE}(\delta_E(q_4, .))$

$= \text{ECLOSE}(q_3)$

$= \{q_3, q_5\}$

When input is d (E)

$\delta(D, d) = \text{ECLOSE}(\delta_E(D, d))$

$= \text{ECLOSE}(\delta_E(q_4, d))$

$= \text{ECLOSE}(q_4)$

$= \{q_4\}$ (D)

Consider the state [E]:

When input is ±:

$\delta_E(E, \pm) = \text{ECLOSE}(\delta_E(E, \pm))$

$= \text{ECLOSE}(\delta_E(\{q_3, q_5\}, \pm))$

$= \text{ECLOSE}(\phi) = \phi$

When input is .

$\delta_E(E, .) = \text{ECLOSE}(\delta_E(E, .))$

$= \text{ECLOSE}(\delta_E(\{q_3, q_5\}, .))$

$= \text{ECLOSE}(\phi) = \phi$

When input is d

$\delta(E, d) = \text{ECLOSE}(\delta_E(E, d))$

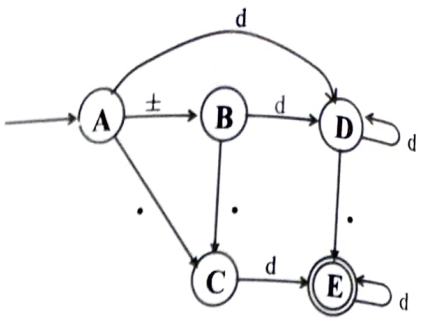
$= \text{ECLOSE}(\delta_E(\{q_3, q_5\}, d))$

$= \text{ECLOSE}(q_3)$

$= \{q_3, q_5\}$ (E)

The transition table along with transition diagram is shown below:

δ	+	−	.	0 to 9
→ A	B	B	C	D
B	ϕ	ϕ	C	D
C	ϕ	ϕ	ϕ	E
D	ϕ	ϕ	E	D
*E	ϕ	ϕ	ϕ	E



Note: Since E = $\{q_3, q_5\}$ has a final state of NFA, E is the final state in DFA.

Example 3.7: Consider the following ε-NFA

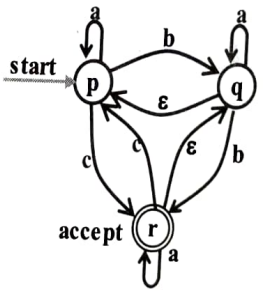
δ	ε	a	b	c
p	ϕ	{p}	{q}	{r}
q	{p}	{q}	{r}	ϕ
*r	{q}	{r}	ϕ	{p}

i) Compute ε-CLOSURE of each state

ii) Give all the strings of length three or less accepted by the automata

iii) Convert the automata to DFA **(08 marks VTU July 2006)**

Solution: The transition diagram for the given ε-NFA can be written as shown below:



1. **Solution:** The ε-CLOSURE of each state can be computed as shown below:

   ECLOSE(p) = {p}

   ECLOSE(q) = {p, q}

   ECLOSE(r) = {p,q,r}.

2. **Solution:** All the strings of length three or less accepted by the automata can be obtained as shown below:

   - The strings accepted by ε-NFA whose length is one: *c*
   - The strings accepted by ε-NFA whose length is two: ac, bb, bc, cb, cc, ca
   - The strings accepted by ε-NFA whose length is three can be obtained by prefixing *a/b* or suffixing *a/b* to the above set as shown:

   aac, abb, abc. acb, acc, aca (by prefixing *a* to strings of length 2)

   aca, bba, bca, cba, cca, caa (by suffixing *a* to strings of length 2)

aac, bbb, bbc, bcb, bcc, bca (by prefixing *b* to strings of length 2)

acb, bbb, bcb, cbb, ccb, cab (by suffixing *b* to strings of length 2)

Solution: The DFA can be obtained using ε-NFA as shown below:

Note: δ_1 represent the transition for ε-NFA

Note: Identify the start state of DFA. Since p is the start state of ε-NFA, ECLOSE(p) is the start state of DFA i.e., **ECLOSE (p) = {p}**..... (A)

Consider the state {p} i.e., A:

On input *a*:

$$\delta(\{p\}, a) = \text{ECLOSE}(\{p\})$$
$$= \{p\} \quad \text{(A)}$$

On input *b*:

$$\delta(\{p\}, b) = \text{ECLOSE}(\{q\})$$
$$= \{p, q\} \quad \text{(B)}$$

On input *c*:

$$\delta(\{p\}, c) = \text{ECLOSE}(\{r\})$$
$$= \{p, q, r\} \quad \text{(C)}$$

Consider the state {p,q} i.e., B:

On input *a*:

$$\delta(\{p,q\}, a) = \text{ECLOSE}(\delta(p,a) \cup \delta(q, a))$$
$$= \text{ECLOSE}(p, q)$$
$$= \text{ECLOSE}(p,q)$$
$$= \{p,q\} \quad \text{(B)}$$

On input *b*:

$$\delta(\{p,q\}, b) = \text{ECLOSE}(\delta(p,b) \cup \delta(q, b))$$
$$= \text{ECLOSE}(q, r)$$
$$= \text{ECLOSE}(p) \cup \text{ECLOSE}(r)$$
$$= \{p,q,r\} \quad \text{(C)}$$

On input *c*:

$$\delta(\{p,q\}, c) = \text{ECLOSE}(\delta(p,c) \cup \delta(q, c))$$
$$= \text{ECLOSE}(r, \phi)$$
$$= \text{ECLOSE}(r)$$
$$= \{p,q,r\} \text{ (C)}$$

Consider the state {p,q,r} i.e., C:

On input *a*:

$$\delta(\{p,q,r\}, a) = \text{ECLOSE}(\delta(p,a) \cup \delta(q, a) \cup \delta(r,a))$$
$$= \text{ECLOSE}(p, q, r)$$
$$= \text{ECLOSE}(p,q,r)$$
$$= \{p,q,r\} \quad (C)$$

On input *b*:

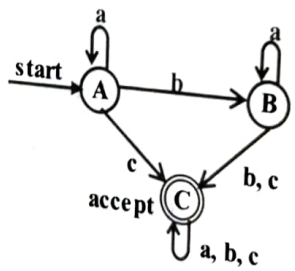
$$\delta(\{p,q,r\}, b) = \text{ECLOSE}(\delta(p,b) \cup \delta(q, b) \cup \delta(r,b))$$
$$= \text{ECLOSE}(q, r, \phi)$$
$$= \text{ECLOSE}(p,q,r)$$
$$= \{p, q, r\} \quad (C)$$

On input *c*:

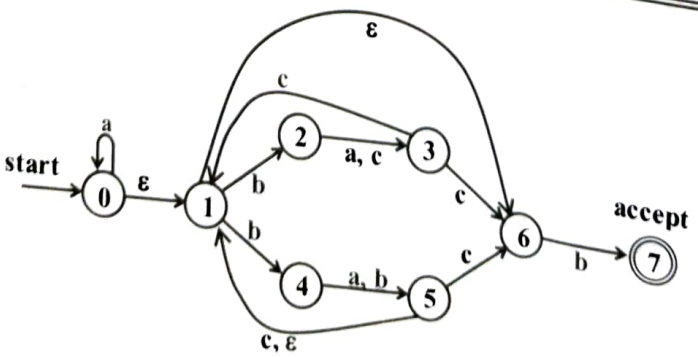
$$\delta(\{p,q,r\}, c) = \text{ECLOSE}(\delta(p,c) \cup \delta(q, c) \cup \delta(r,c))$$
$$= \text{ECLOSE}(r, \phi, \phi)$$
$$= \text{ECLOSE}(r)$$
$$= \{p, q, r\} \quad (C)$$

The transition table along with transition diagram is shown below:

δ	a	b	c
→ A	A	B	C
B	B	C	C
*C	C	C	C



Example 3.8: Obtain the equivalent DFA for the following ε-NFA



Note: δ_E represent the transition for ε-NFA and δ represent the transitions for DFA

Step 1: Identify the start state of DFA. Since 0 is the start state of ε-NFA, ECLOSE(0) is the start state of DFA i.e., **ECLOSE (0) = {0, 1, 6}** **(A)**

Consider the state A:

$\delta(A, a) = \phi$

$\delta(A, b) = ECLOSE(0, 2, 4, 7)$

$= \{0, 1, 6, 2, 4, 7\}$

$= \{0, 1, 2, 4, 6, 7\}$ **(B)**

Consider the state B:

$\delta(B, a) = ECLOSE(3, 5)$

$= \{1, 3, 5, 6\}$ **(C)**

$\delta(B, b) = ECLOSE(0, 2, 4, 5, 7)$

$= \{0, 1, 2, 4, 5, 6, 7\}$ **(D)**

$\delta(B, c) = ECLOSE(3)$

$= \{3\}$ **(E)**

Consider the state C:

$\delta(C, a) = ECLOSE(\phi)$

$= \phi$

$\delta(C, b) = ECLOSE(2, 4, 7)$

$= \{2, 4, 7\}$ **(F)**

$\delta(C, c) = ECLOSE(1, 6)$

$= \{1, 6\}$ **(G)**

Consider the state D:

$\delta(D, a) = ECLOSE(3, 5)$ **(C)**

$= \phi$

$\delta(D, b) = ECLOSE(0, 2, 4, 5, 7)$

$= \{0, 1, 2, 4, 5, 6, 7\}$ **(D)**

$\delta(D, c) = ECLOSE(3, 1, 6)$

$= \{1, 3, 6\}$ **(H)**

Consider the state E:

$\delta(E, a) = ECLOSE(\phi) = \phi$

$\delta(E, b) = ECLOSE(\phi) = \phi$

$\delta(E, c) = ECLOSE(1, 6)$ **(G)**

Consider the state F:

$\delta(F, a) = ECLOSE(3, 5)$ **(C)**

$\delta(F, b) = \text{ECLOSE}(5) = \{1, 5, 6\}$ (I)

$\delta(F, c) = \text{ECLOSE}(3)$ (E)

Consider the state G:

$\delta(G, a) = \text{ECLOSE}(\phi) = \phi$

$\delta(G, b) = \text{ECLOSE}(2, 4, 7) = \{2, 4, 7\}$ (F)

$\delta(G, c) = \text{ECLOSE}(\phi) = \phi$

Consider the state H:

$\delta(H, a) = \text{ECLOSE}(\delta) = \delta$

$\delta(H, b) = \text{ECLOSE}(2, 4, 7) = \{2, 4, 7\}$ (F)

$\delta(H, c) = \text{ECLOSE}(1, 6)$ (G)

Consider the state I:

$\delta(I, a) = \text{ECLOSE}(\phi) = \phi$

$\delta(I, b) = \text{ECLOSE}(2, 4, 7) = \{2, 4, 7\}$ (F)

$\delta(I, c) = \text{ECLOSE}(1, 6)$ (G)

Identify the final state: The state 7 is the final state of ε-NFA and it is in states B, D, F and so **the states {B, D, F} are the final states.**

So, the final DFA can be written as $\mathbf{M = (Q, \Sigma, \delta, q_0, F)}$ where

- $Q = \{A, B, C, D, E, F, G, H, I\}$
- $\Sigma = \{a, b, c\}$
- q_0 is the start state
- $F = \{B, D, F\}$
- δ is shown using the transition table:

δ	a	b	c
→ A	-	B	ϕ
*B	C	D	D
C	ϕ	F	G
*D	C	D	H
E	ϕ	ϕ	G
*F	C	I	E
G	ϕ	F	ϕ
H	ϕ	F	G
I	ϕ	F	G

3.3 From FSMs to Operational Systems (Applications of Finite Automata)

Now, let us see **"Why to study finite automata? or What are applications of finite automata?" [VTU-JULY 2007]** Some of the applications where automata play an important role are shown below:

Design of digital circuits: The Finite Automata (also called Finite State Machine) can be translated into a circuit design and can be directly implemented in hardware.

Compiler construction: Used in the design of *lexical analyzer* (the first phase of compiler design) which breaks the input text into various units such as identifiers, keywords, punctuation etc.

String matching: In designing a software for identifying the words, phrases and other patterns in large bodies of text (such as collection of web pages).

String processing: To write software for processing the natural language (Ex: Speech processing). Large natural vocabularies can be described which includes the applications such as spelling checkers and advisers, multi-language dictionaries, indenting the documents etc.

Software design: In building the software to verify the systems having finite number of states (for example, communication protocols in computer networks).

Other applications: The FA are used in variety of applications in Artificial intelligence and knowledge engineering, in game theory and games, computer graphics, linguistics etc.

3.4 Difference Between DFA, NFA and ε-NFA

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

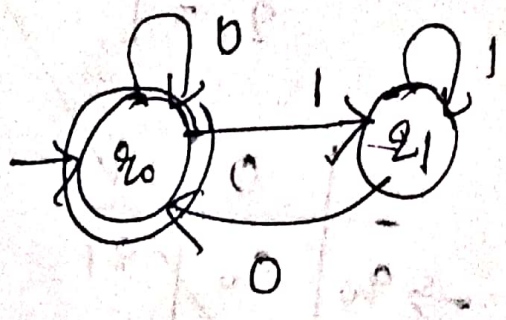
DFA	NFA	ε -NFA
The DFA is a 5-tuple **$M = (Q, \Sigma, \delta, q_0, F)$** where ♦ Q is set of finite states ♦ Σ is set of input alphabets ♦ **$\delta : Q \times \Sigma$ to Q** ♦ q_0 is the start state ♦ $F \subseteq Q$ is set of final states	Q is set of finite states **$M = (Q, \Sigma, \delta, q_0, F)$**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

① Binary numbers divisible by 2

remainders = 0, 1

$q_0 = 0$, $q_1 = 1$

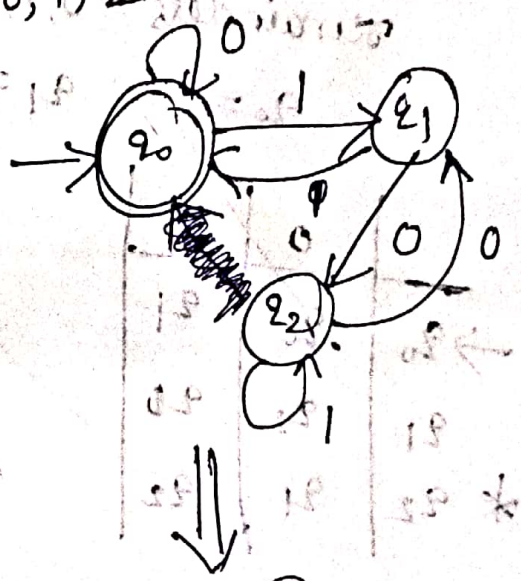
	0	1
* → q_0	q_0	q_1
q_1	q_0	q_1

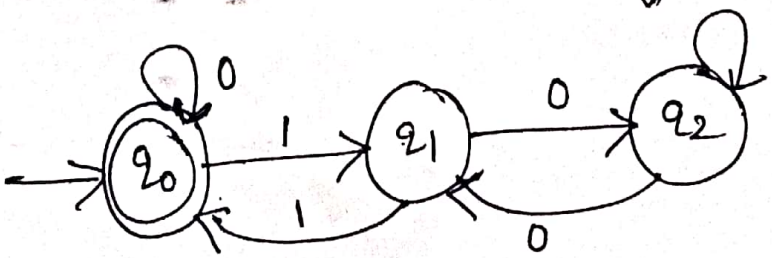


② Binary numbers divisible by 3

remainders = 0, 1, 2

	0	1
* → q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2





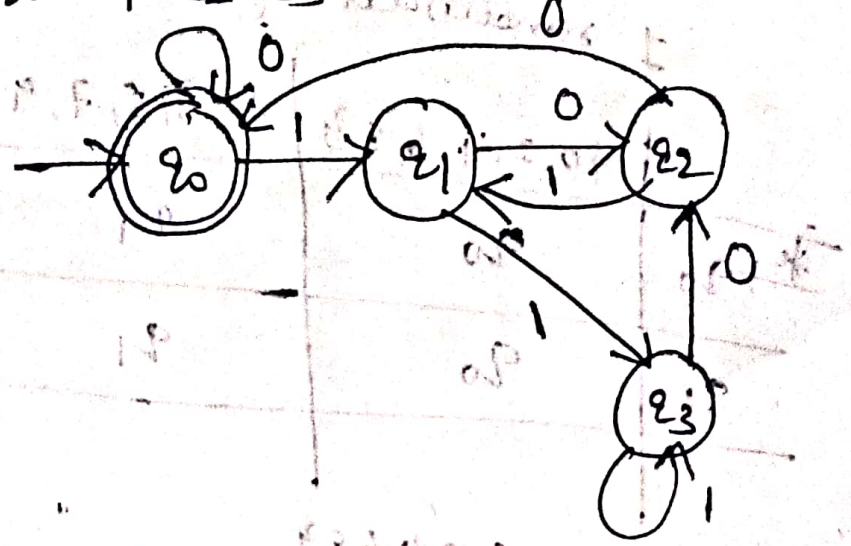
(8) Binary numbers divisible by 4

remainders = 0, 1, 2, 3 . . . . .

↓ ↓ ↓ ↓

q_0 q_1 q_2 q_3

	0	1
*→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3



(4) Decimal numbers, divisible by 2.

$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

remainder = 0, 1

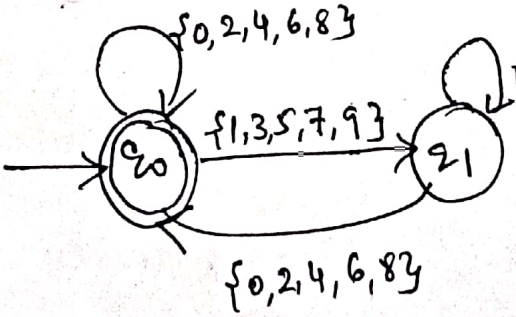
q_0 q_1

	0	1	2	3	4	5	6	7	8	9
→ * q_0	q_0	q_1	q_0	q_1	q_0	q_1	q_0	q_1	q_0	q_1
q_1	q_0	q_1	q_0	q_1	q_0	q_1	q_0	q_1	q_0	q_1

0 remainder = {0, 2, 4, 6, 8}

1 remainder = {1, 3, 5, 7, 9}

	{0,2,4,6,8}	{1,3,5,7,9}
→ * q_0	q_0	q_1
q_1	q_0	q_1



$Q = \{q_0, q_1\}$

$\Sigma = \{\{0,2,4,6,8\}, \{1,3,5,7,9\}\}$

$q_0 = \{q_0\}$

$F = \{q_0\}$

(5) Decimal numbers divisible by 3.

$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

remainder = 0, 1, 2

q_0 q_1 q_2

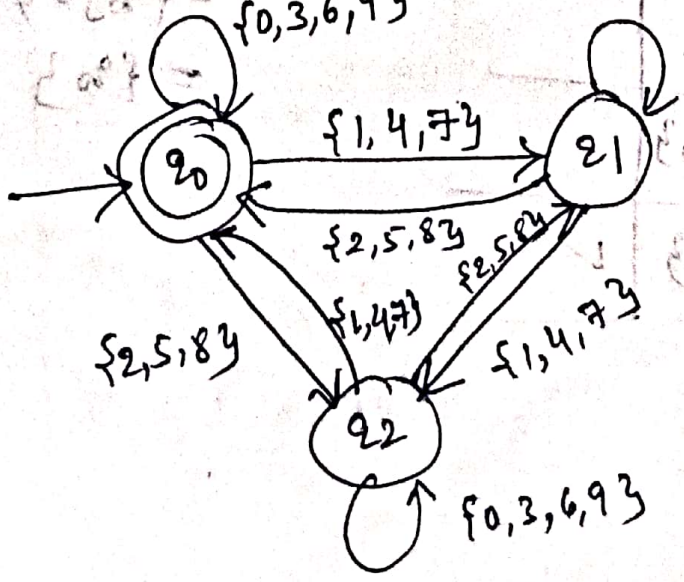
	0	1	2	3	4	5	6	7	8	9
→ q_0 *	q_0	q_1	q_2	q_0	q_1	q_2	q_0	q_1	q_2	q_0
q_1	q_1	q_2	q_0	q_1	q_2	q_0	q_1	q_2	q_0	q_1
q_2	q_2	q_0	q_1	q_2	q_0	q_1	q_2	q_0	q_1	q_2

0 remainder → {0, 3, 6, 9}

1 remainder → {1, 4, 7}

2 remainder → {2, 5, 8}

	{0, 3, 6, 9}	{1, 4, 7}	{2, 5, 8}
→ q_0 *	q_0	q_1	q_2
q_1	q_1	q_2	q_0
q_2	q_2	q_0	q_1



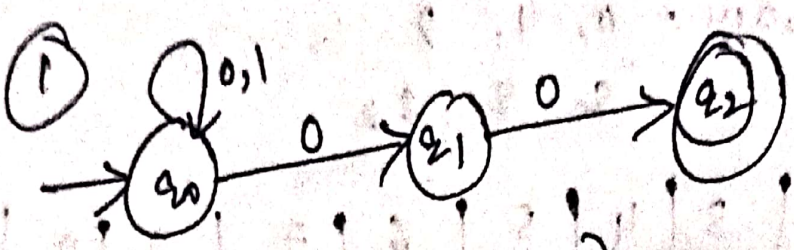
$Q = \{q_0, q_1, q_2\}$

$q_0 = \{q_0\}$

$F = \{q_0\}$

NFA to DFA

Subset construction Method



NFA table (transition)

	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	q_2	ϕ
* q_2	ϕ	ϕ

NFA - n = 3 states

DFA - $2^n = 2^3 = 8$ states

DFA transition table

	0	1	
ϕ	ϕ	ϕ	
→ q_0	$\{q_0, q_1\}$	q_0	✓
q_1	q_2	ϕ	
q_2	ϕ	ϕ	
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$	✓
$\{q_1, q_2\}$	q_2	ϕ	
$\{q_2, q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$	
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$	✓

$\delta(\{q_0, q_1\}, a) = \delta(q_0, a) \cup \delta(q_1, 0)$

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

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

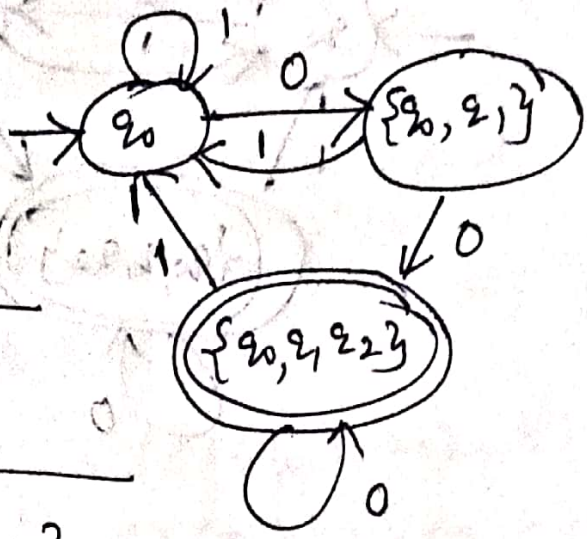
$\delta(\{q_0, q_1\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1)$

$= \{q_0\} \cup \phi$

$= \{q_0\}$

Final DFA transition table

δ:	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0 q_1, q_2\}$	q_0
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	q_0

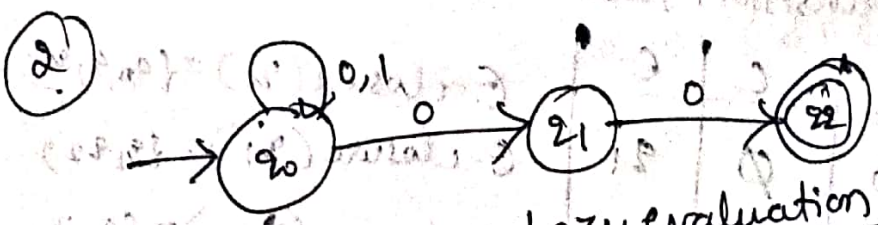


$Q = \{ q_0, \{q_0, q_1\}, \{q_0, q_1, q_2\} \}$

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

$q_0 = \{q_0\}$

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



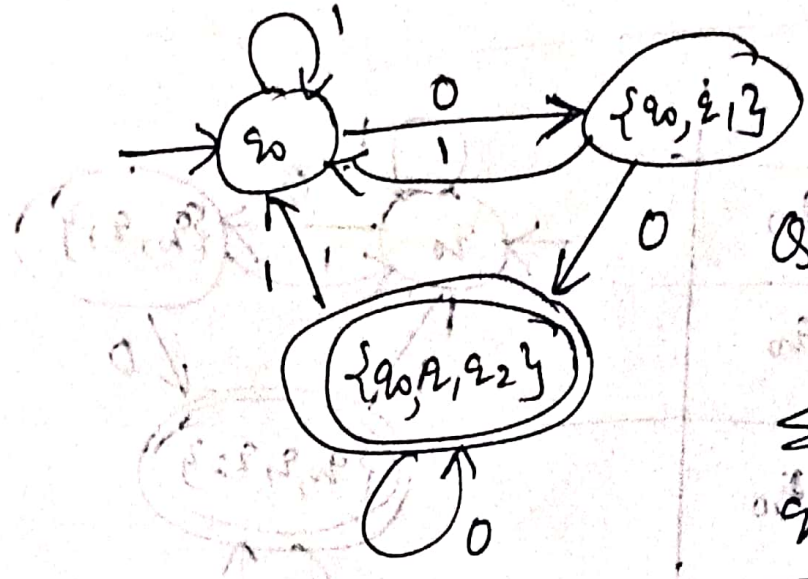
Using Lazy evaluation method
Convert NFA to DFA.

NFA Transition table

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
q_1	q_2	ϕ
$* q_2$	ϕ	ϕ

DFA transition table

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	q_0
$*\{q_0, q_1, q_2\}$	$\{q_0 q_1, q_2\}$	q_0



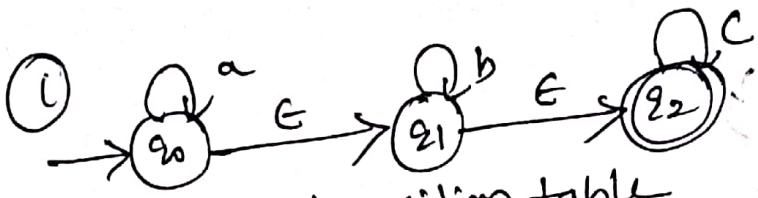
$Q = \{q_0, \{q_0,q_1\}, \{q_0,q_1,q_2\}\}$

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

$q_0 = \{q_0\}$

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

ϵ-NFA to DFA



ϵ-NFA transition table

	a	b	c	ϵ
→ q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
* q_2	ϕ	ϕ	q_2	ϕ

ϵ-closure(q_0) = $\{q_0,q_1,q_2\}$

ϵ-closure(q_1) = $\{q_1,q_2\}$

ϵ-closure(q_2) = $\{q_2\}$

DFA table

	a	b	c
* $\{q_0,q_1,q_2\}$	$\{q_0,q_1,q_2\}$	$\{q_1,q_2\}$	$\{q_2\}$
* $\{q_1,q_2\}$	ϕ	$\{q_1,q_2\}$	$\{q_2\}$
* $\{q_2\}$	ϕ	ϕ	$\{q_2\}$

$\delta(\{q_0,q_1,q_2\}, a) = \epsilon\text{-closure}(\delta(\{q_0,q_1,q_2\}, a))$

$= \epsilon\text{-closure}(\delta(q_0,a) \cup \delta(q_1,a) \cup \delta(q_2,a))$

$= \epsilon\text{-closure}(q_0 \cup \emptyset \cup \emptyset)$

$= \epsilon\text{-closure}(q_0)$

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

$\delta(\{q_0,q_1,q_2\}, b) = \epsilon\text{-closure}(\delta(\{q_0,q_1,q_2\}, b))$

$= \epsilon\text{-closure}(\delta(q_0,b) \cup \delta(q_1,b) \cup \delta(q_2,b))$

$= \epsilon\text{-closure}(\emptyset \cup q_1 \cup \emptyset)$

$= \epsilon\text{-closure}(q_1)$

$= \{q_1,q_2\}$

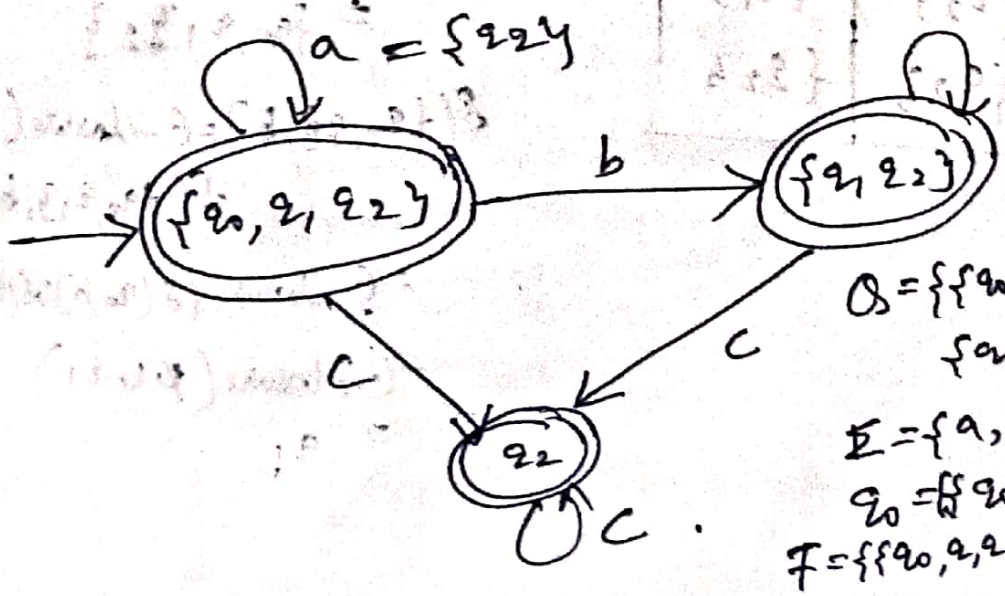
$\delta(\{q_0,q_1,q_2\}, c) = \epsilon\text{-closure}(\delta(\{q_0,q_1,q_2\}, c))$

$= \epsilon\text{-closure}(\delta(q_0,c) \cup \delta(q_1,c) \cup \delta(q_2,c))$

$= \epsilon\text{-closure}(\emptyset \cup \emptyset \cup q_2)$

$= \epsilon\text{-closure}(q_2)$

$= \{q_2\}$



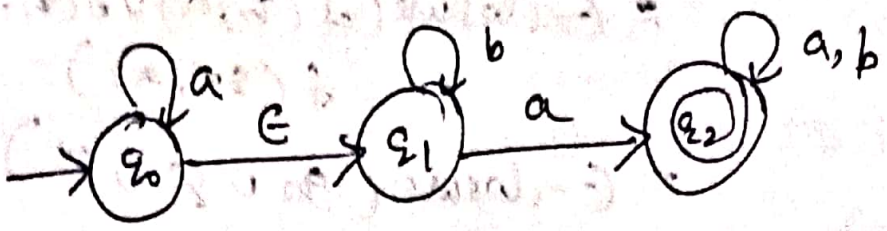
$Q = \{\{q_0,q_1,q_2\}, \{q_1,q_2\}, q_2\}$

$\Sigma = \{a,b,c\}$

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

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

(2) Convert from ϵ-NFA to DFA



ϵ-NFA transition table

	a	b	ϵ
→ q_0	q_0	∅	q_1
q_1	q_2	q_1	∅
* q_2	q_2	q_2	∅

ϵ-closure(q_0) = {q_0, q_1}

ϵ-closure(q_1) = {q_1}

ϵ-closure(q_2) = {q_2}

	a	b
→ {q_0, q_1}	{q_0, q_1, q_2}	q_1
* {q_0, q_1, q_2}	{q_0, q_1, q_2}	{q_1, q_2}
* {q_1, q_2}	{q_2}	{q_1, q_2}
{q_1}	{q_2}	{q_1}
* {q_2}	{q_2}	{q_2}

$\delta(\{q_0, q_1\}, a) = \epsilon\text{-closure}(\delta(\{q_0, q_1\}, a))$

$= \epsilon\text{-closure}(\delta(q_0, a) \cup \delta(q_1, a))$

$= \epsilon\text{-closure}(q_0 \cup q_2)$

$= \epsilon\text{-closure}(q_0, q_2)$

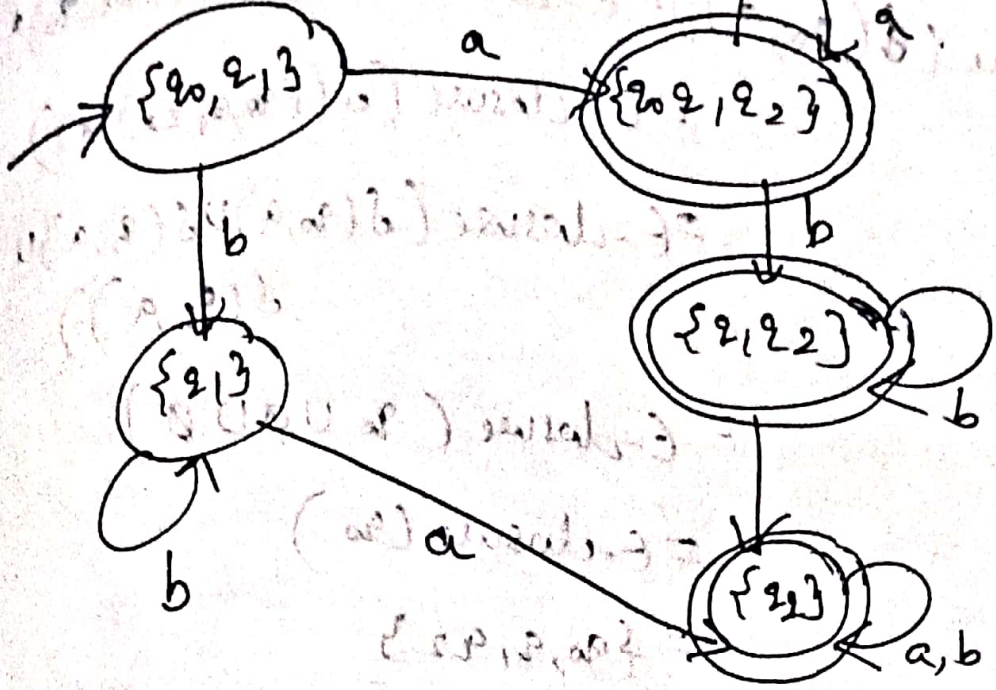
$= \{q_0, q_1, q_2\}$

$\delta(\{q_0, q_1\}, b) = \epsilon\text{-closure}(\delta(\{q_0, q_1\}, b))$

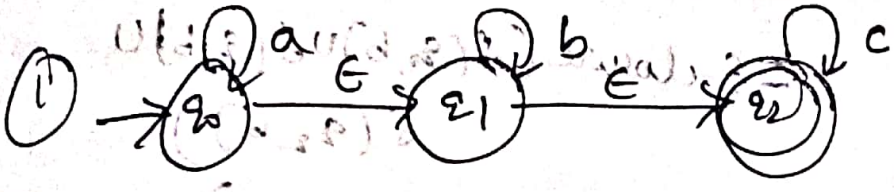
$= \epsilon\text{-closure}(\delta(q_0, b) \cup \delta(q_1, b))$

$= \epsilon\text{-closure}(\emptyset \cup q_1)$

$= q_1$



∈-NFA to NFA:



∈-NFA table

	a	b	c	∈	
→ q0	q0	∅	∅	q1	∈-closure(q0) = {q0,q1,q2}
q1	∅	q1	∅	q2	∈-closure(q1) = {q1,q2}
* q2	∅	∅	q2	∅	∈-closure(q2) = {q2}

NFA transition table δ:

	a	b	c
→ q0	{q0,q1,q2}	{q1,q2}	{q2}
* q1	∅	{q1,q2}	{q2}
* q2	∅	∅	{q2}

ϵ-closure $(\delta(q_0, a)) = \epsilon$-closure $(\delta(\epsilon$-closure$(q_0), a))$

$= \epsilon$-closure $(\delta(\{q_0, q_1, q_2\}, a))$

$= \epsilon$-closure $(\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a))$

$= \epsilon$-closure $(q_0 \cup \phi \cup \phi)$

$= \epsilon$-closure (q_0)

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

ϵ-closure $(\delta(q_0, b)) = \epsilon$-closure$(\delta(\epsilon$-closure$(q_0), b))$

$= \epsilon$-closure $(\delta(\{q_0, q_1, q_2\}, b))$

$= \epsilon$-closure $(\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b))$

$= \epsilon$-closure $(\phi \cup q_1 \cup \phi)$

$= \epsilon$-closure (q_1)

$= \{q_1, q_2\}$

ϵ-closure $(\delta(q_0, c)) = \epsilon$-closure $(\delta(\epsilon$-closure$(q_0), c))$

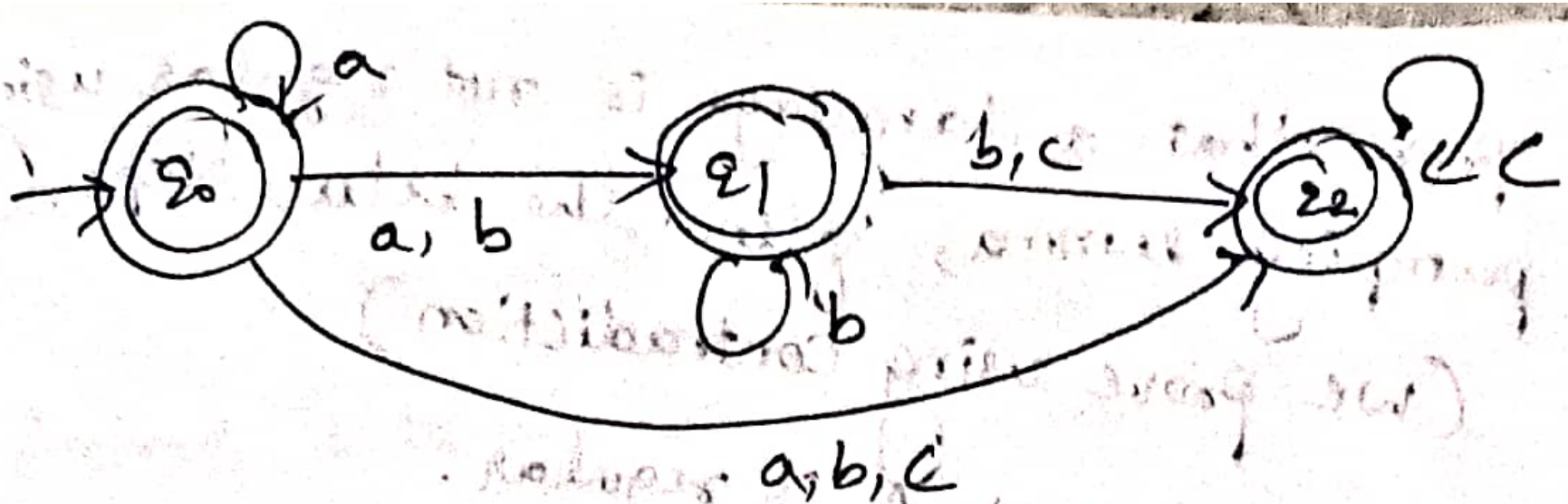
$= \epsilon$-closure $(\delta(\{q_0, q_1, q_2\}, c))$

$= \epsilon$-closure $(\delta(q_0, c) \cup \delta(q_1, c) \cup \delta(q_2, c))$

$= \epsilon$-closure $(\phi \cup \phi \cup q_2)$

$= \epsilon$-closure (q_2)

$= \{q_2\}$



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

$\Sigma = \{a, b, c\}$

$q_0 = \{q_0\}$

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