UNIT 2   NON-DETERMINISTIC FINITE AUTOMATA

2.0   INTRODUCTION

In our daily activities, we all encounter the use of various sequential circuits. The elevator control which remembers to let us out before it picks up people going in the opposite direction, the traffic-light systems on our roads, trains and subways, all these are examples of sequential circuits in action. Such systems can be mathematically represented by Finite state machines, also called finite automata or other powerful machine like turning machines. In the previous unit, we introduced the concept of Deterministic Finite Automata (DFA), in which on an input in a given state of the DFA, there is a unique next state of DFA. However, if we relax the condition of uniqueness of the next state in a finite automata, then we get Non-Deterministic Finite Automata (NFA).

A natural question which now arises is whether a non-deterministic automata can recognize sets of strings which cannot be recognized by a deterministic finite automata. At first, you may suspect that the added flexibility of non-deterministic finite automata increases their computational capabilities. However, as we shall now show, there exists an effective procedure for converting a non-deterministic FA into an equivalent deterministic one. This leads us to the conclusion that non-deterministic FA's and DFA's have identical computational capabilities.

2.1   OBJECTIVES

After studying this unit, you should be able to

define a non-deterministic finite automata;

show the equivalence of NFA and DFA;

compute any string or language in any NFA;

state and prove pumping lemma;

apply pumping lemma for a language which is not regular;

apply closure properties of regular language and finite automata; and

find an equivalent regular expression from a transition system and vice-versa.

In unit 1 we discussed about finite automata. You may wonder that in finite automata for each input symbol there exists a unique state for processing of it. Do you think that there may be more than one possible state, or there may not be any state for

processing of any letter. If for processing of any letter there is more than one state or none state, then, the auotmata is known as non-deterministic finite automata (NFA).

2.2 NON-DETERMINSTIC FINITE AUTOMATA

You have already studied finite automata (though 'automata' is a plural form of the noun 'automaton', the word 'automata' is also used in singular sense). Now consider an automata that accepts all and only strings ending in 01, represented diagrammatically, as follows:

Fig. 1: Transition Diagram

In the case of the finite automata shown in figure 1, the following points may be noted:

(i)   On input 0 in state q_0, the next state may be either of the two states viz., q_0 or q_1.
(ii)   There is no next state on input 0 in the state q_1.
(iii) There is no next state on input 0 and 1 in the state q_2.

In this transition system, what happens when this automata processes the input .00101?

Fig. 2: Processing of string 00101
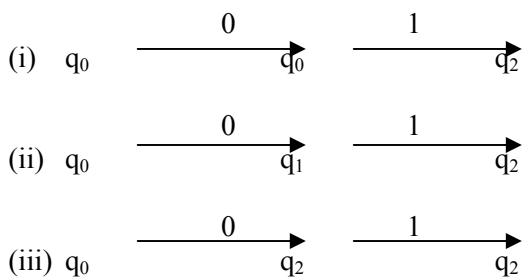
Here from the initial state q_0, for the processing of alphabet 0, there are two states at once or viewed another way, it can be 'guessed" which state to go to next. Such a finite automata allows to have a choice of 0 or more next states for each state input pair and is called a non-deterministic finite automata. An NFA can be in several states at once.

Fig. 3: Transition diagram

Before going to the formal definition of NFA, let us discuss one more case of non-determinism of finite automata. Suppose Q = {q_0, q_1, q_2},   = {0, 1}, q_0 is an initial state and q_2 is final state. Again, suppose the processing of any input symbol does not result in the transition to a unique state, but results a chain of states. Let us consider a machine given in figure 3.

For the sake of convenience, let us check the processing of any input symbol.   From the state q_0, after processing 0, resulting states are q_0, q_1, q_2 and for input symbol 1, there are three possible states q_0, q_1 and q_2 not a unique state. It clarifies that a non-deterministic automata can have more than one possible state or none state after processing any input symbol from   .

Let us check how the string 01 is processed by the above automata. Here we have three paths to reach to the final state:

(i) q_0 $\xrightarrow{\quad 0 \quad}$ q_0 $\xrightarrow{\quad 1 \quad}$ q_2

(ii) q_0 $\xrightarrow{\quad 0 \quad}$ q_1 $\xrightarrow{\quad 1 \quad}$ q_2

(iii) q_0 $\xrightarrow{\quad 0 \quad}$ q_2 $\xrightarrow{\quad 1 \quad}$ q_2

A generalisation which is obtained here by allowing of several states as a result of the processing of an input symbol is called non-determinism. If from any state, we can reach to several states or none state, then the finite automata becomes non-deterministic in nature.

Formally, a non-deterministic finite automata is a quintuple

   A = (Q,   ,   , q_0, F)

Where

* Q is a finite set of states
*   is a finite alphabet for inputs
*   is a transition function from Q   to the power set of Q i.e. to 2^Q
* q_0 Q is the start/initial state
* F   Q is a set of final/accepting states.

The NFA, for the example just considered, can be formally represented as:

   ({q_0, q_1, q_2}, {0,1},   , q_0, {q_2})

where   the transition function, is given by the table 1:

Table1

States	0	1
→q_0	{q_0,q_1}	{q_0}
q_1	Ø	{q_2}
q_2	Ø	Ø

Now, let us prove that the NFA

Fig. 4: NFA accepting x01

accepts the language $\{x01 : x \in \Sigma^*\}$ of all the strings that terminate with the sub-string 01. A mutual induction on the three statements below proves that the NFA accepts the given language.

1. $w \in \Sigma^* \quad q_0 \in \hat{\delta}(q_0, w)$
2. $q_1 \in \hat{\delta}(q_0, w) \quad w = x0$
3. $q_2 \in \hat{\delta}(q_0, w) \quad w = x01$

If $|w| = 0$ then $w = \epsilon$. Then statement (1) follows from def., and statement & (2) and (3) show that all the string x01 will be accepted by the above non-deterministic automata.

Example 1: Consider the NFA with the formal description as $(Q, \Sigma, \delta, q_0, F)$ where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, q_0 is the initial state and q_1 is only the final state, and δ is given by the following table:

Table 2

State	Input from	
	a	b
q_0	q_1, q_2	q_0
q_1	q_1	-
q_2	q_1	q_2

In NFA, though the function δ maps to a sub-set of the set of states, yet we generally drop braces, i.e., instead of $\{q_0, q_1\}$, we just write q_0, q_1.

The computation for an NFA is also similar to that of DFA. Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w is a string over the alphabet Σ. The string w is accepted by NFA if corresponding to the input sequence, there exists a sequence of transitions from the initial state to any of the possible final states.

Now, let us check computations (in NFA, there are many possible computations) of the string aba.

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

$$= \hat{\delta}(q_1, ba) \text{ or } \hat{\delta}(q_2, ba)$$

$$= \hat{\delta}(\delta(q_1, b), a) \text{ or } \hat{\delta}(\delta(q_2, b), a)$$

$$= \text{stuck or } \hat{\delta}(q_2, a)$$

$$= q_1 \text{ (an accepting state)}$$

The above sequence of states shows the final state q_1 which is an accepting state. Hence, the string aba is accepted by the system and the input sequence of states for the input is $q_0 \xrightarrow{a} q_2 \xrightarrow{b} q_2 \xrightarrow{a} (q_1)$

Try some exercises:

Ex.1)   Consider an NFA given in figure 5. Check whether the strings 001, 011101, 01110, 010 are accepted by the machine, or not?

Fig. 5

Ex.2) Give an NFA which accepts all the strings starting with ab over {a,b}.

In unit 1, we discussed DFA and in previous section we discussed NFA. Now a simple question arises, are these two automata equivalent? Reply for that is it is always possible to find an equivalent DFA to every NFA. In next section we shall discuss the equivalence of DFA and NFA.

2.3 EQUIVALENCE OF NFA AND DFA

Every time we find that if we are constructing an automata, then it is quite easy to form an NFA instead of DFA. So, it is necessary to convert an NFA into a DFA and this is also said to be equivalence of two automata. Two finite automata M and N are said to be equivalent if L(M) = L(N).

From the definitions of NFA and DFA, it is clear that they are similar in all respects except for the transition function. In DFA, the transition function takes a state and an input symbol to the next state, whereas in NFA, the transition function takes a state and an input symbol or the empty string into the set of possible next states. If empty string   is used as an input symbol, then the NFA is called   -NFA. As an NFA is obtained by relaxing some condition of DFA, intuitively it seems that there may be some NFAs to which no DFA may correspond. However, it will be shown below that by relaxing the condition, we are not able to enhance computational power of the DFAs. In other words, we establish that for each NFA, there is a DFA, so that both recognise/accept the same set of strings.

We now try to find the equivalence between DFA and NFA. Some DFA can be designed to simulate the behaviour of an NFA. Let us consider M = (Q,   ,   , q_0, F) be an NFA accepting L(M). We design a DFA, viz., M   as described below and show that the language accepted by M   is the same that accepted by M, i.e., the language L(M). M   = (Q,   ,   , q_0', F) where Q   = 2^Q (any state in Q   is denoted by [q_1, q_2 - - - q_j] where q_1, q_2 - - - q_j Q), q_0 = [q_0] and F   is the set of all subsets of Q containing an element of F.

Before defining   , let us look at the construction of Q , q_0 and F . Machine M is initially at q_0 state. But on application of an input symbol, say a, M can reach any of the states in   (q_0,a). So M   has to remember all these possible states at any point of time. Therefore, subsets of Q can be defined as the states of M . Initial state of M   is q_0, which is defined as [q_0]. A string w accepted by the machine M if a final state is one of the possible states M reaches on processing w. So, a final state in M   is any subset of Q containing some final state of M. Next we can define the transition function   as

$([q_1,.....q_N], a) = \overset{N}{\underset{i \ 1}{U}} (q_i, a)$. So, we have to apply to (q_i, a) for each $i = 1,2.....N$ and take their union to get $([q_1, q_2.....q_N], a)$. Defining with the help of in this way is also said to be **subset construction** approach.

Example 2: Construct a DFA equivalent to the NFA M, diagrammatically given by

Fig. 6: NFA

when for M is given in terms of a transition table, the construction is simpler. Now, let us have a look at the following table3.

Table 3

State/	0	1
q_0	q_0	q_1
q_1	q_1	q_0, q_1

(i)   In this given M, the set of states is $\{q_0, q_1\}$. The states in the equivalent DFA are the   subsets of the states given in the NFA. So the states in DFA are subsets of $\{q_0, q_1\}$, i.e.,   , $[q_0]$, $[q_1]$, $[q_0, q_1]$.

(ii) $[q_0]$ is the initial state.

(iii) $[q_0]$ and $[q_0, q_1]$ are the final states as these are the only states containing q_0, the only final state of M.

Therefore, F $= \{[q_0], [q_0, q_1]\}$

(iv)   is defined by the following state table:

Table 4

State/	0	1
$[q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

We start the construction by considering $[q_o]$ first. We get $[q_o]$ and $[q_1]$. Then, we construct   for $[q_1]$ we get $[q_1]$ and $[q_o, q_1]$. As $[q_1]$ already exists in left most column, so we construct   for $[q_o, q_1]$. We get $[q_0, q_1]$ and $[q_0, q_1]$. We do not get $[q_0, q_1]$ and $[q_0, q_1]$. We do not get any new states and so we terminate the construction of   .

When a non-deterministic finite automate has n states, the corresponding finite automata has 2^n states. However, it is not necessary to construct   for all these 2^n states, but only for those states reachable from the initial state. This is because our interest is only in constructing the equivalent DFA. Therefore, we start the construction of   for initial state and continue by considering only states appearing

earlier under input columns and constructing   for such states. If no more new states appear under the input columns, we halt.

To prove the equivalence of both automata, we will prove the following theorem:

Theorem1: A language L is accepted by some NFA if and only if it is accepted by some DFA.

In the theorem, there are two parts to prove:

If L is accepted by DFA M , then L is accepted by some NFA M.
If L is accepted by NFA M , then L is accepted by some DFA M .

The first is the easier to prove.

Theorem1(a) (one direction) : If L is accepted by DFA M , then L is accepted by some NFA M.

Proof : Let us compare the definitions of NFA and DFA.

Definition :   A Deterministic Finite Automata (DFA) M is
                    defined by the 5-tuple.

$M = (Q ,$, $, q_0', F)$ where
Q - The finite set of states.
   - The finite set of symbols, the input alphabet.
  - Transition function   : Q   Q.
q_0 - An initial state, q_0' $Q q_0'$.
F - A set of final states or accept states, F   Q .

Definition :   A Non-deterministic Finite Automata (NFA) M
                    is a 5-tuple

$M = (Q,$, $, q_0, F)$ where
Q - is a finite set of states.
   - is a finite input alphabet.
  - is a transition function : Q   ({ }) 2^Q.
q_0 Q is the start state.
F   Q, is the set of accepting states.

The above definitions follow that every DFA is also an NFA, which implies that if w   L(M), then w   L(M).

The other half of the theorem is in the following theorem:

Theorem1(b): If L is accepted by NFA M = (Q,   ,   , q_0, F), then L is accepted by some DFA M = $(Q ,$, $, q_0', F)$.

Proof : Construct M as in the Subset Construction Algorithm. We will show using induction on the length of w.

Base case : Let w be an empty string, i.e., if $|w| = 0$ then $w = \epsilon$. By definition of NFA and DFA both $\delta(q_0,w)$ and $\delta'(q_0',w)$ are in state $\{q_0\}$. Hence, the result.

Let us assume that this result is true for each string of length n, we will now show that this result is true for strings of length (n+1).

Let $w = sa$ with $|w| = (n+1)$ and $|s| = n$, also a is the final symbol fo w. As $|s| = n$, therefore, by induction

$$\delta'(q_0',s) = \delta(q_0 s)$$

If $\{P_1, P_2,\ldots P_k\}$ be the set of states for non-deterministic finite automata M, then

$$\delta(q_0,w) = \bigcup_{i=1}^{k} \delta(P_i, a) . \tag{i}$$

Next,

$$\delta'(\{P_1, P_2,\ldots P_k\},a) = \bigcup_{i=1}^{k} \delta(P_i, a) \tag{ii}$$

also $\delta'(q_0', s) = \{P_1, P_2,\ldots P_k\}$. $\tag{iii}$

Using Equations (i), (ii) and (iii) we get

$$\delta'(q_0', w) = \delta'(\delta'(q_0', s), a)$$
$$= \delta'(P_1, P_2,\ldots P_k\}, a)$$
$$= \bigcup_{i=1}^{k} \delta(P_i, a)$$
$$= \delta(q_0, w).$$

which shows that the result is true for $|w| = n+1$ when the result is true for a string of length n.

Here the result is true for length 0 and for length (n+1) which is implied by the length n.

Therefore, the given statement is true for all the strings.

Hence, M and M both accept the same string w iff $\delta'(q_0',w)$ or $\delta(q_0,w)$ contains a state in F, or F respectively. Therefore,

$$L(M) = L(M)$$

For every non-deterministic finite automaton, there exists an equivalent deterministic infinite automaton which accepts the same language. In this way, two finite automata, M and M are said to be equivalent if $L(M) = L(M)$.

Example 3: Construct a non-deterministic finite automata accepting the set of all strings over {a,b} ending in aba. Use it to construct a DFA accepting the same set of strings.

Solution: Required NFA is the one that accepts strings of the form xaba where $x \in \{a,b\}^*$

Fig. 7: NFA accepting all the string ended by aba

Transition table of the diagram shown in Figure 7 is given in table 5.

Table 5

State/	A	B
q_0	Q_0, q_1	q_0
q_1	-	q_2
q_2	q_3	-
q_3	-	-

Now, let us construct its equivalent DFA. [q_o] is the initial state in corresponding DFA so starting the function using [q_0] as an initial state, we represent it in table6.

Formally, the DFA is

A = ({[q_0], [q_0, q_1], [q_0, q_2], [q_0, q_1, q_3]}, {a,b}, , [q_0], {[[q_0, q_1, q_3]]})

where is given by the table 6.

Table 6

Diagramatically, DFA is given in figure 8.

Fig. 8: DFA

This example also highlights one of the reasons for studying NFAs. The reason is that generally, it is easier to construct an NFA that accepts a language than to construct the corresponding DFA.

Try some exercises to check your understanding:

Ex.3) Construct an NFA accepting {01, 10} and use it to find a DFA accepting the same.

Ex.4) M = ({q_1, q_2, q_3}, {0,1}, , q_1, {q_3}) is a NFA, where is given by

, (q_1, 0) = {q_2, q_3}, (q_1, 1) = {q_1}
, (q_2, 0) = {q_1, q_2}, (q_2, 1) =
, (q_3, 0) = {q_2}, (q_3, 1) = {q_1, q_2}

construct an equivalent DFA

Ex.5) Construct a transition system which can accept strings over the alphabet a,b, --------- containing either cat or rat.

Ex.6) Give examples of machines distinguishing DFA and NFA.

2.4 EQUIVALENCE OF -NFA AND NFA

There exist some transitions graphs when no input is applied. If no input is applied then the transition systems are associated with a null symbol . Every time we can find an equivalence in between the systems with -move and without -moves. With the help of an example, we shall find the equivalence of -NFA and NFA.

Suppose we want to remove -move from the transition shown in figure 9:

Fig. 9: -NFA

In the above transition q_0 is an initial state and q_f is a final state. For this, we proceed as follows:

If q_i, q_j are two states and null string is from q_i to q_j then :

(a) Duplicate all the edges starting from q_i which are starting from q_j

(b) If q_j is a final state, make q_i as a final state and if q_i is an initial state, make q_j as an initial state.

Now let us apply these two rules to the transition in figure 9. First of all, removing in between q_1 and q_f.

Fig. 10: Removal of one

Now, again apply the same rule to remove the remaining -move.

Fig. 11: After removing both

This transition system is free from and is equivalent to the -NFA.

2.5   PUMPING LEMMA

As you know that a language which can be defined by a regular expression is called a regular language, there are several questions related to regular languages that one can ask. The important one is: are all languages regular? The simple answer is no. The languages which are not regular are called non-regular languages. In this section, we give a basic result called "pumping lemma". Pumping lemma gives a necessary condition for an input string to belong to a regular set, and also states a method of pumping (generating) many input strings from a given strings all of which should be in the language if the language is regular. As this pumping lemma gives a necessary (but not sufficient) condition for a language to be regular, we cannot use this lemma to establish that a given language is regular, but we can use it to prove that a language is not regular by showing that the language does not obey the lemma.

The pumping lemma uses the **pigeonhole principle** which states that if p pigeons are placed into less than p holes, some hole has to have more than one pigeon in it. The same thing happens in the proof of pumping lemma. The pumping lemma is based on this fact that in a transition diagram with n states, any string of length greater than or equal to n must repeat some state.

Pumping lemma (PL):

If L is a regular language, then there exists a constant n such that every string w in L, of length n or more, can be written as w = xyz, where

(i) $|y| > 0$

(ii) $|xy|$ n

(iii) $xy^i z$ is in L, for all i   0 here y^i denotes y repeated i times and y^o =

Before proving this PL, a question that may have occurred by now is: Are there any languages that are not accepted by DFA's?

Consider the language L = {w   w=$0^k 1^k$, where k is a positive integer}.

Proof of (PL): Since we have L is regular, there must be a DFA, say A such that

> L = L(A)

Let A have n states, and a string w of length   n in L which is expressed as

> w = $a_1 a_2$ - - - - - - a_K where k   n with general elements a_i, a_j, for

> 1   i   j   k, the string w can be written as

> w = $a_1 a_2$ - - - - - - - $a_{i\ 1} a_i a_{i+1} a_{i+2}$ - - - - $a_{j\ 1} a_j a_{j+I}$ - - - a_k

> and w = xyz

> so x = $a_1 a_2$ - - - a_i

>   y = a_{i+1}, a_{i+2} - - - a_j

> and z = a_{j+1}, a_{j+2}- - - - a_k

Let q_0 be the initial state and further let

$q_1 = (q_0, a_1), q_2 = (q_0, a_1 a_2)$

q_i be the state in which A is after reading the first i symbols of w.

Since there are only n different states at least two of $q_0, q_1 - - - q_n$ which are (n+1) in numbers, must be same say, $q_i = q_j$ where $0 \leq i \leq j \leq n$. Then by repeating the loop from q_i to q_i with label $a_{i+1} - - - - a_j$ zero times once, or more, we get $xy^i z$ is accepted by A, because in case of each of the string $xy^i z$ for $i = 1, 2 \ldots$, the string when given as an input to the machine in the initial state q_0, reaches the final state q_n. Diagrammatically.

Fig. 12: representation of $xy^i z$

Hence $xy^i z \in L(A) \; \forall \; i \geq 0$.

How to use PL in establishing a given language as non-regular?

We use the PL to show that a language L is not regular through the following sequence of steps:

Step1: Start by assuming L is regular.

Step2: Suppose corresponding DFA has n states.

Step3: Choose a suitable w such that $w \in L$ with $|w| \geq n$.

Step4: Apply PL to show that there exists $i \geq 0$ such that $xy^i z \notin L$, where $w = xyz$ for some strings xyz.

Step5: Thus, we derive a contradiction by picking i, which concludes that assumption in step 1 is false.

Example 4: Consider $L = \{0^{n^2} | n \geq 0\}$.

Suppose L is regular. Then there exists a constant n satisfying the PL conditions.

Now $w = 0^{n^2} \in L$ and $|w| \geq n^2$

Write $w = xyz$; where $|xy| \leq n$ and $|y| > 0$ and hence $|y| \leq n$

By PL, $xyyz \in L$.

Here $|w| \geq n^2$

$\qquad |xyz| \geq n^2$

$\qquad |x| + |y| + |z| \geq n^2$

$\qquad n^2 + n \geq |x| + |y| + |y| + |z| . \geq n^2; \; [as \; |x| > 0 \; and \; |xy| \leq n]$

$\qquad n^2 + n \geq |xyyz| \geq n^2$

$$(n \quad 1)^2 \quad |xyyz| \quad n^2.$$

$(n+1)^2$ is the next perfect square after n^2, therefore,

xyyz is not of square length and is not in L. Since we have derived a contradiction, which concludes that L is not regular.

Let us try some exercises:

Ex.7) Show that the following languages are not regular

    (i) $L_1 = \{0^m 1^m : m \quad 0\}$

    (ii) $L_2 = \{0^i 1^j \ 2^K : 0 \quad i < j < K\}$

    (iii) $L_3 = \{a^p : p \text{ is prime}\}$

    (iv) $L_4 = \{ww \quad w \quad \{0,1\}^*\}$

    (v) $L_6 = \{0^n 1^{n!} \quad n > 0\}$

Ex.8) Give an example of a language which is not regular. Justify your answer.

2.6 CLOSURE PROPERTIES (Regular Languages and Finite Automata)

Suppose L and M are two regular languages, then if the operations applied to L and M results regular language, then the property is called closure property. The closure properties are very useful for regular languages and finite automata. The operations applied for regular languages produce regular language are union, intersection, concatenation, complementation, Kleenstar and difference. With the help of closure properties, we can easily construct the finite automata which accepts the language which is union, intersection, …, of regular languages.

Before discussing the closure properties, let us define a language of a DFA. Suppose $M = (Q, \quad , \quad , q_0, F)$, and the language accepted by M is L(M) and is defined as L(M) $= \{S \quad ^*(q_0, S) \quad F\}$. That is each string in L(M) is accepted by M. If L = L(M), then L is regular language. Let us discuss few theorems, showing the closure properties of regular languages and finite automata.

Theorem2: If L and M are regular languages, then L+M, LM and L^* are also regular languages.

L and M being given to be regular languages can be denoted by some regular expressions, say, **l** and **m**. Then, **(l+m)** denotes the language L+M. Also, the regular expression lm denotes the language LM. **(l)*** denotes the language L^*. Therefore, all three of these sets (i.e., languages) of words are definable by regular expressions, and hence are themselves regular languages.

Note: If any language can be denoted by a regular expression, then that language is by definition a regular language. ,

Complements and Intersection

Definition: If L is a language over the alphabet Σ, we define its complement, $\overline{L}$, to be the language of all strings of letters from Σ^* that are not in L, i.e., $\overline{L} = \Sigma^* - L$.

Example 5: Let L be the language over the alphabet $\Sigma = \{a,b\}$ having all the words which start with the letter a and no other words over Σ. Then, $\overline{L}$ is the language of the all other words that do not have the first letter as a.

Example 6: Suppose L is a language over $\{a,b\}$ ending with ba, then $\overline{L}$ is the language of over $\{a,b\}$ of all other words not ending with ba.

Theorem 3: If L is a regular language, then $\overline{L}$ is also a regular languages. In other words, the set of regular languages is closed under complementation.

Proof : We establish the result by constructing an FA say $\overline{M}$, the language $\overline{L}$. As L is given to be regular, therefore there is as FA, say M that recognizes L.

If L = Σ^*, then $\overline{L} = \emptyset$, which is, by definition, a regular language.

If L $\subset \Sigma^*$ is a regular language, then there is some FA that accepts the language L.

At least one of the states of the FA is a final state and as L $\subset \Sigma^*$, at least one of the states must not be a final state. The required FA has the same set of states, same set of input symbols, same transition function and same initial state as M. However, if S is the set of all states of M and F is the set of all final states of M, then set S – F of all non-final states of M serves as set of final states of the proposed FA viz. $\overline{M}$.

The fact that $\overline{M}$ is the FA that recognises the language $\overline{L}$, follows from the following:

Let x $\in \overline{L} = \Sigma^* - L \implies$ x $\notin$ L

$\implies$ the string x when given to M as input string in the initial state terminates in a non-final state of M, i.e., terminates in a state belonging to S – F.

$\implies \overline{M}$ accepts x

Theorem 4: If L and M are regular languages, then L $\cap$ M is also a regular language. In other words, the set of regular languages is closed under intersection.

Proof: We can prove this theorem in two ways: One by De Morgan's Law or by constructing an appropriate FA. Here the proof with the help of De Morgan's law is given, and leave the proof based on construction of an appropriate FA to the students as an exercise.

For any two general sets L and M, whether regular languages or not, by De Morgan's Laws, we have

L $\cap$ M = $\overline{(\overline{L} + \overline{M})}$.

In view of the fact that complement of a regular language is regular, the languages $\overline{L}$ and $\overline{M}$ are regular languages, given L and M are regular. Further, the fact that the sum of two regular languages is regular, makes $\overline{L} + \overline{M}$ as a regular language.

Hence, its complement $(\overline{L} + \overline{M}) = L$ M, is regular.

The following discussion, based on processing of two FAS in parallel, helps us in the construction of an FA for the union of two regular languages.

Example 7: Suppose we take the two machines whose state graphs are given in the figure below:

<div style="text-align:center">

(a) :M₁   (b):M₂

Fig.13

</div>

We can easily verify that the machine (M$_1$) of figure 14 accepts all strings (over {a, b}) which begin with two b's. The other machine (M$_2$) in figure 15accepts strings which end with two b's. Let's try to combine them into one machine which accepts strings which either begin or end with two b's.

Why not run both machines at the same time on an input? We could keep track of what state each machine is in, by placing pebbles upon the current states and then advancing them according to the transition functions of each machine. Both machines begin in their starting states, as pictured in the state graphs below:

<div style="text-align:center">

(a)   (b)

Fig. 14: Pebbel on s$_0$ and q$_0$

</div>

With pebbles on s_0 and q_0, if both machines now read the symbol b on their input tapes, they move the pebbles to new states and the machines assume the following configurations:

<div align="center">(a)   (b)</div>

<div align="center">**Fig. 15: Pebble on s_1 and q_1**</div>

with pebbles on s_1 and q_1. The pebbles have advanced according to the transition functions of the machines. Now let's have them both read as an a. At this point, they both advance their pebbles to the next state and enter the configurations

<div align="center">(a)   (b)</div>

<div align="center">**Fig. 16: Pebble on s_3 and q_0**</div>

With this picture in mind, let's trace the computations of both machines as they process several input strings. Pay particular attention to the *pairs* of states the machines go through. Let our first string be bbabb, which is accepted by both the machines.

<div align="center">**Table7**</div>

Input	B	b	a	b	b
M_1's states	s_0	s_1	s_2	s_2	s_2	s_2
M_2's states	q_0	q_1	q_2	q_0	q_1	q_2

Now, let us look at an input string which neither of the two machines accepts say babab.

<div align="center">**Table 8**</div>

Input	b	a	b	a	b
M_1's states	s_0	s_1	s_3	s_3	s_3	s_3
M_2's states	q_0	q_1	q_0	q_1	q_0	q_1

And finally, we consider the string baabb which will be accepted by M_2 but not M_1.

<div align="center">**Table 9**</div>

Input	b	a	a	b	b
M_1's states	s_0	s_1	s_3	s_3	s_3	s_3
M_2's states	q_0	q_1	q_0	q_0	q_1	q_2

If we imagine a multi-processing finite automaton with two processessors (one for M_1 and one for M_2), it would probably look just like the pictures given above. Each of its state is a pair of states, one from each machine, corresponding to the pebble positions. Then, if a pebble ended up on an accepting state, for either machine (that is, either s_2 or q_2), our multi-processing finite automaton would accept the string.
The above discussion helps us in seeing the truth of the following statement intuitively: We construct the required machine by simulating the multi-processing pebble machine discussed above.

Theorem 5: The class of sets accepted by finite automata is closed under union.

Proof Sketch : Let $M_1 = (S, \Sigma, \delta, s_0, F)$ and $M_2 = (Q, \Sigma, \delta, q_0, G)$ be two arbitrary finite automata. To prove the theorem, we must show that there is another machine (M_3) which accepts every string accepted by M_1 or M_2 and no other string.

We show that the required machine is $M_3 = (S \times Q, \Sigma, \delta, < s_0, q_0 >, H)$ where δ and H will be described presently.

The transition function δ is defined as

$$\delta(<s_i, q_i >, a) = < \delta(s_i, a), \delta(q_i, a)>.$$

It can easily be seen that δ is a function from $S \times Q$ to $S \times Q$.

A state in M_3 is a final state in M_3 if and only if either its first component is in F, i.e., is a final state of M_1 or its second component is in G, i.e., is a final state of M_2. In cross product notation, this is :

$$H = (F \times Q) \cup (S \times G).$$

This completes the definition of M_3. We can easily see that M_3 is indeed a finite automaton because it satisfies the definition of finite automata. We claim it does accept $T(M_1) \cup T(M_2)$ since it mimics the operation of our intuitive multi-processing pebble machine. The remainder of the formal proof (which we shall leave as an exercise) is merely an introduction on the length of input strings to show that for all strings x over the alphabet I:

$$x \in T(M_1) \cup T(M_2) \text{ iff } \delta^*(s_0,x) \in F \text{ or } \delta^*(q_0,x) \in G$$
$$\text{iff } \delta^*(<s_0, q_0 > x) \in H.$$

Thus, by construction we have shown that the class of sets accepted by finite automata is closed under union.

By manipulating the notation, we have shown that two finite automata given in figure 13 (a) and (b) can be combined in a special way to prove the desired result, as shown in figure 17.

Fig. 17: Union of M_1 and M_2

Note that not all pairs of states are included in the state graph. (For example, $<s_0, q_1>$ and $<s_1, q_2>$ are missing.) This is because it is impossible to get to these states from $<s_0, q_0>$.

This is indeed a complicated machine! But, if we are a bit clever, we might notice that if the machine enters state s_2, q_2, then it remains in one of the states (s_2, q_0), (s_2, q_1), (s_2, q_2) all of which are final states. We may replace all such stages of M_3 by a single state say $s_2 q_1$, which is also a final state and get a smaller but equivalent machine as shown in Figure 18:

Fig. 18: Reduced Union Machine

Now check your understanding by the following exercises.

Ex. 9)   For each of the following pairs of regular languages, L and M find a regular
expression and an FA that correspond to L   M:

L   M

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

2. $(a+ab)^*(a+)$ $(a+ba)^*a$

3. $(ab^*)^*$ $b(a+b)^*$

4. $(a+b)^*a$ $(a+b)^* aa (a+b)^*$

5.   All strings of even length $b(a+b)^*$
= $(aa+ab+ba+bb)^*$

2.7   EQUIVALENCE OF REGULAR EXPRESSION AND FA

As you have seen in Unit 1, all the regular languages can be written as regular
expression and vice-versa. Do you find any relation in regular expression and a
transition system? A regular expression can have   ,   , any input symbol, +, *,
concatenation. Let us find the transition system of these.

Fig. 19: Transition diagram equivalent to

Fig. 20: Transition diagram equivalent to Ø

Fig. 21: Transition diagram equivalent to a

Fig. 22: Transition diagram equivalent to R= P + Q

Fig. 23: Transition diagram equivalent to R = PQ

Fig. 24: Transition diagram equivalent to R = P*

Using above equivalence of regular expression and transition systems, we can easily make use of equivalence of -NFA and NFA and also of NFA and DFA, and finally we can find the equivalence between a regular expression and FA.

Example 8: Let us try to get the finite automata which is equivalent to regular expression $(a+b)^*$ $(ab+ba)$ $(a+b)^*$.

Step 1: Construction of equivalent -NFA is:

$(a+b)^*$ $(ab+ba)$ $(a+b)^*$ is

Fig. 25: A Complete regular expression

It is concatenation of $(a+b)^*$, $(ab+ba)$ and $(a+b)^*$,after applying concatenation we get

Fig. 26: After concatenation

Then removing the * from $(a+b)^*$ at both places and applying union rule for ab + aa we get

Fig. 27: after removing *and +

Now concatenating ab and ba, we get

Fig. 28: equivalent   -NFA

Step 2 : Construction of equivalent NFA, Let us remove every   one by one

Fig. 29: Removing

Fig. 30: Removing   and Minimizing the state
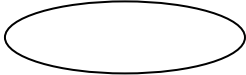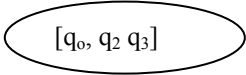
Fig. 31: Removing

Fig. 32: Removing

After minimizing the number of states, we get,

Fig. 33: Equivalent NFA

Step 3: Construction of equivalent DFA.

Table 10

States	Input	
	A	**b**
$[q_0]$	$[q_0, q_1]$	$[q_0, q_2]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_2, q_3]$
$[q_0, q_2]$	$[q_0, q_2, q_3]$	$[q_0, q_2]$
⬭	$[q_0, q_1, q_3]$	$[q_0, q_2, q_3]$
$[q_0, q_2\ q_3]$	$[q_0, q_1, q_3]$	$[q_0, q_2, q_3]$

Diagrammatically it is shown in Figure 34.

Now try some exercises.

Ex.10) Find the finite automata equivalent to the following regular expressions:

(i) $ba+(aa+b) a^*b$

(ii) $b+aa+aba^*b.$

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

As you have seen that there exists an equivalent NFA with -transitions, NFA without -transitions and DFA to each regular expression. But if there is some transition system, then there exists equivalent regular expression. The algorithm we are going to discuss for this purpose is not restricted to NFA, DFA. This algorithm can be applied to each transition system to each transition system to find its equivalent regular expression. We convert a transition system to a regular expression by reducing the states. These states are reduced by replacing each state one by one with a corresponding regular expression. The following steps are used:

If the label is (a, b), then it is replaced by **a+b**.

First of all, eliminate all the states which are not initial or final states. If we replace the state q_e from the transition given below,

Fig. 35

then q_e is eliminated by writing its corresponding regular expression $\mathbf{R_1R_2^*R_3+R_4}$ from q_a to q_b, as follows:

Fig. 36

Continue the process till only initial and final states remain.

If initial state is final state and the regular expression is **R**, such as

Fig. 37

the equivalent regular expression is $\mathbf{R}^*$

If initial state is not final state and is like,

Fig. 38

then this can also be written as

Fig. 39

which is the equivalent regular expression. If these are n final states and **R₁, R₂, R₃…Rₙ** are the regular expressions accepted by these states, then the regular expression accepted by the transition system will be $R_1+R_2+\dots\dots+R_n.$

Now let us try some examples to understand the algorithm well.

Example 9: Find the regular expression equivalent to the given system.

Fig. 40

There is no state which is neither initial n or final so this can be written as

Fig. 41

The equivalent r.e. is $b^*a(a+b)^*$.

Example10: Find a regular expression equivalent to

Fig. 42

Firstly, after eliminating q_2, we get

Fig. 43: Equivalent NFA

There are two final states, q_0 and q_1. The regular expression accepted by q_0 is a^* and the regular expression accepted by q_1 is a^*bb^*. Then, the regular expression accepted by the transition system is

$$a^* + a^*(bb^*) = a^*(\ +bb^*) \text{ (distributive property)}$$

$$= a^*b^* \text{ is the equivalent regular expression.}$$

Try some exercise.

Ex.11) Take any regular expression and find the transition system. Using this transition system, find equivalent regular expression and check your result.

2.8   SUMMARY

In this unit, we have covered the following:

1.   Non-deterministic finite automata.

2.   There exist, an equivalent DFA for every NFA.
3.   Two Automata M and N are said to be equivalent iff L(M) = (L(N).

4.   Pumping lemma with its proof.

5.   Application of pumping lemma in establishing a given language as non-regular.

6.   Closure properties of regular language and finite automata.

7.   Equivalence of regular expression and finite automata. The regular language can be found from a regular expression as well as finite automata. So, these two approaches of regular languages are equivalent.

2.9   SOLUTIONS/ANSWERS

Ex.1)   is given by

State	Input	
	0	1
q_0	q_0, q_1	q_0
q_1	-	q_2
q_2	-	-

$(q_0, 001) = (q_0, 01) = (q_1, 1) = q_2$ (Accepting state)

$(q_0, 011101) = (q_0, 11101)$

$\qquad = (q_0, 1101)$

$\qquad = (q_0, 101)$

$\qquad = (q_0, 01)$

$\qquad = (q_0, 1)$

$\qquad = \boxed{q_2}$ (accepting state)

$(q_0, 01110) = (q_0, 1110)$

$\qquad = (q_0, 110)$

$\qquad = (q_0, 10)$

$\qquad = (q_0, 0)$

$\qquad = q_0$ or q_1 (Not an accepting state)

$(q_0, 010) = (q_0, 10)$

$\qquad = (q_0, 0)$

$\qquad = q_0$ or q_1 (Not an accepting state)

So, strings 001 and 011101 are accepted by the given automata.

Ex.2)

Fig. 44

Ex.3)   NFA

Fig. 45

Transition function is given in the table below:

States	0	1
q_0	q_1	q_2
q_1	-	q_3
q_2	-	-
(q_3)	-	-

Equivalent DFA is

States	0	1
$[q_0]$	$[q_1]$	$[q_2]$
$[q_1]$	[ ]	$[q_3]$
$[q_2]$	$[q_3]$	[ ]
$[q_3]$	[ ]	[ ]

Ex.4)   DFA is

State	0	1
$[q_1]$	$[q_2, q_3]$	$[q_1]$
$[q_2, q_3]$	$[q_1 q_2]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_1, q_2, q_3]$	$[q_1]$
$[q_1, q_2, q_3]$	$[q_1, q_2, q_3]$	$[q_1, q_2]$

Ex.5)

Fig. 46

Ex.7)   (i) $L_1 = \{0^m 1^m : m \geq 0\}$

$w = xyz = 0^m 1^m$ $|w| \geq 2m$.

Consider three cases

Iff $y = 0^k$ $w = 0^{m-k} 0^k 1^m$

$xy^i z = 0^{m-k} 0^{ik} 1^m$

$xy^i z = 0^{m+(I-1)k} 1^m$

   $xy^i z \notin L_1$ as $m+(i-1)k \neq m$.

Similarly, case II with $y = 0^k 1^l$ and case III with $y = 1^k$ can be assumed and will not belong to L_1.

So, L_1 is not regular

(ii)   Similarly, as part (i)

(iii) L_3 {a^p:p is prime}

Suppose $y = a^m$, m>0 and w = xyz = a^p so $|w|$ p

$xy^i z = xyy^{i-1} z$

$$xy^i z = |x| \quad |y| \quad (i \quad 1)|y| \quad |z|$$

$$= |w| \quad (i \quad 1)|y|$$

$$= p \quad (i \quad 1)|y|$$

$$= p \quad (i \quad 1)m.$$

If we choose i   1 a multiple of p, then we get

$xy^i z = p+kpm$

$= (1+km)p$

which is not a prime number

so, $xy^i z \quad L_3$

(iv)   Suppose L is regular, and n be the number of states in automata M.

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

Let us consider ww = $01^n 01^n$ L_4

and $|ww|$ 2(n   1)   n.

I case : y has no 0's, i.e., y = 1^k; k   1
II case : y has only one 0.
Here, y cannot have two 0's. If so $|y|$ n   2. But $|y|$ $|xy|$ n.

In case I, assume i = 0. Then $xy^i z$ = xz and is of the form 0 1^m 0 1^n, where m = n   k < n or of the form 0 1^n 0 1^m. These both values cannot be written in form of ww with w   {0, 1}* and so xz   L. In case II also, take i = 0 then 0 will be removed and xz = 0 1^n 1^n again this cannot be written in ww form. Thus, in both the cases we get a contradiction. Therefore, L is not regular.

(v)   Left as an exercise.

Ex.8) Any example of a language may be given which is not regular. Use again pumping lemma to justify

Ex.9) 1.
       2.   **(a*)a**
       3.
       4.   **(a+b)* aa**
       5.   **(aa+ab+ba+bb)***

Ex.10) (i)   Equivalent NFA is

Fig. 47

(ii)   NFA is

Fig. 48