Chapter1

FINITE AUTOMATA

Finite automata are basic modules of a computer system, which addresses simple real time problems. Figure 1.1 shows conceptual view of finite automata. It has an input tape, read head and finite controller. Input tape is divided into finite cells and each cell stores one input symbols. Read head is used to read the input symbol from the input tape. The finite controller used to record the next state of finite automat.

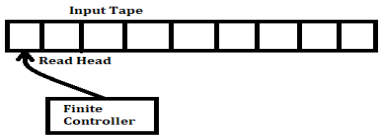


Figure 1.1: Conceptual view of finite automata

Before discussing finite automata in detail, it is very much important to know the basic requirement for the finite automata. They are

i. Sets.
ii. Graphs.
iii. Languages.
iv. Star closure.
v. Plus closure.

Sets (S):

A set is collection of elements, each element has its own identity. A set can be represented by enclosing its elements in curly braces. for example, the set of small letters a, b, c is shown as

S= {a, b, c}

Set {2, 4, 6,…} denotes the set of all positive even integers. we can use more explicit notation, in which we write

S={i | i is > 0 and is even}

We can perform following operations on sets, they are

a. union (∪),
b. intersection (∩), and
c. difference (−)
d. complementation defined as

$S_1 \cup S_2 = \{x \mid x \in S_1 \text{ or } x \in S_2\}$

$S_1 \cap S_2 = \{x \mid x \in S_1 \text{ and } x \in S_2\}$

$S_1 - S_2 = \{x \mid x \in S_1 \text{ and } x \notin S_2\}$

The complement of a set S, denoted by Ṡ consists of all elements not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$\dot{S}=\{x \mid x \in U, x \notin S\}$

The set with no elements, called the empty set or the null set, is denoted by ε. From the definition of a set, it is obvious that

S U ε = S - ε =S

S ∩ ε = ε

ε̅= U

De Morgan's laws: The following useful identities, known as De Morgan's laws.

$\overline{S_1 \cup S_2} = S_1 \cap S_2$

$\overline{S_1 \cap S_2} = S_1 \cup S_2$

are needed on several occasions.

Subset and Proper Subset: A set S_1 is said to be a subset of S, every element of S_1 is also an element of S. We write this as $S_1 \subseteq S$. If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a proper subset of S; we write this as $S_1 \subset S$.

Disjoint Sets: If S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varepsilon$ then the sets are said to be disjoint.

Finite and Infinite sets: A set is said to be finite if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by |S|.

Power- Set: A given set has many subsets. The set of all subsets is called the power-set and is denoted by 2^s. Observe that 2^s is a set of sets.

If S is the set {a, b, c}, then its powerset is

2^s= {ε,{a}, {b}, {c}, {a, b},{b, c},{a, c}, {a, b. c}}

Here |S| = 3 and $|2^s| = 8$.

Cartesian product: Ordered sequences of elements from other sets are said to be the Cartesian product. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$S= S_1 \times S_2$

Let S1 = {2, 4} and S2 = {2, 3, 5, 6}. Then S1 × S2 = {(2, 2), (2, 3), (2, 5), (2, 6), (4, 2), (4, 3), (4, 5), (4, 6)}. Note that the order in which the elements of a pair are written matters. The pair (4, 2) is in S1 × S2, but (2, 4) is not.

Graphs:

A graph is a construct consisting of two finite sets, the set V = {υ1, υ2,…, υn} of vertices and the set E = {e1, e2,…, em} of edges. Each edge is a pair of vertices from V, for instance,

$e_i=(v, v_j)$.

Languages:

A finite, nonempty set of symbols are called the alphabet and is represented with symbol ∑. From the individual symbols we construct strings, which are finite sequences of symbols from the alphabet. For example, if the alphabet ∑ = {0}, then ∑* to denote the set of strings obtained by concatenating zero or more symbols from ∑.

We can write ∑*= {ε, 0, 00, 000 …}

If ∑ = {0, 1}
0 and 1 are input alphabets or symbols, then:
(0^2=00, 0^3=000, 1^2=11, 1^3=111, where as in mathematics they are numbers 0^2=0, 1^2=1)
L=∑*={ ε, 0, 1, 00, 01, 10, 11, 000, 001, 010 …….}

Kleen closure(*closure):
*closure = {$\sum^0 U \sum^1 U \sum^2 U \sum^3$ ……}
$\sum^0$={ ε} $\sum^1$={0, 1} $\sum^2$={00, 01, 10, 11}
$\sum^3$={000, 001, 010, 011, 100, 101, 110, 111}
Therefore ∑*={ ε , 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101,……}
$^+$Closure on ∑ = {0, 1}
$\sum^+$= {$\sum^1 U \sum^2 U \sum^3$…}
$\sum^* = \sum^+ + \varepsilon$

Therefore $\sum^+ = \sum^* - \varepsilon$

Formal Languages and Automata Theory: IT is a mathematical model used to design computer programs and sequential logic circuits. It is state machines which captures and makes transition to accept or reject.
Finite automata has input tape, read head and finite controller **Input tape**: It is divided into finite cell and each cell is used to store one input symbol. **Read head**: Read head is used to read input symbol from input tape. **Finite Controller**: Finite controller records, the transition to next state. The default value of finite controller is initial state q_0. States are represented with circles; they are of three types,

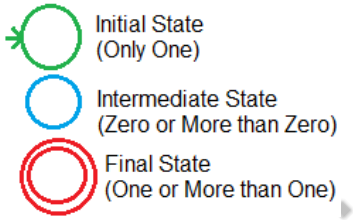


Figure 1.2: Different types of states used in finite automata

States are connected by lines called transition. It has two ends, one end is called initiation point and the other end is called termination point (>).

Initiation ⟶ Termination

The combination of transitions and states is called a transition diagram.

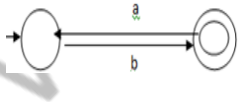


Figure 1.3: A typical transition diagram

Figure 1.4 shows a transition diagram to map cities and their roads.

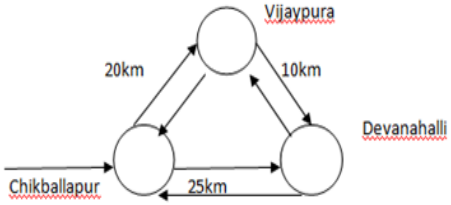


Figure 1.4: Finite automata with real time example.

Finite Automata are classified into two types. They are:
1) Deterministic Finite Automata (DFA)
2) Non Deterministic Finite Automata (NFA)
In Deterministic Finite Automata number of outgoing transitions are well defined from each state depending upon input symbols. For example, if there is one Input symbol, then you find one outgoing transition from each state, for two input symbols two outgoing transitions are present and so on.

Example 1.1: Draw the DFA, the input symbol is ∑= {0}

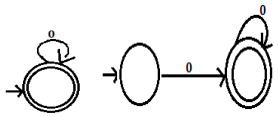


Figure 1.5: DFA's for input symbol ∑= {0}.

Example 1.2: Draw the DFA, the input symbol is ∑= {0, 1}

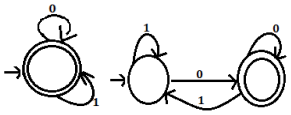


Figure 1.6: DFA's for input symbol ∑= {0, 1}.

Definition of DFA: we define DFA as,

M_{DFA} = (Q, ∑, δ, Initial state, Final state)

Where, Q is set of states, ∑ =set of input symbols, δ: transition function is $Q\times\Sigma \rightarrow Q$

From the table 1.

$Q=\{q_0,q_1,q_2\}$ ∑ ={a, b, c}

δ:
$\delta(q_0, a)=q_1$ $\delta(q_0,b)=q_0$ $\delta(q_0,c)=q_2$
$\delta(q_1,a)=q_1$ $\delta(q_1,b)=q_2$ $\delta(q_1,c)=q_0$
$\delta(q_2,a)=q_2$ $\delta(q_2,b)=q_1$ $\delta(q_2,c)=q_2$
Note: Initial State is q_0 and final state is q_2

Table 1.1: Transition table

Q\∑	a	b	c
→q_0	q_1	q_0	q_2
q_1	q_1	q_2	q_0
*q_2	q_2	q_1	q_2

Nondeterministic Finite Automata (NFA):

If outgoing transitions are not dependent on number of input symbols, then such automata are called as Nondeterministic Finite Automata.

For example: ∑ = {0, 1}

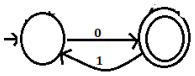


Figure 1.7: Nondeterministic Finite Automata for ∑ = {0, 1}

There is no outgoing transition from initial state, for a symbol '1 'and similarly there is no outgoing transition from final state for '0'.

Transition Function for string:

Note: 1) δ is the transition function, used for symbols.

2) For the string (Set of symbols), $\hat{\delta}$ is used

{ } = ε

{a}=a

$\hat{\delta}$ (q_0, ε)= q_0 //q_0 is reading empty string, string is empty therefore, transition to same state q_0.

We already Proved $\hat{\delta}(q_0,\varepsilon) = q_0$

$\hat{\delta}$ (q_0, a)= $\hat{\delta}$ (q_0, εa)= δ ($\hat{\delta}$ (q_0, ε),a)= δ (,a)

Therefore $\hat{\delta}$ (q_0, a)= δ (q_0, a)

Therefore for single symbol string $\hat{\delta}$ = δ

$\hat{\delta}$ (q_0, wa)

Where 'w' is the multi-symbol string and 'a' is a symbol, hence we separate the transition function for strings and symbols as given below:

δ ($\hat{\delta}$ (q_0, w), a)

Example 1.3: Write a transition table for a DFA shown in figure 1.8.

DFA over alphabet ∑ = {a, b}

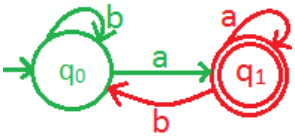


Figure 1.8: Transition diagram o0f a DFA.

Table 1.2: Transition table for the figure 1.8

Q \ ∑	a	b
→q0	δ(q0,a)	δ(q0, b)
*q1	δ(q1, a)	δ(q1,b)

Q \ ∑	a	b
→q0	q1	q0
*q1	q1	q0

M_{DFA} =(Q, ∑, δ , IS, FS)
Q ={ q_0, q_1}
∑ ={a ,b}
δ (q_0, a) =q_1 S δ q_0,b) =q_0
δ (q_1, a) =q_1 δ (q_1, b) =q_0
IS =q_0 FS={q_0}

Example 1.4: Construct transition table for the diagram shown in figure 1.9, over the alphabet ∑={0, 1, 2}

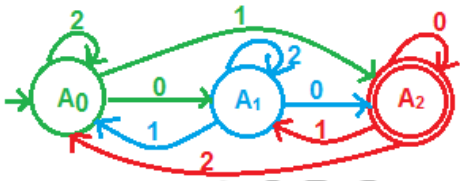


Figure 1.9: Transition diagram

Table 1.3: Transition table for the figure 1.9

A \ ∑	0	1	2
→A0	A1	*A2	A0
A1	*A2	A0	A1
*A2	*A2	A1	A0

We can define above DFA as: M_{DFA}=(A, ∑, δ, IS, {FS})

δ:
δ (A_0, 0)=A_1 δ (A_0, 1)=A_2 δ(A_0, 2)=A_0
δ (A_1, 0)=A_2 δ (A_1, 1)=A_0 δ(A_1, 2)=A_1

$\delta(A_2, 0)=A_2$ $\delta(A_2, 1)=A_1$ $\delta(A_2, 2)=A_0$

$A=\{A_0,A_1,A_2\}$; $\sum=\{0,1,2\}$; $IS=A_0$; $FS=A_2$

String Acceptance:

The first symbol of string begins with initial state. Later, traverses zero or more than zero intermediate state(s) for the intermediate symbol(s). Finally last symbol of the string ends with final state of automata. This is called string accepted or otherwise string is not accepted.

Example 1.5: Find out whether the given string is accepted by the DFA. Assume the string as abab. From the given string $\sum = \{a,b\}$. Note that always string is traversed by left side to right side,

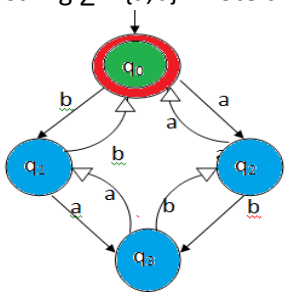


Figure 1.10: Transition diagram for example 1.5.

$\hat{\delta}(q_0, a)= \delta(q_0, a)= q_2$ —— 1

$\hat{\delta}(q_0, ab)= \delta(\hat{\delta}(q_0, a)\ b)$ from 1 $\hat{\delta}(q_0, a)= q_2$

$= \delta(q_2, b) = q_3$ —— 2

$\hat{\delta}(q_0, aba)= \delta(\hat{\delta}(q_0, ab), a)$ **from equation 2** $\hat{\delta}(q_0, ab)= q_3$

$= \delta(q_3, a)=q_1$

$\hat{\delta}(q_0, abab)= \delta(\hat{\delta}(q_0, aba), b)$

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

Given string is accepted by a DFA.

Find out whether given string 10110 is accepted by DFA.

From given string $\sum=\{0,1\}$

10110

$(q_0, 1)=\delta(q_0, 1)=q_2$ ------- ①

10110

$(q_0, 10)=\delta(\ (q_0, 1),0)=\delta(q_2, 0)=q_2$ ------- ②

10110 $(q_0, 101)=\ (\ (q_0, 10),1)=\delta(q_2, 1)=q_1$

10110 $(q_0, 1011)=\delta(\ (q_0, 101),1)=\delta(q_1, 1)=q_2$

10110 $(q_0, 10110)=\delta(\ (q_0, 1011),0)=\delta(q_2, 1)=q_2$

q2 is a final state hence given string 10110 is accepted.

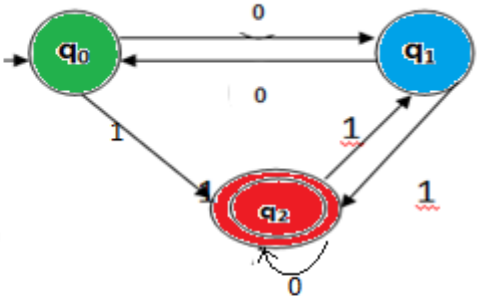


Figure 1.11 : Transition diagram

Example 1.6: Find out the language accepted by the DFA.

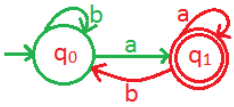


Figure 1.12: Transition diagram of Example 1.6.

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

L={a, aa, ba, baa, aba, …..}

In general we can write:

L= {w |w is a string ending with symbol a}

Example 1.7: Find out the language accepted by the DFA.

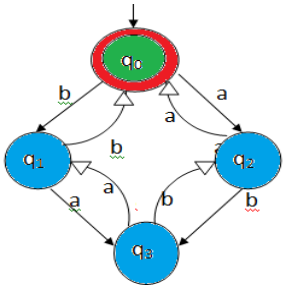


Figure 1.13: Transition diagram of Example 1.7

Initial state is also a final state, hence it accept empty string ε.

$\hat{\delta}(q_0, \varepsilon)=q_0$;

$\hat{\delta}(q_0, aa)=\delta(\hat{\delta}(q_0, a)a)=\delta(q_2, a)=q_0$;

$\hat{\delta}(q_0, bb)=\delta(\hat{\delta}(q_0, b)b)=\delta(q_1, b)=q_0$;

Therefore language L={ ε, aa, bb, aabb, abab…}

In general we can write L={w |w is a string consist of even number of a's and even number of b's}

Design of DFA :

Example 1.8: *Design a DFA which accept all the strings ends with symbol a over the alphabet ∑={a, b}.*

L={a, aa, ba, aba, bba…}

i. Draw the NFA for the smallest string a of the language.
The NFA for the string a is

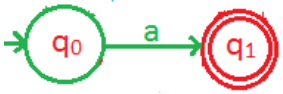


Figure 1.14 : NFA for the string a

Convert all states of NFA to DFA, that is from each state two outgoing transitions, one for symbol 'a' another for 'b'. From initial state only one outgoing transition on symbol 'a', one more outgoing transition is required that is on 'b'. If the condition in the given problem ends with, the outgoing transition on the other symbol b, definitely loop to a initial state.

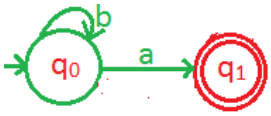


Figure 1.14: second outgoing transition from state q_0

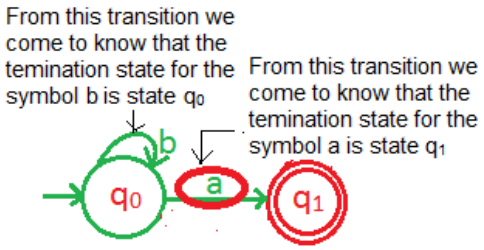
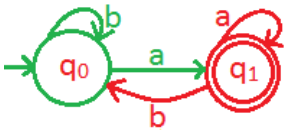


Figure 1.15 : q_0 is termination for symbol b and q_1 is termination for symbol a

Termination state for symbol 'a' is q_1 and for 'b' it is q_0. Outgoing transition from q_1 on symbol 'a' is loop, and for 'b' it is to q_0 .



Figyre 1.16 :DFA which accept all the strings ends with a.

We can define above DFA as M_{DFA} =(Q, ∑, δ , IS, {FS})
Q ={ q_0, q_1}
∑ ={a , b}
δ (q_0, a) =q_1 δ (q_0,b) =q_0
δ (q_1, a) =q_1 δ (q_1, b) =q_0
IS =q_0 FS={q_1}

Example 1.8: Design a DFA which accept all the strings ends with substring 01 over the alphabet

∑={0, 1}.

L={01, 001, 101, 0001,1001…}.

i. Draw the NFA for the smallest string 01 of the language.
The NFA for the string 01 is

Figure 1.17: NFA for the string 01.

ii. Convert all states of NFA to DFA, that is from each state two outgoing transitions, one for symbol '0' another for '1'. From initial state only one outgoing transition on symbol '0', one

more outgoing transition is required that is on '1'. If the condition in the given problem is, ends with, the outgoing transition on the other symbol 1, definitely loop to a initial state.

Figure 1.18 : conversion of NFA state A_0 to DFA

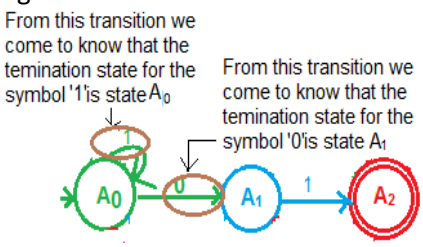
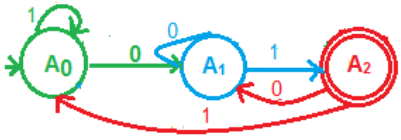


Figure 1.19 : A_0 is termination for symbol 1 and A_1 is termination for symbol 0

Termination state for symbol '0' is A_1 and for '1' it is A_0. Outgoing transition from A_1 on symbol '0' is loop. From A_2 on symbol '0' to A_1 and on symboll '1' to A_0.



Figyre 1.20 :DFA which accept all the strings ends with 01.

We can define above DFA as $M_{DFA}=(A, \sum, \delta, IS, \{FS\})$

$A=\{ A_0, A_1, A_2\}$; $\sum=\{0, 1\}$

δ :

$\delta (A_0, 0)=A_1$ $\delta (A_0, 1)=A_0$

$\delta (A_1, 0)=A_1$ $\delta (A_1, 1)=A_2$

$\delta (A_2, 0)=A_1$ $\delta (A_2, 1)=A_0$

$IS=A_0$; $FS=A_2$

Example 1.9: Design a DFA which accept all the strings ends with substring 012 over the alphabet

$\sum=\{0, 1, 2\}$.

L={012, 0012, 1012, 00012,10012…}.

i. Draw the NFA for the smallest string 012 of the language.
The NFA for the string 012 is

Figure 1.21: NFA for the string 012.

ii. Convert all states of NFA to DFA, that is from each state three outgoing transitions, one for symbol '0' second for '1' third from '2'. From initial state only one outgoing transition on symbol '0', two more outgoing transitions are required that is on '1' and '2'. If the condition

in the given problem ends with, the outgoing transition on the other symbols '1'and '2', definitely loop to a initial state.

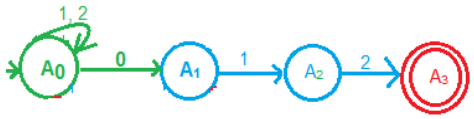


Figure 1.22: conversion of NFA state A_0 to DFA state.

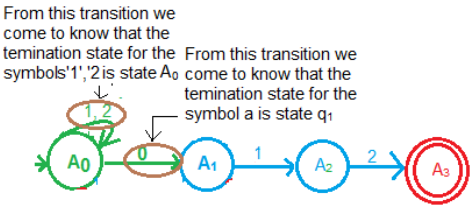


Figure 1.23: A_0 is termination for symbol 1, 2 and A_1 is termination for symbol 0

Termination state for symbol '0' is A_1 and for '1' and '2' is A_0. Outgoing transition from A_1 on symbol '0' is loop and for'2' is A_0. From A_2 on symbol '0' to A_1 and on symbol '1' to A_0. From A_3 on symbol '0' to A_1 and on symbols '1' and '2' to A_0.

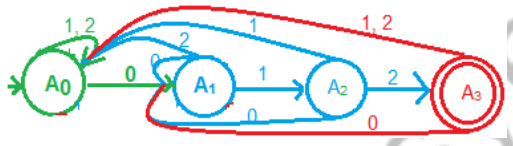


Figure 1.24 :DFA which accept all the strings ends with 012.

We can define above DFA as $M_{DFA}=(A, \sum, \delta, IS, \{FS\})$

$A=\{ A_0, A_1, A_2, A_3\}$; $\sum=\{0, 1, 2\}$

δ :

$\delta (A_0, 0)=A_1$ $\delta (A_0, 1)=A_0$ $\delta(A_0, 2)=A_0$
$\delta (A_1, 0)=A_1$ $\delta (A_1, 1)=A_2$ $\delta(A_1, 2)=A_0$
$\delta (A_2, 0)=A_1$ $\delta (A_2, 1)=A_0$ $\delta(A_2, 2)=A_3$
$\delta (A_3, 0)=A_1$ $\delta (A_3, 1)=A_0$ $\delta(A_3, 2)=A_0$
$IS=A_0$; $FS=A_3$

Example 1.10: Design a DFA which accept all the strings begins with symbol 'a' over the alphabet

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

L={a, aa, ab, abb, aba, aaa…}

i. Draw the NFA for the smallest string a of the language.
The NFA for the string a is

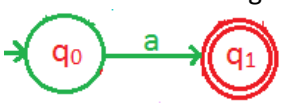


Figure 1.25: NFA for the string a.

Convert all states of NFA to DFA, that is from each state two outgoing transitions, one for symbol 'a' another for 'b'. From initial state only one outgoing transition on symbol 'a', one more outgoing transition is required that is on 'b'. If the condition in the given problem is, begin with, the outgoing transition on the other symbol 'b', is to new state called dead state.

Dead state: In finite automata the string travers from initial state to final state, in some situation we make finite automata to die called dead state. Dead state is a non-final state, from which all outgoing transitions are loop to the same state.

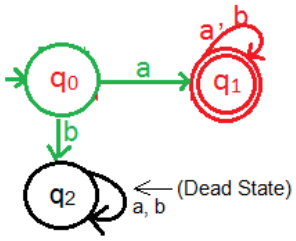


Figure 1.26 : DFA which accept all the string begin with a.

ii. Condition is begin, outgoing transitions from final state is loop.

We can define above DFA as M_{DFA} =(Q, ∑, δ , IS, {FS})

Q ={ q_0, q_1, q_2}

∑ ={a , b}

δ (q_0, a) =q_1 δ (q_0,b) =q_2

δ (q_1, a) =q_1 δ (q_1, b) =q_1

δ (q_2, a) =q_2 δ (q_2, b) =q_2

IS =q_0 FS={q_1}

Example 1.11: Design a DFA which accept all the strings begins with substring 01 over the alphabet

∑={0, 1}.

L={01, 010, 011, 0100, 0101…}.

i. Draw the NFA for the smallest string 01 of the language.
The NFA for the string 01 is

Figure 1.27: NFA for the string 01.

ii. Convert all states of NFA to DFA, that is from each state two outgoing transitions, one for symbol '0' another for '1'. From initial and intermediate states only one outgoing transition, one more outgoing transition is required. If the condition in the given problem is, begin with, the outgoing transition from both the states to new state called dead state.

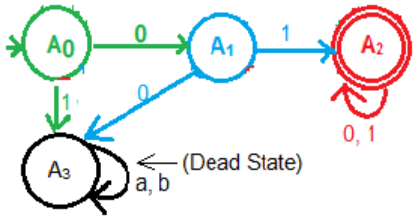


Figure 1.28 : DFA which accept all the string begin with 01.

iii. Condition is begin, outgoing transitions from final state A_2 on symbol '0' and '1' is loop.

We can define above DFA as $M_{DFA}=(A, \sum, \delta, IS, \{FS\})$

$A=\{ A_0, A_1, A_2 A_3\}$; $\sum=\{0, 1\}$

δ :

$\delta (A_0, 0)=A_1$ $\delta (A_0, 1)=A_0$

$\delta (A_1, 0)=A_1$ $\delta (A_1, 1)=A_2$

$\delta (A_2, 0)=A_1$ $\delta (A_2, 1)=A_0$

$\delta (A_3, 0)=A_3$ $\delta (A_3, 1)=A_3$

$IS=A_0$; $FS=A_2$

Example 1.12: Design a DFA which accept all the strings begins with 01 or ends with 01 or both over the alphabet $\sum=\{0, 1\}$.

L={01, 010, 011, 0100, 001, 101, 0001,1001, 0101, 01101…}

We have already designed both the DFAs, joining of these DFAs, is the solution for the given problem.

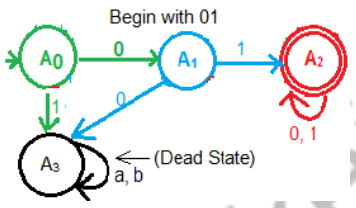


Figure 1.29: DFA which accept all the string begin with 01.

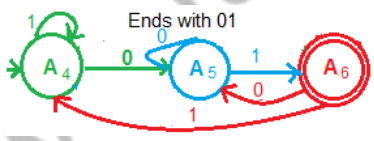
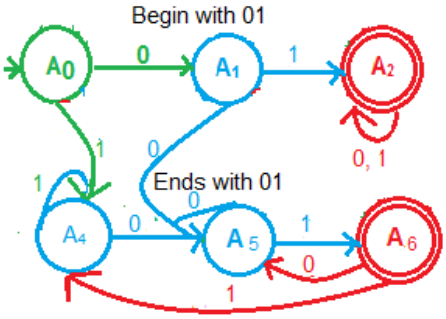


Figure 1.30: DFA which accept all the string ends with 01.



We can define designed DFA as $M_{DFA}=(A, \sum, \delta, IS, \{FS\})$

$A=\{A_0, A_1, A_2 A_4, A_5, A_6\}$ $\sum=\{0, 1\}$

δ :

$\delta(A_0, 0)=A_1$ $\delta(A_0, 1)=A_4$; $\delta(A_1, 0)=A_5$ $\delta(A_1, 1)=A_2$

$\delta(A_2, 0)=A_1$ $\delta(A_2, 1)=A_0$; $\delta(A_4, 0)=A_5$ $\delta(A_4, 1)=A_4$

$\delta(A_5, 0)=A_5$ $\delta(A_5, 1)=A_6$; $\delta(A_6, 0)=A_5$ $\delta(A_6, 1)=A_4$

$IS=A_0$; $FS=\{A_2, A_6\}$

Figure 1.31: DFA which accept all the string begin with 01, end with 01 begin and end with 01.

Example 1.13: Design a DFA which accept all the binary strings divisible by 2.

L={ ε, 10, 100, 110, …}

To design DFA we need to make the following assumptions.

$q_0 \rightarrow 0 \rightarrow q_0$

$q_0 \rightarrow 1 \rightarrow q_1$

$q_0 \rightarrow 10 \rightarrow q_0$

$q_0 \rightarrow 11 \rightarrow q_1$

$q_0 \rightarrow 100 \rightarrow q_0$

$q_0 \rightarrow 101 \rightarrow q_1$

From the above assumption, we can draw the following DFA.

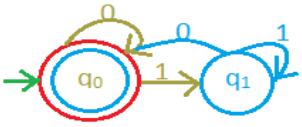


Figure 1.32: DFA which accepts all the strings divisible by 2.

We can define above DFA as $M_{DFA}=(Q, \sum, \delta, IS, \{FS\})$

$Q=\{q_0, q_1\}$

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

$\delta(q_0, 0)=q_0$ $\delta(q_0, 1)=q_1$

$\delta(q_1, 0)=q_0$ $\delta(q_1, 1)=q_1$

$IS=q_0$ $FS=\{q_0\}$

Example:

Example 1.14: Design a DFA which accept all the binary strings divisible by 5.

L={ ε, 101, 1010, 1111, …}

To design DFA we need to make the following assumptions.

$q_0 \rightarrow 0 \rightarrow q_0$

$q_0 \rightarrow 1 \rightarrow q_1$

$q_0 \rightarrow 10 \rightarrow q_2$

$q_0 \rightarrow 11 \rightarrow q_3$

$q_0 \rightarrow 100 \rightarrow q_4$

$q_0 \rightarrow 101 \rightarrow q_0$

$q_0 \rightarrow 110 \rightarrow q_1$

$q_0 \rightarrow 111 \rightarrow q_2$

$q_0 \rightarrow 1000 \rightarrow q_3$

$q_0 \rightarrow 1001 \rightarrow q_4$

$q_0 \rightarrow 1010 \rightarrow q_5$

According to assumption we can draw the following DFA

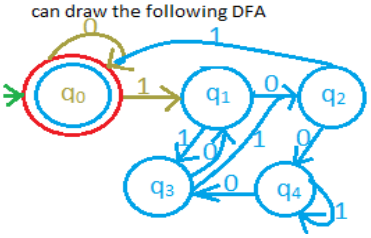


Figure 1.33: DFA which accept all the strings divisible by 5.

We can define, designed DFA as $M_{DFA}=(A, \sum, \delta, IS, \{FS\})$

$A=\{ q_0, q_1, q_2 q_3, q_4\}$ $\sum=\{0, 1\}$

δ :

$\delta (q_0, 0)=q_0$ $\delta (q_0, 1)=q_1$; $\delta (q_1, 0)=q_2$ $\delta (q_1, 1)=q_3$
$\delta (q_2, 0)=q_4$ $\delta (q_2, 1)=q_0$; $\delta (q_3, 0)=q_1$ $\delta (q_3, 1)=q_2$
$\delta (q_4, 0)=q_3$ $\delta (q_4, 1)=q_4$;

$IS=q_0$; $FS=q_0$

Example 1.15: Design a DFA which accept string over x, y every block of length 3 contains at least one x.

To design a DFA we need to make following assumptions,

1. If there is no 'x' in the string, such a string should not be accepted.

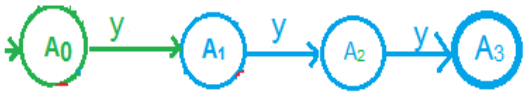


Figure 1.34: NFA which is not accepting three y's string.

2. One 'x' in the string, accepted by finite automata.

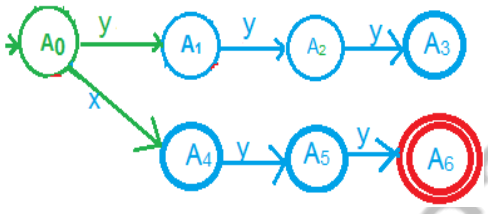


Figure 1.35: NFA which is accepting three symbol string in which one symbol is x.

3. The position of A_1 and A_5 is same as A_0 and A4. In between A_0 and A_4 the symbol is 'x', inbetwwen A_1 and A_5 is also x and so on.

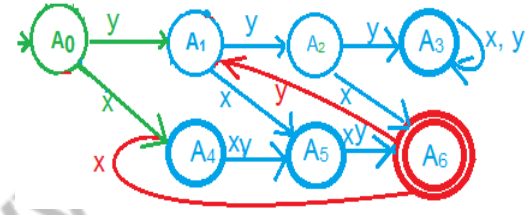


Figure 1.36: *DFA which accept string over x, y if every block of length 3 contains at least one x.*

We can define designed DFA as $M_{DFA}=(A, \sum, \delta, IS, \{FS\})$

$A= \{ A_0, A_1, A_2, A_4, A_4, A_5, A_6\}$ $\sum=\{x, y\}$

δ :

$\delta (A_0, x)= A_4$ $\delta (A_0, y)= A_1$; $\delta (A_1, x)=A_5$ $\delta (A_1, y)=A_2$
$\delta (A_2, x)=A_6$ $\delta (A_2, y)=A_3$; $\delta (A_3, x)=A_3$ $\delta (A_3, y)=A_3$
$\delta (A_4, x)=A_5$ $\delta (A_3, y)=A5$; $\delta (A_5, x)=A_6$ $\delta (A_5, y)=A_6$;

$\delta(A_6, x)=A_4$ $\delta(A_6, y)=A_1$

IS=A_0; FS={A_6}

Example 1.16: Design a DFA which accept even number of r and even number of s over the symbol ∑={r, s}.

i. Draw separate DFA, one accept even number of r's and another one accept even number of s's.

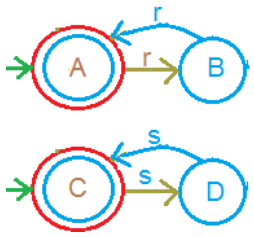


Figure 1.37 : *DFA's accept even symbols string over r, s.*

ii. Join two DFA, we get four new states AB, AC, BC, BD. Designed DFA accept even number of r and even number of s, out of four states AB satisfied this condition, hence this state is a final state.

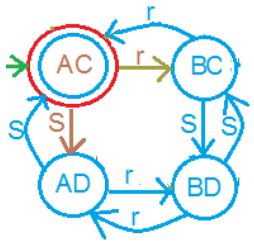


Figure 1.38: *DFA which accept even number of r and s.*

We can define designed DFA as M_{DFA}=(State, ∑, δ, IS, {FS})

State={ AC, AD, BC, BD} ∑={r, s}

δ :
δ (AC, r)= BC δ (AC, s)= AD ; δ (BC, r)=AC δ (BC, s)=BD
δ (AD, r)=BD δ (AD, s)=BD; δ (BD, r)=AD δ (BD, s)=BC

IS=AC; FS={ AC}

Example 1.17: Design a DFA which accept all the strings described by the language L={N_0(w)≥ 1 and N_1(w)=2}

L={011, 110, 0011, 1010, 11100, 10101...}

i. Exchange I symbol of first string with the last symbol of the first string, the string obtained is the second string of the language.
ii. Draw NFA for two smallest strings, for both NFA initial and final states are same.

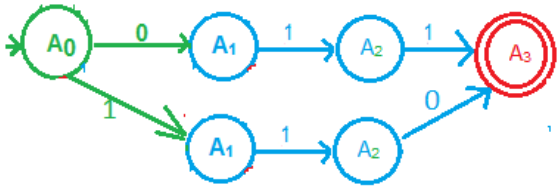


Figure 1.39: Two DFAs for the string 011 and 110 with same initial and final state.

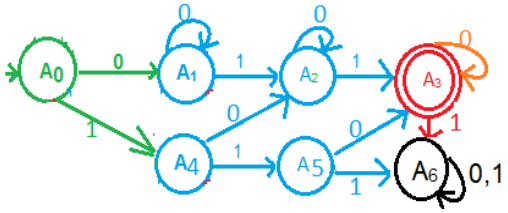


Figure 1.40: DFA which accept all the strings described by the language L={$N_0(w) \geq 1$ and $N_1(w)=2$}.

We can define designed DFA as $M_{DFA}=(A, \sum, \delta, IS, \{FS\})$

A={ $A_0, A_1, A_2 A_3, A_4, A_5, A_6$} ∑={0, 1}

δ :

δ (A_0, 0)=A_1 δ (A_0, 1)=A_4 ; δ (A_1, 0)=A_0 δ (A_1, 1)=A_2
δ (A_2, 0)=A_2 δ (A_2, 1)=A_3 ; δ (A_4, 0)=A_2 δ (A_4, 1)=A_5
δ (A_5, 0)=A_3 δ (A_5, 1)=A_6 ; δ (A_6, 0)=A_6 δ (A_6, 1)=A_6

IS=A_0; FS={A_3}

Nondeterministic Finite Automata (NFA)

In DFA the outgoing transitions are defined in advance, where as in NFA the outgoing transitions are not defined in advance. In DFA, the string is either accepted or not accepted, where as in NFA both are possible. The finite automata shown in the figure 1.41 is a NFA. There is no outgoing transition for the symbol 'a' from the state q_1.

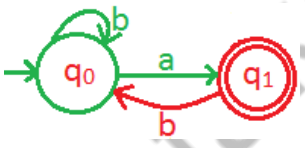


Figure 1.41: Nondeterministic Finite Automata

We can define above NFA as M_{DFA} =(Q, ∑, δ , IS, {FS})
Q ={ q_0, q_1,}
∑ ={a , b}
δ:
$Q^*\sum \rightarrow 2^Q$
δ (q_0, a) =q_1 δ (q_0,b) =q_0
δ (q_1, b) =q_0
δ (q_2, a) =q_2 δ (q_2, b) =q_2
IS =q_0 FS={q_1}

The NFA shown in figure 1.42 is, accepting as well as not accepting the string hod.

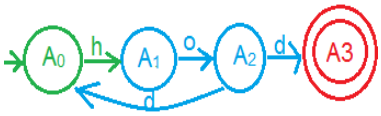


Figure 1.42:Non-deterministic finite automata

Example 1.18: Design NFA which accept all strings, if second symbol from right side is 1 over the alphabet ∑={0, 1}.

L={ 10, 110, 010, 0110…}

i. First step draw the final state.

Figure 1.43: Final state

ii. Draw the intermediate state and transition for the symbols '0' and '1'.

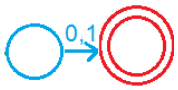


Figure 1.44: Intermediate and final states

iii. Draw the initial state and transition for the symbol '1'.

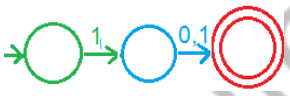


Figure 1.45: Non-deterministic finite automata accepting two strings 10, 11.

iv. Designed NFA has to accept all the strings, if second symbol from right side is symbol '1', to do this, mark loop to initial state on symbols '0' and '1'.

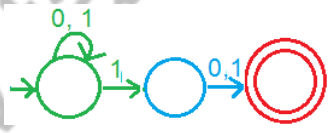


Figure 1.46: NFA which accepts all strings, if second symbol from right side is 1 over the alphabet ∑={0, 1}.

v. Name the states and stop.

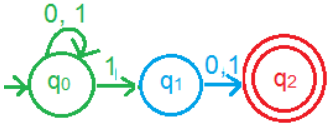


Figure 1.47: NFA which accept all strings, if second symbol from right side is 1 over the alphabet ∑={0, 1}.

Definition of NFA: NFA can be defined as

$M_{DFA} = (Q, \sum, \delta, IS, \{FS\})$
$Q = \{q_0, q_1, q_2\}$
$\sum = \{0, 1\}$
δ:
$\delta(q_0, 0) = q_0 \quad \delta(q_0, 1) = \{q_0, q_1\}$
$\delta(q_1, 0) = q_2$
$\delta(q_1, 1) = q_2$
$IS = q_0 \quad FS = \{q_2\}$

Equivalence of NFA and DFA:

We have already seen both NFA and DFA, both are same in the definition but they differ in transition function. If number of states in NFA has two states q_0 and q_1. Here, q_0 is initial state and q_1 is final state. The equivalent states in DFA's are $[q_0]$, $[q_1]$ and $[q_0, q_1]$. Where q_0 is initial state and (q_1), (q_0, q_1) are the final states. The pair (q_0, q_1) is final state because; one of its states is final state. They are represented as shown in figure 1.48

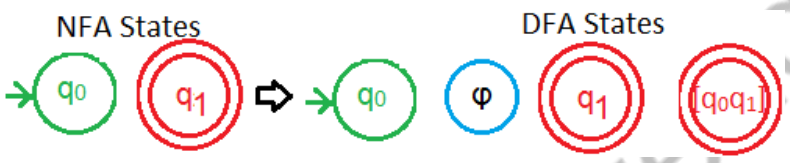


Figure 1.48: Equivalence states of NFA and DFA.

Example 1.19: Convert the NFA shown in figure 1.49 into its equivalent DFA.

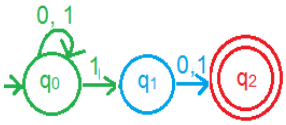


Figure 1.49: Non-deterministic finite automata of Example 1.19.

The conversion begins with the initial state of the NFA which is also an initial state of DFA. Conversion generates number of new states, for every new state we are finding the outgoing transitions for different symbols.

Table 1.4: Different tables for the conversion of NFA to DFA.

Q \ Σ	0	1
→q0	q0	[q0 q1]

New state

Q \ Σ	0	1
→q0	q0	[q0 q1]
[q0q1]	[q0q2]	[q0q1q2]

New state

New states

$\delta([q_0q_1], 0) = \delta(q_0, 0) \cup \delta(q_1, 0)$
$= q_0 \cup q_2$
$= q_0q_2$
$\delta([q_0q_1], 1) = \delta(q_0, 1) \cup \delta(q_1, 1)$
$= q_0 q_1 \cup q_2$
$= q_0q_1q_2$

Q \ Σ	0	1
→q0	q0	[q0 q1]
[q0q1]	q0q2	q0q1q2
q0q2	q0	[q0q1]

$\delta([q_0q_2], 0)=\delta(q_0, 0)\cup\delta(q_2, 0)$
$=q_0 \cup \emptyset$
$=q_0$
$\delta([q_0q_2], 1)=\delta(q_0, 1)\cup\delta(q_2, 1)$
$=q_0q_1 \cup \emptyset$
$=q_0q_1$

Q \ Σ	0	1
→q0	q0	[q0 q1]
[q0q1]	q0q2	q0q1q2
[q0q2]	q0	[q0q1]
[q0q1q2]	[q0q2]	[q0q1q2]

$\delta([q_0q_1q_2],0)=\delta(q_0, 0)\cup\delta(q_1, 0)\cup\delta(q_2, 0)$
$=q_0 \cup q_2 \cup \emptyset$
$=q_0q_2$
$\delta([q_0q_1q_2],1)=\delta(q_0, 1)\cup\delta(q_1, 1)\cup\delta(q_2, 1)$
$=q_0\, q_1 \cup q_2 \cup \emptyset$
$=q_0q_1q_2$

No further new states; mark the final state of DFA. Any pair of state which consists of final state of the NFA, the particular pair is a final state in DFA.

Q \ Σ	0	1
→q0	q0	[q0 q1]
[q0q1]	q0q2	q0q1q2
*[q0q2]	q0	[q0q1]
*[q0q1q2]	[q0q2]	[q0q1q2]

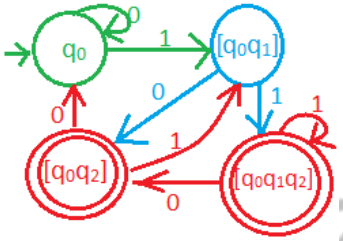


Figure 1.50: Equivalence DFA of figure 1.49

Application of Finite automata:

Finite automat is used to search a text. There are two methods used to search text, they are:

i. Nondeterministic Finite Automata(NFA)
ii. Deterministic Finite Automata(DFA)

To search a text "shine" and "hire" the NFA shown in figure 1.51 is used

The input alphabets are ∑={s, h, i, n, e, r }

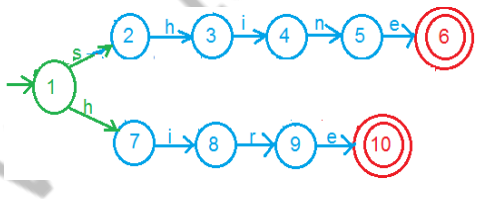


Figure 1.51: NFA used to search strings shine and hire

To search a text "shine" and "hire" the DFA shown in figure 1.52 is used

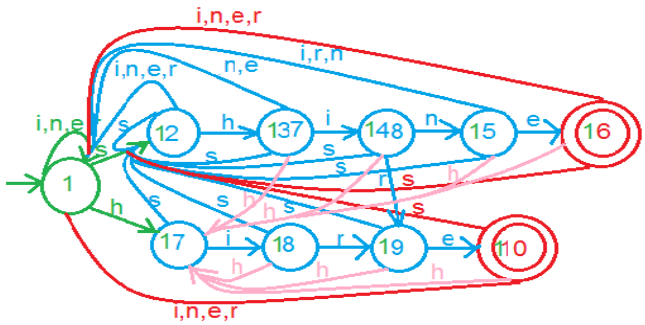


Figure 1.52: DFA used to search strings ending with shine and hire

Example 1.20: Design a finite automata, that reads string made up of letters HONDA, and recognize those string that contain the word HOD as a substring.

i. Find out the alphabet from the string HONDA. The input symbols are ∑={H, O, N, D, A}
ii. Draw the NFA for the substring HOD.

Figure 1.53: NFA used to search string HOD

iii. Convert NFA state of figure 1.53 into DFA states.

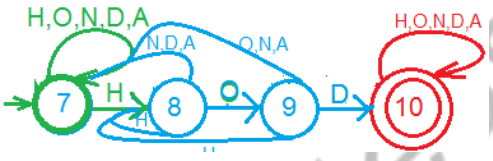


Figure 1.54: DFA accept all the strings ends with HOD over the symbols {H, O, N, D, A}

NFA with ε-transition:

Till now whatever automata we studied were characterized by a transition on the symbols. But there are some finite automata having transition for empty string also, such NFAs are called NFA- ε transition.

ε-closure (q₀): states which are closed / related to state q_0 on empty transition(ε) is called ε-closure (q_0).

In DFA and NFA $\hat{\delta}(q_0, \varepsilon)=q_0$, where as in ε-NFA

i. $\hat{\delta}(q, \varepsilon)$= ε-closure($\hat{\delta}(q, \varepsilon)$)= ε-closure(q).
ii. $\hat{\delta}(q, x)$= ε-closure(δ($\hat{\delta}(q, \varepsilon)$, x)= ε-closure($\delta$(ε-closure(q), x). $\hat{\delta} \neq \delta$
iii. $\hat{\delta}(q, wx)$= ε-closure(δ($\hat{\delta}(q, w)$, x).

Example 1.21: Find out ε-closure of all the states of ε-NFA shown in figure 1.55.

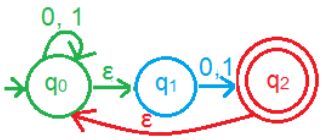


Figure 1.55: NFA with ε-transition of Example 1.21.

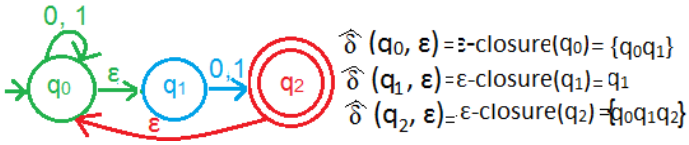


Figure 1.56: ε-closure of all the states of figure 1.55

Example 1.22: Convert the ε-NFA shown in figure 1.57 into NFA

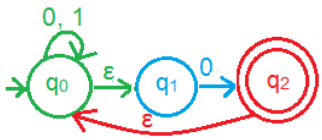


Figure 1.57: ε-NFA of Example 1.22.

First find out the ε-closure of all the states.

Note: state ε-closure consist of final state of ε-NFA, the particular state is a final state in NFA.

$\hat{\delta}(q_0, \varepsilon)$ =ε-closure(q_0)= {q_0q_1}
$\hat{\delta}(q_1, \varepsilon)$ =ε-closure(q_1)= q_1
$\hat{\delta}(q_2, \varepsilon)$ =ε-closure(q_2) ={$q_0q_1q_2$}

$\hat{\delta}(q_0, 0)$= ε-closure(δ($\hat{\delta}(q_0, \varepsilon)$, 0).

= ε-closure(δ({q_0q_1}, 0).

= ε-closure(δ(q_0, 0) U δ(q_1, 0)).

=ε-closure(q_0 U q_2).

=ε-closure(q_0) U ε-closure(q_2).

={q_0q_1} U {$q_0q_1q_2$}

={$q_0q_1q_2$}

$\hat{\delta}(q_0, 1)$= ε-closure(δ($\hat{\delta}(q_0, \varepsilon)$, 1).

= ε-closure(δ({q_0q_1}, 1).

= ε-closure(δ(q_0, 1) U δ(q_1, 1)).

=ε-closure(q_0 U ϕ).

=ε-closure(q_0) U ε-closure(ϕ).

={q_0q_1}

$\hat{\delta}$(q_1, 0)= ε-closure(δ($\hat{\delta}$(q_1, ε), 0).

= ε-closure(δ(q_1, 0)).

= ε-closure(q_2).

={$q_0q_1q_2$}

$\hat{\delta}$(q_1, 1)= ε-closure(δ($\hat{\delta}$(q_1, ε), 1).

= ε-closure(δ(q_1, 1)).

= ε-closure(ϕ).

=ϕ

$\hat{\delta}$(q_2, 0)= ε-closure(δ($\hat{\delta}$(q_2, ε), 0).

= ε-closure(δ({$q_0q_1q_2$}, 0).

= ε-closure(δ(q_0, 0) U δ(q_1, 0)U δ(q_2, 0)).

=ε-closure(q_0 U q_2 U ϕ).

=ε-closure(q_0) U ε-closure(q_2) U ε-closure(ϕ).

={q_0q_1} U {$q_0q_1q_2$} U (ϕ)

={$q_0q_1q_2$}

$\hat{\delta}$(q_2, 1)= ε-closure(δ($\hat{\delta}$(q_2, ε), 1).

= ε-closure(δ({$q_0q_1q_2$}, 1).

= ε-closure(δ(q_0, 1) U δ(q_1, 1) U δ(q_2, 1)).

=ε-closure(q_0 U ϕ U ϕ).

=ε-closure(q_0) U ε-closure(ϕ) U ε-closure(ϕ)

=(q_0 q_1) U (ϕ) U (ϕ)

= {q_0q_1}

Table 1.5: Equivalence table of ε-NFA and NFA.

Q \ Σ	0	1
→q0	{q0q1q2}	{q0q1 }
q1	{q0q1q2}	φ
(q2)	{q0q1q2}	{q0q1}

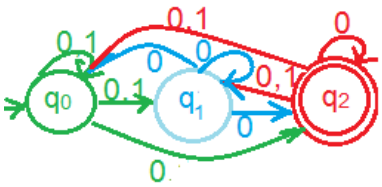


Figure 1.58: NFA without ε-transition.

Conversion of ε-NFA to DFA:

It is same as conversion of NFA to DFA, begins with initial state of ε-NFA.

Example 1.23: Convert the ε-NFA shown in figure1.59 into DFA.

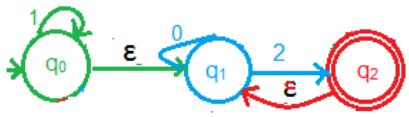


Figure 1.59: ε-NFA of Example 1.23.

First find out the ε-closure of all the states

$\hat{\delta}(q_0, \varepsilon)$ = ε-closure(q_0) = {q_0q_1}
$\hat{\delta}(q_1, \varepsilon)$ = ε-closure(q_1) = q_1
$\hat{\delta}(q_2, \varepsilon)$ = ε-closure(q_2) = {q_1q_2}

Table 1.6: Equivalence table of ε-NFA and NFA.

Q \ Σ	0	1	2
→q0	q1	[q0q1]	[q1q2]
q1	q1	φ	[q1q2]
[q0 q1]	q1	[q0 q1]	[q1 q2]
[q1q2]	q1	φ	[q1 q2]

$\hat{\delta}(q_0, 0)$ = ε-closure(δ($\hat{\delta}(q_0, \varepsilon)$, 0).

= ε-closure(δ({q_0q_1}, 0).

= ε-closure(δ(q_0, 0) U δ(q_1, 0)).

=ε-closure(ϕ U q_1).

=ε-closure(ϕ) U ε-closure(q_1).

= ϕ U q_1

=q_1

$\hat{\delta}$(q_0, 1)= ε-closure(δ($\hat{\delta}$(q_0, ε), 1).

= ε-closure(δ({q_0q_1}, 1).

= ε-closure(δ(q_0, 1)U δ(q_1, 1)).

=ε-closure(q_0U ϕ).

=ε-closure(q_0) U ε-closure(ϕ).

={q_0q_1}U ϕ

=[q_0q_1]

$\hat{\delta}$(q_0, 2)= ε-closure(δ($\hat{\delta}$(q_0, ε), 2).

= ε-closure(δ({q_0q_1}, 2).

= ε-closure(δ(q_0, 2)U δ(q_1, 2)).

=ε-closure(ϕ U q_2).

=ε-closure(q_2).

=[q_1q_2]

$\hat{\delta}$(q_1, 0)= ε-closure(δ($\hat{\delta}$(q_1, ε), 0).

= ε-closure(δ(q_1, 0).

= ε-closure(q_1).

= q_1

$\hat{\delta}$(q_1, 1)= ε-closure(δ($\hat{\delta}$(q_1, ε), 1).

= ε-closure(δ(q_1, 1).

= ε-closure(ϕ).

= ϕ

$\hat{\delta}(q_1, 2)$= ε-closure(δ($\hat{\delta}(q_1, \varepsilon)$, 2).

= ε-closure(δ(q_1, 2)

= ε-closure(q_2)

= [q_1 q_2]

$\hat{\delta}([q_0q_1], 0)$= ε-closure(δ($\hat{\delta}([q_0q_1], \varepsilon)$, 0)

= ε-closure(δ($\hat{\delta}(q_0, \varepsilon)$U $\hat{\delta}(q_1, \varepsilon)$), 0).

= ε-closure(δ((q_0, q_1) U q_1), 0)).

= ε-closure(δ((q_0, q_1), 0)).

= ε-closure(δ(q_0, 0) U δ(q_1, 0)).

= ε-closure(ϕ U q_1)

= ε-closure(q_1)

= q_1

$\hat{\delta}([q_0q_1], 1)$= ε-closure(δ($\hat{\delta}([q_0q_1], \varepsilon)$, 1)

= ε-closure(δ($\hat{\delta}(q_0, \varepsilon)$U $\hat{\delta}(q_1, \varepsilon)$), 1).

= ε-closure(δ((q_0, q_1) U q_1), 1)).

= ε-closure(δ((q_0, q_1), 1)).

= ε-closure(δ(q_0, 1) U δ(q_1, 1)).

= ε-closure(q_0 U ϕ)

= ε-closure(q_0)

= [q_0, q_1]

$\hat{\delta}([q_0q_1], 2)$= ε-closure(δ($\hat{\delta}([q_0q_1], \varepsilon)$, 2)

= ε-closure(δ($\hat{\delta}(q_0, \varepsilon)$U $\hat{\delta}(q_1, \varepsilon)$), 2).

= ε-closure(δ((q_0, q_1) U q_1), 2)).

= ε-closure(δ((q_0, q_1), 2)).

= ε-closure(δ(q_0, 2) U δ(q_1, 2)).

= ε-closure(ϕ U q_2)

= ε-closure(q_2)

= [q_1 q_2]

$\hat{\delta}$([q_1q_2], 0)= ε-closure(δ($\hat{\delta}$([q_1q_2], ε), 0)

= ε-closure(δ($\hat{\delta}$(q_1, ε)U $\hat{\delta}$(q_2, ε), 0)).

= ε-closure(δ(q_1U (q_1 q_2)), 0).

= ε-closure(δ(q_1, q_2), 0)).

= ε-closure(δ(q_1, 0) U δ (q_2, 0))

= ε-closure(q_1 U ϕ).

= ε-closure(q_1)

= q_1

$\hat{\delta}$([q_1q_2], 1)= ε-closure(δ($\hat{\delta}$([q_1q_2], ε), 1)

= ε-closure(δ($\hat{\delta}$(q_1, ε)U $\hat{\delta}$(q_2, ε), 1)).

= ε-closure(δ(q_1U (q_1 q_2)), 1).

= ε-closure(δ(q_1, q_2), 1)).

= ε-closure(δ(q_1, 1)U δ (q_2, 1))

= ε-closure(ϕ U ϕ).

= ε-closure(ϕ)

= ϕ

$\hat{\delta}$([q_1q_2], 2)= ε-closure(δ($\hat{\delta}$([q_1q_2], ε), 2)

= ε-closure(δ($\hat{\delta}$(q_1, ε)U $\hat{\delta}$(q_2, ε), 2)).

= ε-closure(δ(q_1U (q_1 q_2)), 2).

= ε-closure(δ(q_1, q_2), 2)).

= ε-closure(δ(q_1, 2)U δ (q_2, 2))

= ε-closure(q_2 U ϕ).

= ε-closure(q_2)

$= [q_1\ q_2]$

Table 1.7: Equivalence table of ε-NFA and DFA.

Q / Σ	0	1	2
q_0	q_1	$[q_0q_1]$	$[q_1q_2]$
q_1	q_1	φ	$[q_1q_2]$
$[q_0\ q_1]$	q_1	$[q_0\ q_1]$	$[q_1\ q_2]$
$[q_1q_2]$	q_1	φ	$[q_1\ q_2]$
φ	φ	φ	φ

$\delta(\phi, (0,1,2)) = \phi$

Mark the final states, any pair which has a final state of ε -NFA the particular pair is afinal state in DFA.

Table 1.7: Equivalence table of ε-NFA and DFA .

Q / Σ	0	1	2
q_0	q_1	$[q_0q_1]$	$[q_1q_2]$
q_1	q_1	φ	$[q_1q_2]$
$[q_0\ q_1]$	q_1	$[q_0\ q_1]$	$[q_1\ q_2]$
*$[q_1q_2]$	q_1	φ	$[q_1\ q_2]$
φ	φ	φ	φ

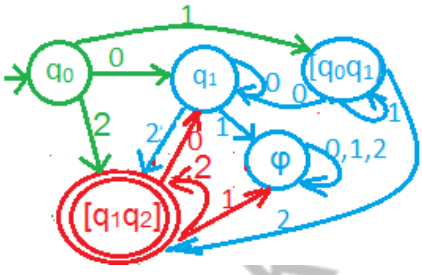


Figure 1.60: DFA of figure 1.59.

Chapter 2

REGULAR EXPRESSIONS

Language represented in the form of expression (symbols separated by operator) is called regular expression. δ_{1U2}

$\sum$= {a, b, 0, 1}

Language L= $\sum^*$= {a, b, 0, 1}* ={ Ɛ, a, b, 0, 1, aa, bb, 00, 11…}

Regular expression R= { Ɛ + a + 0 + 1 + aa + bb + 00 + 11…}

Before studying regular expression in detail, we will discuss some fundamental rules of the regular expression.

i. Φ+1=1 (φ is like, number 0).
ii. Ɛ + 1= Ɛ+1 (Ɛ is like number 1)
iii. φƐ=φ
iv. Ɛ+φ=Ɛ
v. 1+1=1
vi. 0+0=0
vii. 0+1=0+1
viii. 0+01=0(Ɛ+1)
ix. 0Ɛ=0
x. 0φ=0
xi. Φ*={Ɛ+ φ+ φφ…}={Ɛ+φ}=Ɛ
xii. 0^++Ɛ=0*
xiii. 0*+Ɛ=0*
xiv. $(1+Ɛ)^+$=1*
xv. $(1+φ)^+=1^+$
xvi. $00^*=0^+$
xvii. $Ɛ+11^*=Ɛ+1^+=1^*$
xviii. $(Ɛ+11)(Ɛ+11)^*=(Ɛ+11)^+=(11)^*$
xix. (00)*(1+01)=(00)*(Ɛ+0)1= (Ɛ+00+0000..)(Ɛ+0)1=0*1
xx. Write a regular expression set of strings made up of 0's and 1's ending with 00
RE=(0+1)*00.
xxi. Write a regular expression for all the set of string made up of 0's and 1's which begin with 0 and end with 1
RE=0(1+0)*1
xxii. Strings begins with 0 or 1 is represented as $(0+1)(0+1)^*$
xxiii. All strings begins with 0 or 1 and not having two consecutive 0's
RE=(0+1)(1+10)
xxiv. Write regular expression which consist of two consecutive 0's
RE=(0+1)*00(0+1)*
xxv. All the strings of 0's and 1's whose last two symbols are same
RE=(0+1)*(00+11)
xxvi. Sets of all strings which starts with either 00 or 11 or ends with 00 or 11

RE=(00+11) (0+1)* +(0+1)*(00+11).

xxvii. Find the regular expression of the language L={$a^n b^m$:n≥0, m≥0}
L={Ɛ+a+aa..}{Ɛ+b+bb…}
RE=a*b*

xxviii. Find the regular expression of the language L={$a^n b^{2m+2}$: n≥0, m≥0}
RE=(aa)*(bb)*bb.

xxix. Find the regular expression of the language L={$a^{2n} b^m$: n≥4, m≤3}
RE=(aaaa)*(Ɛ+b+bb+bbb)

xxx. Find the regular expression of the language L={$a^n b^m$/n+m is even}
RE=(aa)*(bb)*+a(aa)*b(bb)*

Conversion from Regular expression to finite automata:

Three basic regular expressions are used in conversion (0+1), 01 and 0*.

The finite automata which accept either 0 or 1 is shown in figure 2.1.

Figure 2.1: Finite automata

But in regular expression each symbol is having its own finite automata.

Definition for the finite automat which accept the symbol ‘1’ is

$M_1 = (Q_1, \sum_1, \delta, IS_1, FS_1)$

$Q_1=\{C, D\}$; $\sum_1=1$; $\delta(C, 1) = D$; IS=C; FS=D

Definition for the finite automat which accept the symbol ‘0’ or ‘1’ is

$M = (Q, \sum, \delta, IS, FS)$

$Q=\{Q0 \cup Q1 \cup E \cup F\} =\{ A, B, C, D, E. F)$;

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

δ

$\delta(A, 0)= B$; $\delta (C, 1)= D$; $\delta(E, Ɛ)=\{A, C\}$; $\delta(D, Ɛ)=F$: $\delta(D, Ɛ)=F$

IS=E; FS=F

For the symbol ‘0’ the finite automata is shown in figure 2.2

Figure 2.2: Finite automata for regular expression 0.

Definition for the finite automat which accept the symbol '0' is

$M_0 = (Q_0, \sum_0, \delta, IS_0, FS_0)$

$Q_0 = \{A, B\}$; $\sum_0 = 0$; $\delta(A, 0) = B$; IS=A; FS=B

For the symbol '1' is shown in figure 2.3

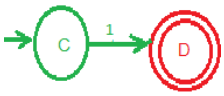


Figure 2.3: Finite automata for regular expression 1.

Definition for the finite automat which accept the symbol '1' is

$M_1 = (Q_1, \sum_1, \delta, IS_1, FS_1)$

$Q_1 = \{C, D\}$; $\sum_1 = 1$; $\delta(C, 1) = D$; IS=C; FS=D

We are joining these two finite automata with disturbing the regular expression 0+1 as shown in figure 2.4. To join these two finite automata we need more extra states, one for the initial state and second is for the final state, states A and B no more initial states and B and D are no more final states. All these states will now become intermediate states as shown in figure 2.4.

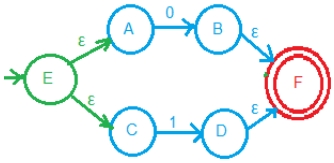


Figure 2.4: Finite automata for regular expression 0+1.

Definition for the finite automat which accept the symbol '0' or '1' is

$M = (Q, \sum, \delta, IS, FS)$

$Q = \{Q_0 \cup Q_1 \cup E \cup F\} = \{A, B, C, D, E. F)$;

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

δ

$\delta(A, 0) = B$; $\delta(C, 1) = D$; $\delta(E, Ɛ) = \{A, C\}$; $\delta(D, Ɛ) = F$: $\delta(D, Ɛ) = F$

IS=E; FS=F

For the regular expression 01 the finite automata is shown in figure 2.6.

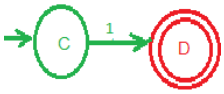


Figure 2.5: Finite automata for regular expression 0 and 1.

Rewrite the expression 01 as 0Ɛ1.

We can join the two finite automata shown in figure2.5 and figure 2.6 without disturbing the regular expression 01 is shown in figure 2.6.

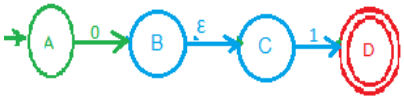


Figure 2.6: Finite automata for regular expression 01.

Definition for the finite automat which accept the string 01 is

M= (Q,∑, δ, IS, FS)

Q={Q_0 U Q_1} ={ A, B, C, D);

∑={0, 1}

δ

δ(A, 0)= B; δ (C, 1)= D; δ(B, Ɛ)= C

IS=A; FS=D

Like that we can design the finite automata for 0*.

0*= {Ɛ, 0, 00, 000…}

The regular expression for symbol ‘0’ is shown in figure 2.7

Figure 2.7: Finite automata for regular expression 0.

The designed finite automata accept all the strings of 0* ie., {Ɛ, 0, 00, 000…}

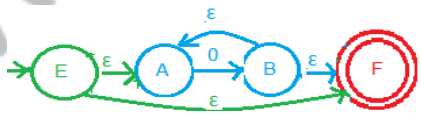


Figure 2.8: Finite automata for regular expression 0*.

Definition for the finite automat which accept the string 0* is

M= (Q,∑, δ, IS, FS)

Q={Q_0 U E U F} ={ A, B, E, F);

∑=0

δ

δ(A, 0)= B; δ (E, Ɛ)= {A, F}; δ(B, Ɛ)= {F, A}

IS=E; FS=F

Example 2.1: Design a finite automat for the regular expression 0^*1

i. Draw the finite automata for the symbol '0'

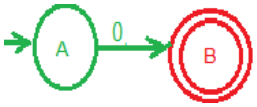


Figure 2.9: Finite automata for regular expression 0.

ii. Convert the finite automata of symbol '0' to 0*

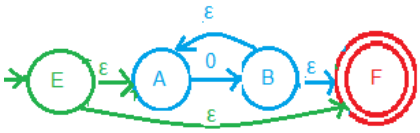


Figure 2.10: Finite automata for regular expression 0*.

iii. Draw the finite automata for the symbol '1'

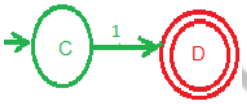


Figure 2.11: Finite automata for regular expression 1.

Join finite automata of 0* and 1, which will be the solution for the regular expression 0*1

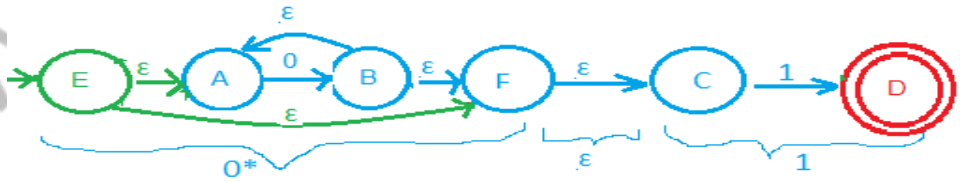


Figure 2.12: Finite automata for regular expression 0*1.

Example 2.2: Design a finite automat for the regular expression $0(0+1)^*1$

i. Draw the finite automata for the symbol '0' and '1'

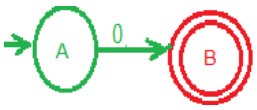


Figure 2.13: Finite automata for regular expression 0.

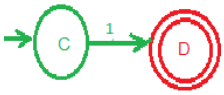


Figure 2.13: Finite automata for regular expression 1.

ii. Construct the finite automata for the expression 0 + 1

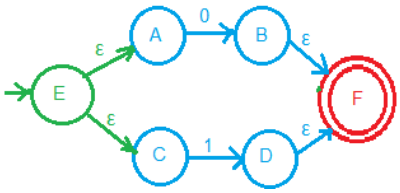


Figure 2.14: Finite automata for regular expression 0+1.

iii. Construct the finite automata for the symbol (0 + 1)*

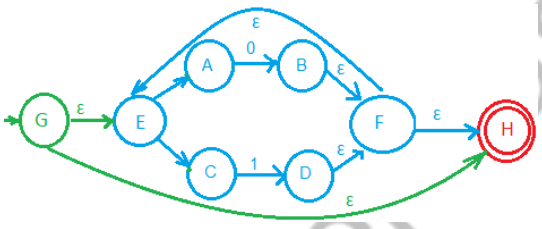


Figure 2.15: Finite automata for regular expression (0+1)*.

iv. Join the finite automata of '0', (0+1)* and '1'

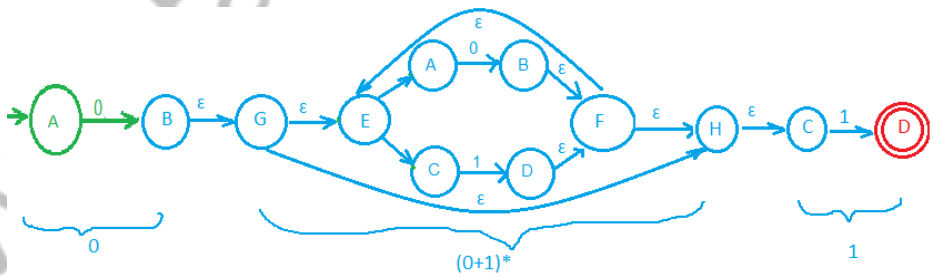


Figure 2.16: Finite automata for regular expression 0(0+1)*1.

Conversion from Finite Automata to Regular Expression:

They are of two methods of conversions, they are:

i. Recursive Method.
ii. State Elimination Method.

i. Recursive Method :

In this method it uses recursive method, to construct new regular expression.
For example the regular expression of DFA shown in figure 2.17 is

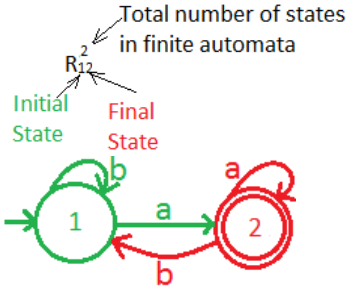


Figure 2.17: A typical Finite automata.

In order to obtain the value of regular expression R_{12}^2 , we use all the (transition) possible regular expression of the DFA. The possible regular expression of the figure 2.17 is shown in the table2.1.

Table 2.1: Recursive table

	k=0	k=1
R_{11}^k		
R_{12}^k		
R_{21}^k		
R_{22}^k		

Equations to obtain the regular expression of all the transitions are as follows:

$$\text{if } k=0 \quad R_{ij}^k = \begin{cases} a \text{ if } i \neq j \\ a \cup \varepsilon \text{ if } i=j \end{cases}$$

where a is the transition between state i and j

$$\text{if } k>0 \quad R_{ij}^k = R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1} + R_{ij}^{k-1}$$

If state 2 and 3 are final states then the regular expression is written as:

$$RE = R_{12}^3 + R_{13}^3$$

Example 2.3: Find the regular expression of DFA shown in figure 2.17 by using recursive method.

	k=0	k=1
R_{11}^k	b+ ε	b*
R_{12}^k	a	b*a
R_{21}^k	b	b*b
R_{22}^k	a+ ε	b*a+ ε

$R_{11}^1 = R_{11}^0(R_{11}^0)^*R_{11}^0 + R_{11}^0$

$R_{11}^1 = (b+\varepsilon)(b+\varepsilon)^*(b+\varepsilon) + (b+\varepsilon)$

$= [(b+\varepsilon)^+ + \varepsilon](b+\varepsilon)$

$= [(b+\varepsilon)^*(b+\varepsilon)$

$= [(b+\varepsilon)^+$

$= b^*$

$R_{12}^1 = R_{11}^0(R_{11}^0)^*R_{12}^0 + R_{12}^0$

$= (b+\varepsilon)(b+\varepsilon)^*a + a$

$= [(b+\varepsilon)(b+\varepsilon)^* + \varepsilon]a$

$= [(b+\varepsilon)^+ + \varepsilon]a$

$= b^*a$

$R_{21}^1 = R_{21}^0(R_{11}^0)^*R_{11}^0 + R_{21}^0$

$= b(b+\varepsilon)^*(b+\varepsilon) + b$

$= [(b+\varepsilon)^*(b+\varepsilon) + \varepsilon]b$

$= [(b+\varepsilon)^+ + \varepsilon]b$

$= (b+\varepsilon)^+ b$

$= b^*b$

$R_{22}^1 = R_{21}^0(R_{11}^0)^*R_{12}^0 + R_{22}^0$

$= b\,b^*a + a + \varepsilon \quad = b^+a + a + \varepsilon$

$= (b^+ + \varepsilon)a + \varepsilon$

$= b^*a + \varepsilon$

$R_{12}^2 = R_{12}^1(R_{22}^1)^*R_{22}^1 + R_{12}^1$

RE $= b^*a(b^*a+\varepsilon)^*b^*a + \varepsilon + b^*a$

Example 2.4: Find the regular expression of DFA shown in figure 2.18 by using recursive method.

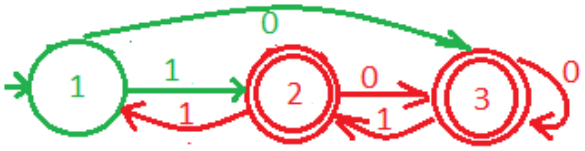


Figure 2.18: DFA

	k=0	k-1	k=2
R_{11}^k	φ+ε =ε	ε	(11)*
R_{12}^k	1	1	1(11)*
R_{13}^k	0	0	1*0
R_{21}^k	1	1	(11)*1
R_{22}^k	ε	11+ε	(11)*
R_{23}^k	0	10+0	(11)*0(1+ε)
R_{31}^k	φ	φ	11(11)*
R_{32}^k	1	1	1(11)*
R_{33}^k	0+ε	0+ε	101*+0+ε

$R_{11}^1 = R_{11}^0(R_{11}^0)^*R_{11}^0 + R_{11}^0$

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

$R_{12}^1 = R_{11}^0(R_{11}^0)^*R_{12}^0 + R_{12}^0$

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

$R_{12}^1 = R_{11}^0(R_{11}^0)^*R_{12}^0 + R_{12}^0$

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

$R_{13}^1 = R_{11}^0(R_{11}^0)^*R_{13}^0 + R_{13}^0$

$= \varepsilon(\varepsilon)^*0 + 0$

$= 0 + 0 = 0$

$R_{21}^1 = R_{21}^0(R_{11}^0)^*R_{11}^0 + R_{21}^0$

$= 1(\varepsilon)\varepsilon + 1 = 1$

$R_{22}^1 = R_{21}^0(R_{11}^0)^*R_{12}^0 + R_{22}^0$

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

$= 11 + \varepsilon$

$R_{23}^1 = R_{21}^0(R_{11}^0)R_{13}^0 + R_{23}^0$

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

$= 10 + 0$

$R_{31}^{1}=R_{31}^{0}(R_{11}^{0})^{*}R_{11}^{0}+R_{31}^{0}$

$=\varphi+\varphi=\varphi$

$R_{32}^{1}=R_{31}^{0}(R_{11}^{0})^{*}R_{12}^{0}+R_{32}^{0}$

$=\varphi+1=1$

$R_{33}^{1}=R_{31}^{0}(R_{11}^{0})^{*}R_{13}^{0}+R_{33}^{0}$

$=\varphi+0+\varepsilon$

$R_{11}^{2}=R_{12}^{1}(R_{22}^{1})^{*}R_{21}^{1}+R_{11}^{1}$

$1(11+\varepsilon)^{*}1+\varepsilon$

$=11(11)^{*}+\varepsilon$

$=(11)^{+}+\varepsilon$

$=(11)^{*}$

$R_{12}^{2}=R_{12}^{1}(R_{22}^{1})^{*}R_{22}^{1}+R_{12}^{1}$

$=1(11+\varepsilon)^{*}(11+\varepsilon)+1$

$=1(11+\varepsilon)^{+}+1$

$=1[(11+\varepsilon)^{+}+\varepsilon]$

$=1(11)^{*}$

$R_{13}^{2}=R_{12}^{1}(R_{22}^{1})^{*}R_{23}^{1}+R_{13}^{1}$

$=1(11+\varepsilon)^{*}(10+0)+0$

$=1(11)^{*}(1+\varepsilon)0+0$

$=[11(11)^{*}+1(11)^{*}]0+0$

$=[(11)^{+}+1(11)^{*}+\varepsilon]0$

$=[(11)^{*}+1(11)^{*}]0$

$=(11)^{*}(\varepsilon+1)0$ $=1^{*}$

$=1^{*}0$

$R_{21}^{2}=R_{22}^{1}(R_{22}^{1})^{*}R_{21}^{1}+R_{21}^{1}$

$=(11+\varepsilon)(11+\varepsilon)^{*}1+1$

$=(11+\varepsilon)^{+}1+1$

$=((11+\varepsilon)^{+}+\varepsilon)1$

$=(11)^{*}1$

$R_{22}^{2}=R_{22}^{1}(R_{22}^{1})^{*}R_{22}^{1}+R_{22}^{1}$

$=(11+\varepsilon)(11+\varepsilon)^{*}(11+\varepsilon)+(11+\varepsilon)$

$=((11+\varepsilon)(11+\varepsilon)^{*}+\varepsilon)\ (11+\varepsilon)$

$=((11+\varepsilon)^{+}+\varepsilon)\ (11+\varepsilon)$

$=((11+\varepsilon)^{*}(11+\varepsilon))$

$=((11+\varepsilon))^{+}$

$=(11)^{*}$

$R_{23}^{2}=R_{22}^{1}(R_{22}^{1})^{*}R_{23}^{1}+R_{23}^{1}$

$=(11+\varepsilon)((11+\varepsilon)^{*}(10+0)+(10+0)$

$=[(11+\varepsilon)((11+\varepsilon)^{*}+\varepsilon](10+0)$

$=[(11+\varepsilon)^{+}+\varepsilon]\ (10+0)$

$=(11+\varepsilon)^{*}(10+0)$

$=(11)^{*}0(1+\varepsilon)$

$R_{31}^{2}=R_{32}^{1}(R_{22}^{1})^{*}R_{21}^{1}+R_{31}^{1}$

$=1(11+\varepsilon)^{*}1+\varphi$

$=11(11)^{*}$

$R_{32}^{2}=R_{32}^{1}(R_{22}^{1})^{*}R_{22}^{1}+R_{32}^{1}$

$=1(11+\varepsilon)^{*}(11+\varepsilon)+1$

$=1((11+\varepsilon)^{+}1$

$=1[(11+\varepsilon)^{+}+(\varepsilon)]$

$=1(11+\varepsilon)^{*}$

$=1(11)^{*}$

$R_{33}^{2}=R_{32}^{1}(R_{22}^{1})^{*}R_{23}^{1}+R_{33}^{1}$

$=1(11+\varepsilon)^{*}(10+0)+(0+\varepsilon)$

$=10(11+\varepsilon)^{*}(1+\varepsilon)+(0+\varepsilon)$

$=10(11)^{*}(1+\varepsilon)+(0+\varepsilon)$

$=101^{*}+0+\varepsilon$

$RE=R_{12}^{3}+R_{13}^{3}$

$=[1^{*}00(11^{*})^{*}]1(11)^{*}+1(11)^{*}\ (101^{*}+0+\varepsilon)^{*}(101^{*}+0+\varepsilon)+1^{*}0$

State Elimination Method:

In state elimination method, we obtain regular expression, by eliminating all the intermediate states. After eliminating the intermediate states, it is easy to find the regular expression for the remaining states (initial and final).

Figure2.19: Finite automata

The regular expression of the finite automata shown in figure2.20 is:

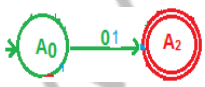


Figure2.20: Finite automata

RE=01

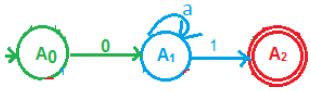


Figure2.21: Finite automata

The regular expression of the finite automata shown in figure 2.2 is:

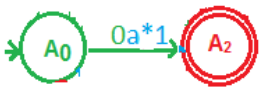


Figure2.22: Finite automata

RE=0a*1

From the above examples we come to know that eliminating state has one predecessor state A_0 and one successor state A_2. In complex finite automata the eliminating state may have more number of predecessors and successors. One Such example is shown in the figure 2.23.

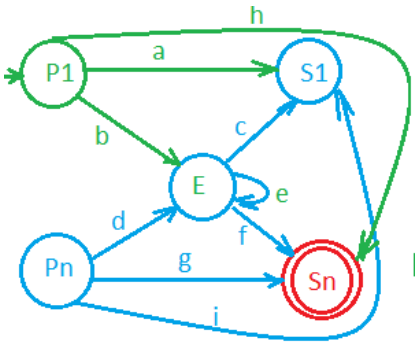


Figure 2.23: Eliminating state E has n predecessor and n successor.

Before eliminating any state from the finite automata, we need to find out the path passing through the eliminating state. The paths passing through the eliminating state are P1ESn, P1ES1, PnES1 and PnESn. After eliminating state E, the automata is shown in the figure 2.24.

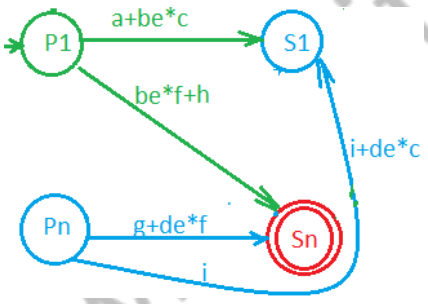


Figure 2.24: Finite automata after eliminating state E.

After eliminating all predecessors and successors, leaving initial state and final state, the remaining machine looks shown in the figure 2.25.

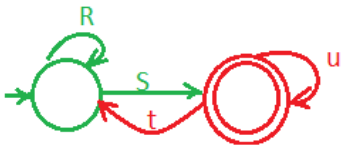


Figure 2.25: Finite automata after eliminating all predecessors and successors.

The regular expression is, language accepted by the initial state of finite automata and Su*.

The language accepted by the initial state is {R*+Su*t+R*Su*t + su*tR*…}, this is like {x*+y*+xy + yx …}

We can simplify it as (x+y)* similarly, we can simplify the language for the initial state as (R+Su*t)*. Therefore, regular expression of finite automata is (R+Su*t)*Su*. Note that in some cases the same state is initial as well as final state.

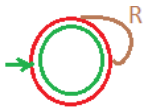


Figure 2.26: Same state is initial as well as final state.

The regular expression is R*.

To eliminate state, three rules are used, they are:

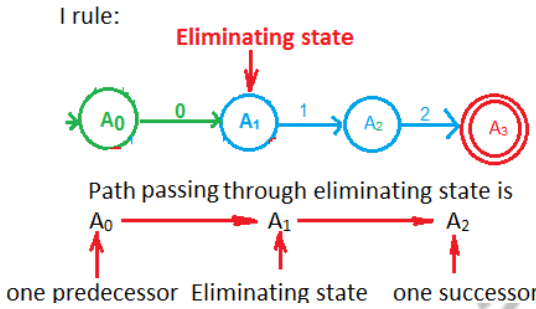


Figure 2.27: One path passing through eliminating state A_1

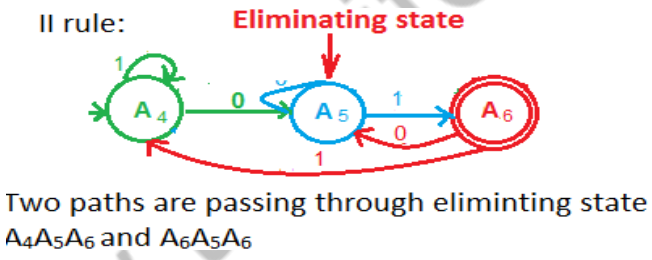


Figure 2.28: Two paths are passing through eliminating state A_1

Predecessor state	Eliminating State	Successor state
A_4	A_5	A_6
A_6	A_5	A_6

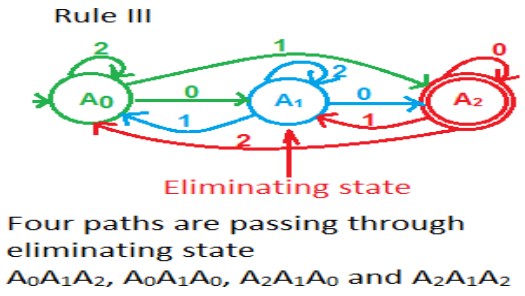


Figure 2.29: Four paths are passing through eliminating state A_1

Predecessor state	Eliminating State	Successor state
A_0	A_1	A_2
A_0	A_1	A_0
A_2	A_1	A_0
A_2	A_1	A_2

Example 2.5: Construct regular expression for the finite automat shown in figure 2.30.

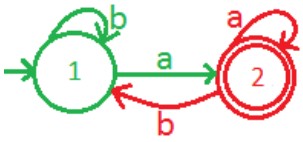


Figure 2.30: Finite Automata

No intermediate in the given finite automata, hence we can't eliminate any of the state.

R=b; S=a; u=a; t=b;

Therefore RE =(R+Su*t)*Su*

= (b+aa*b)*aa*

=b(Ɛ+aa*)*aa*

=b(aa*)*aa*

Example 2.6: Construct regular expression for the finite automat shown in figure 2.31 by using state elimination method.

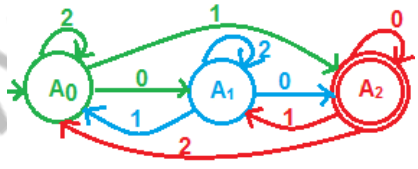


Figure 2.31: Finite Automata.

We can eliminate one intermediate state A_1, the paths passing through the eliminating state are $A_0A_1A_0$, $A_0A_1A_2$, $A_2A_1A_0$ and $A_2A_1A_2$.

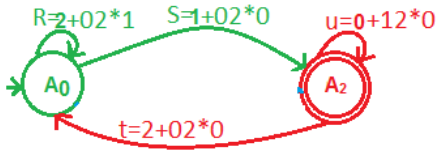


Figure 2.32: Final finite Automata

RE=(R+Su*t)*Su*

= ((2+02*1)+(1+02*0)(0+12*0)*(2+02*0))*((1+02*0)(0+12*0)*.

Application of Regular expression:

Pumping lemma:

It is used to prove language as non-regular for example

L=(00, 000, 0000…}

The automat is shown in figure 2.33 which accept 00.

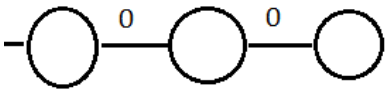


Figure 2.33: Finite Automata

The automat is shown in figure 2.34 which accept 000.

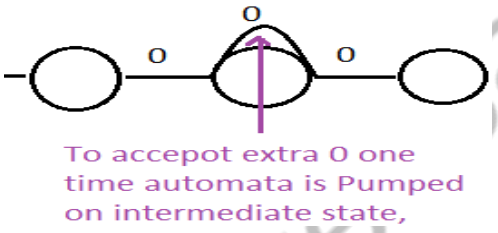


Figure 2.34: Finite Automata pumped at intermediate state one time.

The automat is shown in figure 2.35 accept 0000.

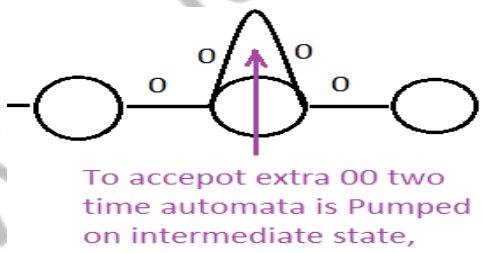


Figure 2.35: Finite Automata pumped at intermediate state one time.

If more 0s, there will be more number of pumping. In such a case it is very difficult to count the number of pumping. To count the number of pump easily we draw circles around the intermediate state so that we can easily count the number of pumping to prove the language as non-regular.

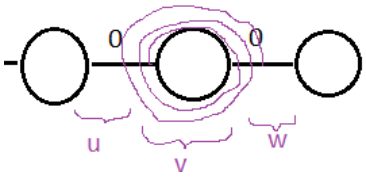


Figure 2.35: Finite Automata pumped at intermediate state finite time.

We can write $uv^iw \in L$ for i={1, 2, 3...}.

To prove the language as non-regular, smallest string of language is divided into uvw by considering two conditions.

i. $V \notin \varepsilon$.
ii. $|UV| \leq n$ where n is the smallest string of the language. To prove the language as a non-regular, throughout the proof the value of n is constant.

Example 2.7: Prove that the language $L=\{0^{i^2} : i \geq 1\}$ is not regular.

L={0, 0000, 000000000...}
n u v w

$uv^iw \in L$ for i={1, 2…}

$0(00)^i000000 \in L$

I=1

000000000 ЄL

I=2

00000000000 ∉ L

When i=2 the string obtained is not belongs to language, hence proved language is not regular.

Example 2.8: Prove that the language $L=\{a^nb^{n+1} : n \geq 2\}$ is not regular.

L={aabbb, aaabbbb....}
n u v w

i=1

aaabbbb Є L

I=2

Aaabaabbbb ∉ L

When i=2 the string obtained is not belongs to language, hence proved the language is non-regular.

Closure Properties of Regular Expression:

Till now we have seen many regular expressions and their applications. These regular expressions are bounded/ related by operators called closure properties of regular expression. The different operators used to relate regular expressions are:

i. Boolean.
ii. Reversal.
iii. String Homomorphism and Inverse Homomorphism.

i. Boolean Operator:

They are of different types

a. Union b. Intersection c. Difference d. Compliment.

a. Union Operation.

To perform union operation it uses following steps.

- This operation is applied on two machines defined as

$M_1=(Q_1, \sum_1, \delta_1, IS_1, FS_1)$

$M_2=(Q_2, \sum_2, \delta_2, IS_2, FS_2)$

- Find the language L1 and L2 of the two machines.
- Perform union operation on the two languages L1 and L2 to obtain language L3
- Combine two machines to perform union operation.
- Apply language L3 on combined machines to obtain machine for union operation.

Same steps are used for the other Boolean operation.

$M_1UM_2=(Q_1UQ_2, \sum_1U\sum_2, \delta_1U\delta_2, ?, ?)$

For example consider the machine 1 is shown in figure 2.36

i.

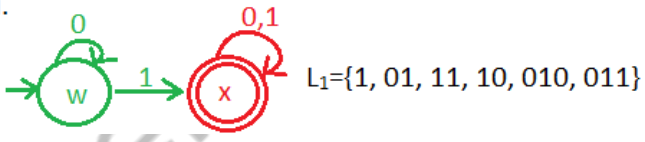


Figure 2.36: Finite Automata

Machine 2 is shown in figure 2.37

ii.

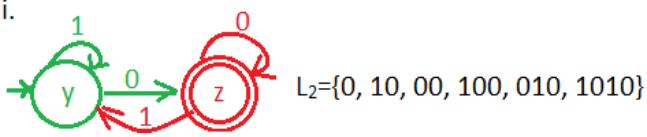


Figure 2.37: Finite Automata

The union operation on two languages is

$$L_1UL_2=\{0, 1, 00, 01, 10, 11, 010, 011, 010, 100, 1010\}$$

In order to obtain the finite automata for the union operation, we need to join two finite automata. The transition table for union operation is shown in table…

Table 2.1: Transition table for Boolean operation

δ_{1U2}	0	1
w, y	w, z	x, y
w, z	w, z	x, y
x, y	x, z	x, y
x, z	x, z	x, y

Equivalent diagram is shown in the figure 2.38

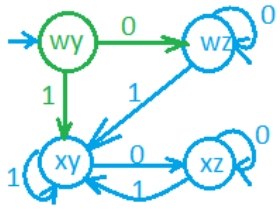


Figure 2.38: Finite Automata after combining figure 2.36 and 2.37.

According to union operation it has to accept 0, 00

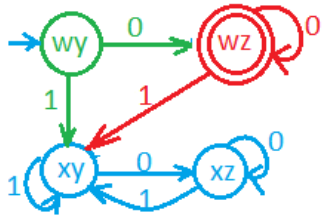


Figure 2.39: Finite Automata to accept 0, 00.

it has to accepts 1, 01, 11, 011

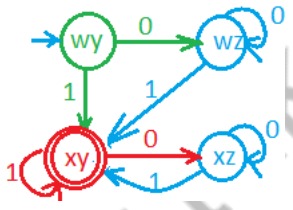


Figure 2.40: Finite Automata accepts 1, 01, 11, 011.

It also accepts 10, 010…

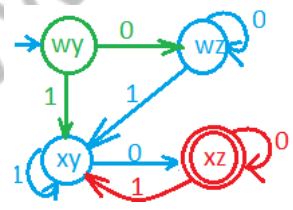


Figure 2.41: Finite Automata accepts 10, 010...

The final automata for the union operation is shown in figure 2.42

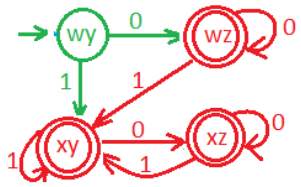


Figure 2.42: Union of two finite automata of figure 2.36 and 2.37.

For intersection M1 ∩ 2={10, 100, 010, 1010…}

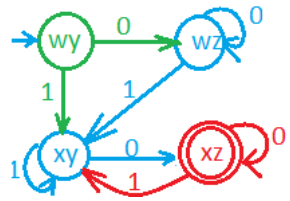


Figure 2.43: Intersection of two finite automata of figure 2.36 and 2.37.

For differentiation M_{1-2} ={1, 11, 01, 011, 101}

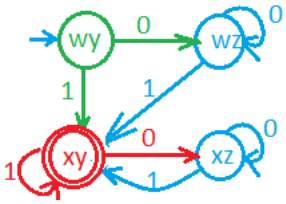


Figure 2.44: Difference of two finite automata of figure 2.36 and 2.37.

Compliment:

To compliment the regular expression it uses the following steps:

i. Obtain the Ɛ-NFA/NFA for the regular expression.
ii. Convert the Ɛ-NFA to DFA.
iii. Compliment DFA.
iv. Convert DFA to regular expression.

Example 2.9: Find the compliment of regular expression 0.

i. Obtain the Ɛ-NFA/NFA for the regular expression.

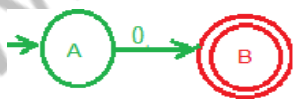


Figure 2.45: finite automata for regular expression 0.

ii. Convert the Ɛ-NFA/NFA to DFA.

δ(A, 0)=B; δ(B, 0)=φ; δ(φ,0)=φ

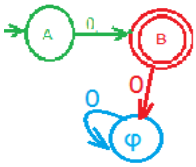


Figure 2.46 : Equivalent DFA for the figure 2.45.

iii. Compliment DFA. Reverse all transitions and convert all final to non final and vice-versa.

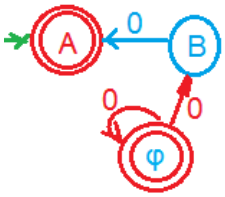


Figure 2.47: Complimented DFA of figure 2.46.

State B and φ are not reachable from the initial State. Hence the two states are removed from the finite automata.

Figure 2.48. Complimented DFA for the regular expression 0.

The regular expression of finite automata shown in figure 2.48 is

RE=Ɛ

Reversal of regular expression:

i. Reverse all transitions.
ii. Convert initial state to final and final state to intermediate state.
iii. Draw epsilon-transition between new initial state to all previous final states, mark new state as an initial state.

Figure 2.49: Non-deterministic finite automata.

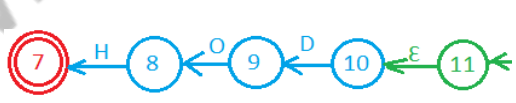


Figure 2.50: Reversed NFA of the figure 2.49

String Homomorphism:

String homomorphism is a function on the language, it replaces string by symbol

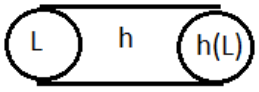


Figure 2.51: Homomorphism.

Inverse Homomorphism:

It is an inverse string homomorphism

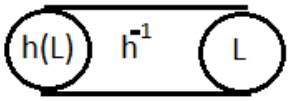


Figure 2.52: Inverse string homomorphism

Example 2.10: Consider the homomorphism, h(0)=abb; h(1)= a; h(011)=abbaa

If the homomorphism from an alphabet {0, 1, 2} to {a, b} is defined by h(0)=ab; h(1)=b; & h(2)=aa, then:

i. What is h(2201).
ii. If L is language 1*02* what is h(L).
iii. If L is language(ab+baa)*bab then what is $h^{-1}(L)$

i. h(2201)=aaaaabb
ii. h(L)=h(1*02*)=b*ab(aa)*
iii. (ab+baa)*bab=(0+12)*10

Equivalence and minimization:

In Equivalence, two different DFA's are given, we need to find out the two DFA's are similar or not. By filling table and Preshapat tree we can decide the two DFAs are similar | not similar.

Preshapat Tree: To find pair of nodes similar | not similar Preshapat tree is used. It consists of root node, intermediate nodes and leaves nodes. The root node of the tree is pair of initial states of two DFAs. Next we construct branches of tree depending on number of input symbols. If leaves node do not have a cross in the difference table and same pairs are repeating, we stop constructing tree and decide the root pair is similar or else dissimilar, any of leaves node has a cross in the difference table.

Figure 2.53: Deterministic finite automata.

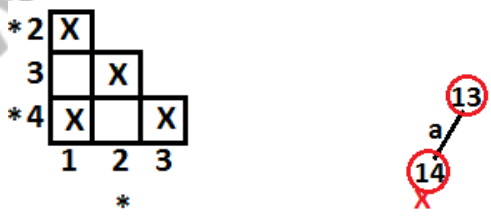


Figure 2.54: Difference table with Preshapat tree.

Leaves node 14 has cross in the difference table, so two DFA's are not equal.

For example

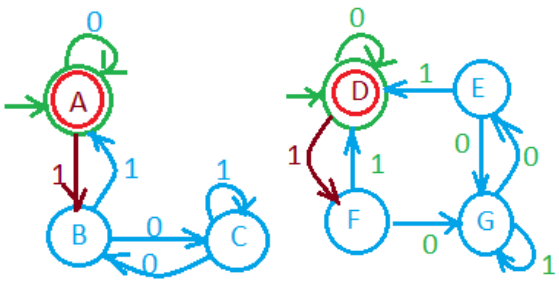


Figure 2.55: Deterministic finite automata.

Note that root of the tree is pair of initial states. Leaves nodes of the tree do not have cross in the difference table and nodes are repeating, hence the two DFA's are similar.

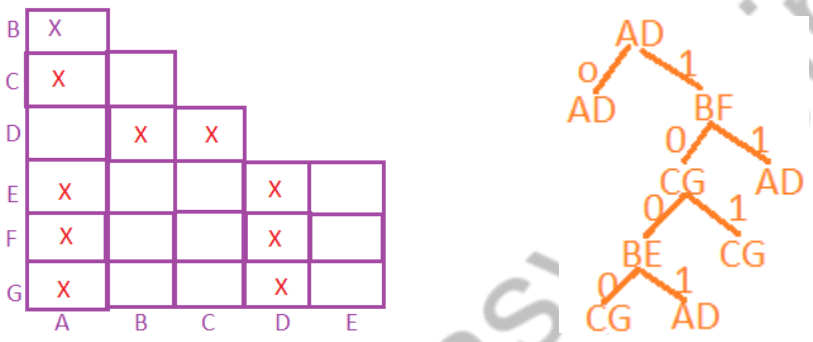


Figure 2.56: Difference table with Preshapat tree.

Example 2.11: Find out the two DFAs are similar or not.

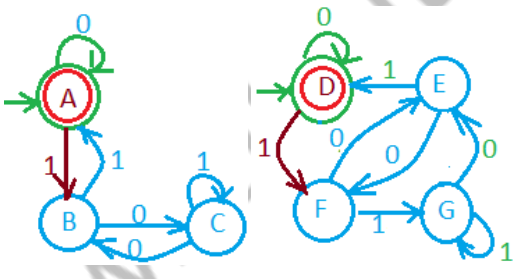


Figure 2.57: Deterministic finite automata.

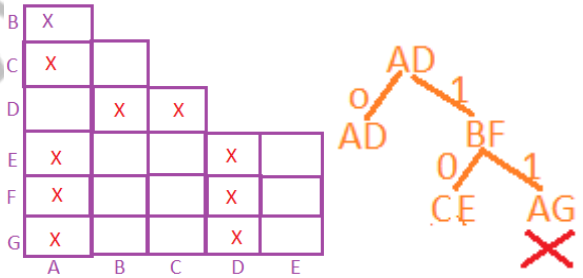


Figure 2.58: Transition tree with table.

The leaves node AG has X in the table; hence two DFAs are dissimilar.

In case of minimization, the two states are minimized to one state if they are similar. For example state X and Y are equal, they are replaced by one state as shown in figure 2.59

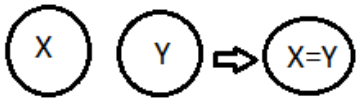


Figure 2.59: Similar states are replced into single state.

To decide the states are similar | not similar we use Shanprepat tree.

Shanprepat Tree: To find equivalence between pair of states Shanprepat tree is used. It consists of root node, intermediate nodes and leaves nodes. The root node of the tree is pair of states used to find the similar | not similar states. Next we construct branches of tree depending on number of input symbols. If leaves node do not have a cross in the difference table and same pairs are repeating, we stop constructing tree and decide the root pair is similar or else decide two states are dissimilar, any of leaves node has a cross in the difference table.

Example 2.12: Using table filling method, minimize the DFA shown in figure 2.60 and draw the transition diagram of the resulting DFA.

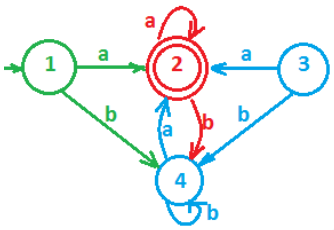


Figure 2.60: Deterministic finite automata.

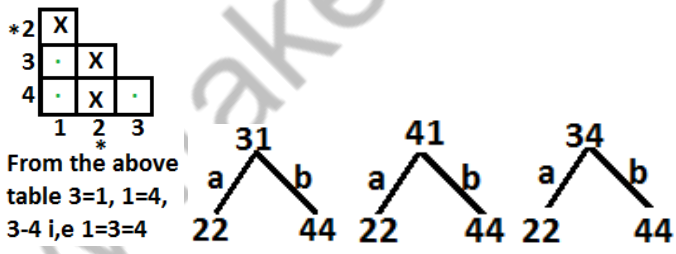


Figure 2.61: Differnce table with Shanprepat Tree

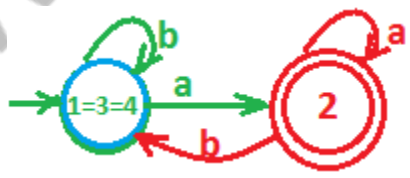


Figure 2.62: Minimized DFA of the figure 2.60

Example 2.13: Using table filling method, minimize the DFA shown in figure 2.63 and draw the transition diagram of the resulting DFA.

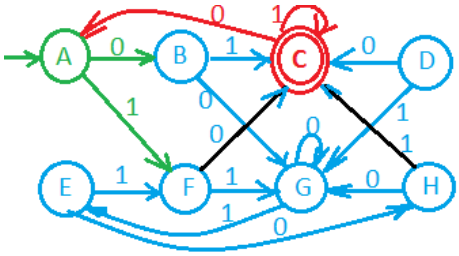


Figure 2.63: Deterministic finite automata.

From the given DFA we draw the dissimilar table as shown in figure. As compared to final state C all states are different (non-final states), hence X is marked.

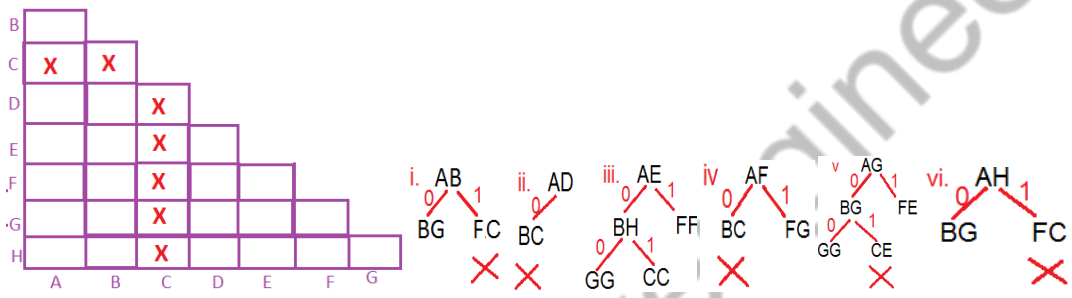


Figure 2.64: Differnce table with Shanprepat Tree.

The leaves node decides the root node. Any leaves node has X in the table, the root node is marked as X or else pair node repeats. Stop when the pair nodes start repeating, and decide the root node pair is similar.

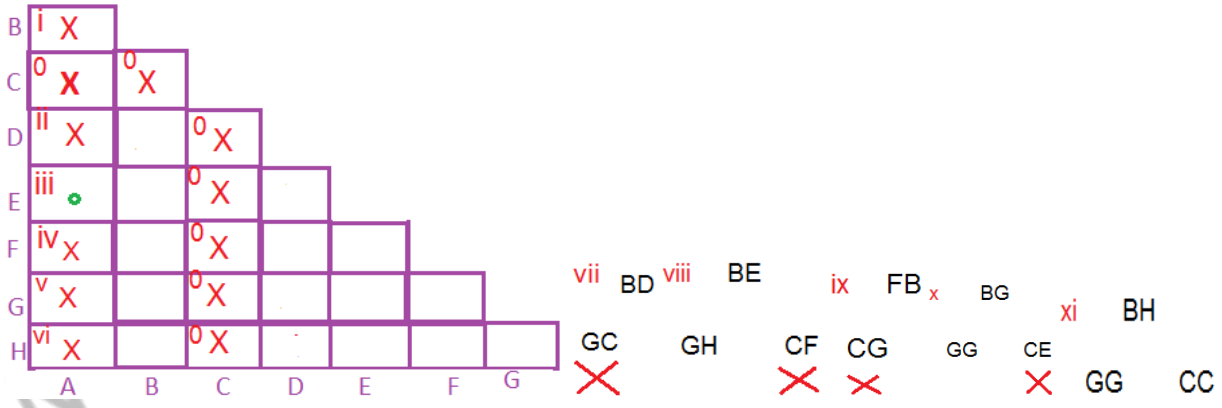


Figure 2.64: Differnce table with Shanprepat Tree.

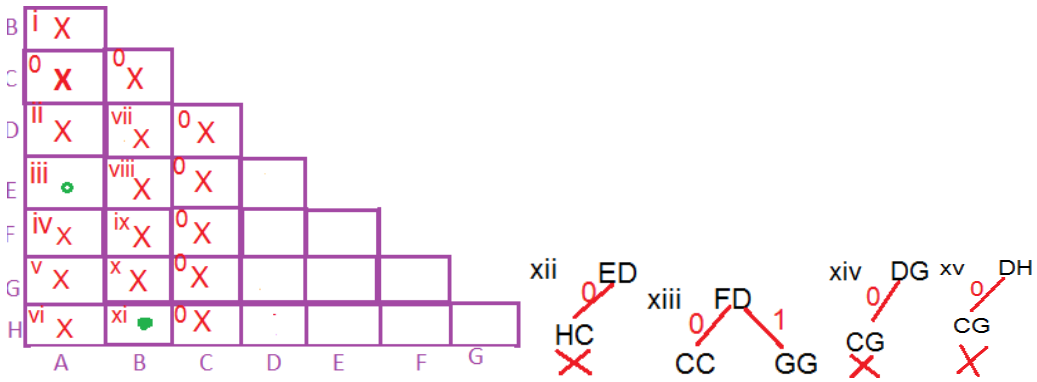


Figure 2.65: Differnce table with Shanprepat Tree.

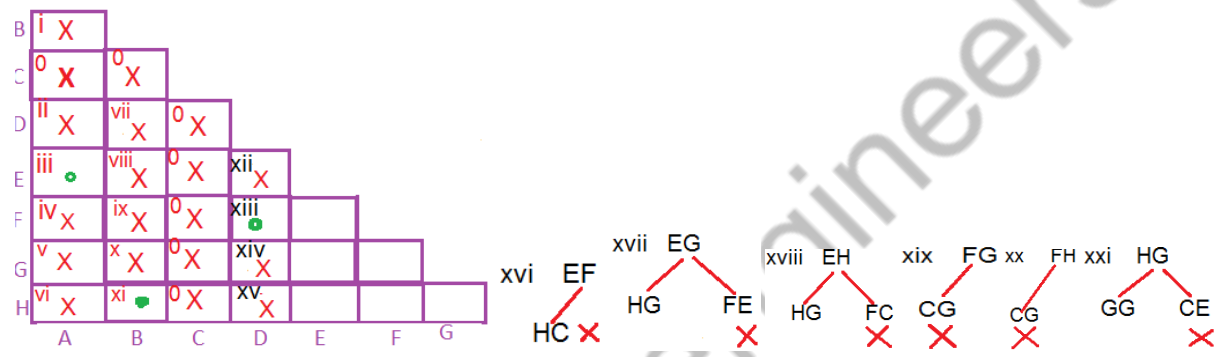


Figure 2.66: Differnce table with Shanprepat Tree.

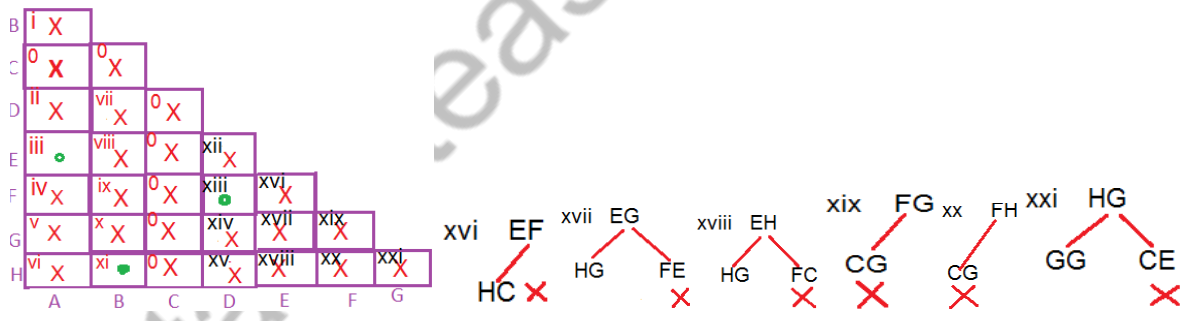


Figure 2.66: Differnce table with Shanprepat Tree.

From the table, we come to know that A=E, B=H and D=F. the resulting DFA is shown in figure..

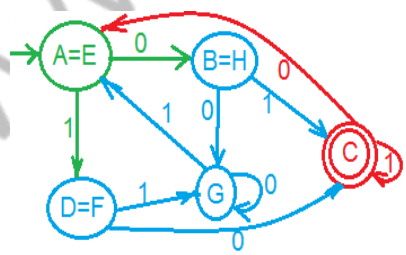


Figure 2.67: Minimized DFA of the figure 2.63

Example 2.14: Using table filling method, minimize the DFA shown in table and draw the transition diagram of the resulting DFA.

δ	0	1
→A	B	E
B	C	F
*C	D	H
D	E	H
E	F	I
*F	G	B
G	H	B
H	I	C
*I	A	E

The table shows the dissimilarity in the states of the given problem.

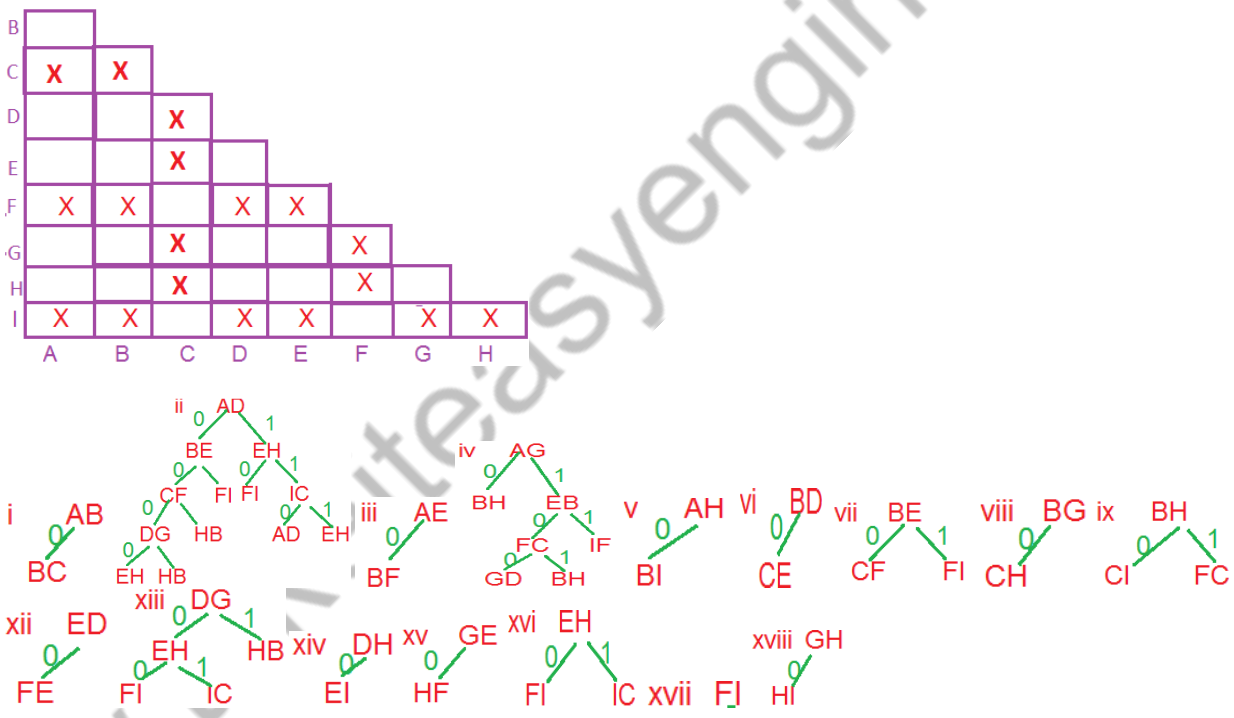
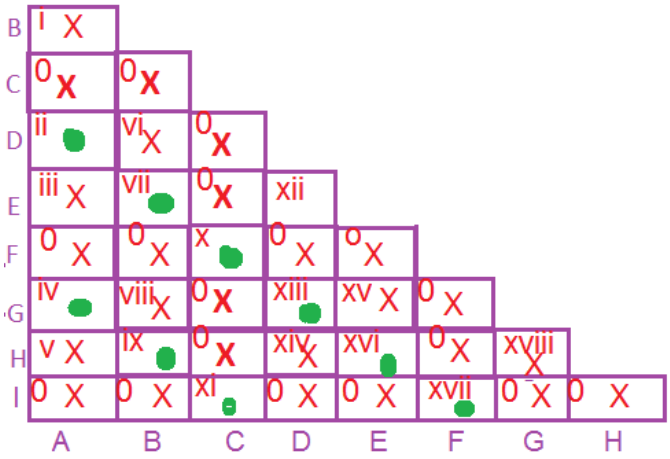


Figure 2.68: Differnce table with Shanprepat Tree.



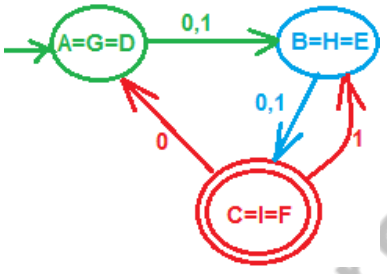


Figure 2.69: Differnce table with minimized DFA.

Chapter 3

CONTEXT FREE GRAMMAR

To define a language, English grammar is used such a concept is called as Context Free Language (CFL) or Context Free Grammar (CFG). For example:

<exp> → <exp><exp>

<exp> → id // it is a symbol.

L={0}*={Ɛ, 0, 00, 000, …}

L is represented in the form of grammar as $S \rightarrow 0S|\varepsilon$

It means S has two values S=Ɛ or S= 0S

In general grammar is defined as G=(V, T, P, S)

Where, V is a set of variables. (First letters)

T is a set of terminals. (Second letters)

P is Productions $(S \rightarrow 0S|\varepsilon)$

S is starting variable of the grammar.

In order to construct grammar it uses four different types of productions:

i. roduction for finite set of strings, 0^*={ Ɛ+0+00…} is $S \rightarrow 0S|\varepsilon$
and 0^+ ={0+00+000…} the production is $S \rightarrow 0S|0$

ii. Production which separate the symbols (a^nb^n where, n={0, 1, 2,,}) is $S \rightarrow 0S1|\varepsilon$
Production which mix the symbols $((ab)^n$ where, n is {0, 1, 2,,}) is $S \rightarrow 01S|01$.

iii. Production for $(0+1)^*$ is $S \rightarrow 0S|1S|\varepsilon$.

Example 3.1: Construct the grammar for the language L={a^nb^n n≥0}

L={Ɛ, ab. aabb, ….}

$S \rightarrow aSb|\varepsilon$

G = (V, T, P, S)

V=S, T= {a, b}

P is production

S is starting symbol = S

Example 3.2: Construct the grammar for the language L={$a^n b^m$ |n<m}

L= {Ɛ, b. bb, …abb, aabbb ….}

$B \rightarrow b|bB$

$S \rightarrow aSb|B$

G=(V, T, P, S)

V={S, B} T= {a, b}

P is production

S=S

Example 3.3: Construct the grammar for palindrome over the ∑={a, b}

L={Ɛ, a, b. aba, abba, bab, baab ….}

$S \rightarrow \varepsilon| a|b|aSa|bSb$

G=(V, T, P, S)

V=S, T={a, b}

P is production

S=S

Example 3.4: Construct the grammar for the language L={x |$n_a(x)=n_b(x)$}

L={Ɛ, ab. ba, aabb, abab….}

$S \rightarrow \varepsilon|abS|aSb|Sab|bSa|baS|Sba$

G= (V, T, P, S)

V=S, T= {a, b}

P is production

S=S

Example 3.5: Construct the grammar for the language L={x |$n_a(x) \neq n_b(x)$}

L={a, aab. aba, baa, ….b, bba, bab, abb…}

$A \rightarrow a|aSb|abS|bSa|baS|Sab|as$

$B \rightarrow b|aSb|abS|bSa|baS|Sab|bs$

$S \rightarrow A|B$

G= (V, T, P, S)

V={A, B, S} T={a, b}

P is production

S=S

Example 3.6: Construct the grammar for the language L= $\{0^i1^j2^k | i=j \text{ or } j=k\}$

Describe the language according to four rules mentioned above

L={ε, 2, 22, 222...01, 0011, 000111.... 0, 00, 000.... 12, 1122, 111222.. 012, 001122..}

i=j j=k

For i=j the productions are

$A \rightarrow \varepsilon|2A$

$B \rightarrow 01|0B1$

For j=k the productions are

$C \rightarrow \varepsilon|0C$

$D \rightarrow 12|1D2$

To obtain the language L= $\{0^i1^j2^k | i=j \text{ or } j=k\}$ club the above productions

$S \rightarrow BA|CD$

G= (V, T, P, S)

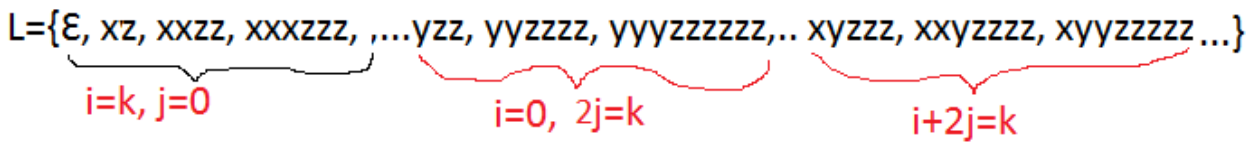
V={S, B, A, C, D} T= {0, 1, 2}

P is production

S=S

Example 3.7: Construct the grammar for the language L= $\{x^iy^jz^k | i+2j=k\}$

Describe the language according to four rules mentioned above



For i=k, j=0 the production is

$A \rightarrow xAz$

For i=0, j=k the production is

$B \rightarrow yBzz|\varepsilon$

The production for the language L= $\{x^i y^j z^k | i+2j=k\}$ is

$A \rightarrow xAz|B$

G= (V, T, P, S)

V= {B, A} T={x, y, z}

P is production

S=A

Example 3.8: Construct the grammar for the language L= {w|wЄ(0+1)*} with at least one occurance of 110

L= {110, 1100, 0110, 1110,1101, …

$A \rightarrow 0A|1A|\varepsilon$

$B \rightarrow 110$

$S \rightarrow ABA$

G=(V, T, P, S)

V= {B, A, S} T={0, 1}

P is production

S=S

Derivation Tree/parse Tree:

It is a tree, consisting of root node, intermediate node and leaf node. They are of two types:

i. Right Most Derivation Tree (RMD)
ii. Left Most Derivation Tree (LMD)

Right Most Derivation Tree (RMD): In RMD right most variable simplified followed by other variables.

Left Most Derivation Tree (LMD): In LMD left most variable simplified followed by other variables.

Example 3.9: Construct LMD and RMD for the string xxxyyy using production

S → BxxB B → xB|yB| ε

LMD:

S → BxxB
S → xBxxB
S → xεxxB
S → xxxB
S → xxxyB
S → xxxyyB
S → xxxyyyε
S → xxxyyy

RMD:

S → BxxyB
S → BxxyyB
S → BxxyyyB
S → Bxxyyyε
S → Bxxyyy
S → xBxxxyyy
S → xεxxyyy
S → xxxyyyε
S → xxxyyy

For LMD and RMD same tree

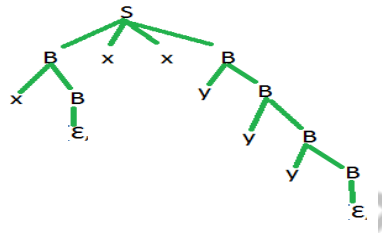


Figure 3.1: LMD and RMD for the string xxxyyy

Identifier:

Identifier is letters followed by zero or more than zero letters or digits. We can write identifier in the form of expression over ∑={a, b, 0, 1} as follows:

I= (a + b) (a+b+0+1)*

Identifier itself is an expression, represented in the form of production as shown below

E → I

Expression can be

E → E+E|E-E|E*E|E/E|(E)

According to mathematical equation of identifier, we can write the following productions for the identifier as

I → a|b|Ia|Ib|I0|I1

Example 3.10: Design grammar for valid expression over operator + and *. The argument of the expressions is valid identifier over symbols a, b, 0, 1. Derive LMD and RMD for the string w=(a11+b0)*(b00+001).

LMD

E→ E*E

E→ (E)*E

E → (E+E)*E
E → (I+E)*E
E → (I1+E)*E
E → (I01+E)*E
E → (I101+E)*E
E → (a101+E)*E
E → (a101+I)*E
E → (a101+I1)*E
E → (a101+b1)*E
E → (a101+b1)*(E)
E → (a101+b1)*(E+E)
E → (a101+b1)*(I+E)
E → (a101+b1)*(I1+E)
E → (a101+b1)*(a1+E)

E→(a101+b1)*(a1+I)
E→(a101+b1)*(a1+b)

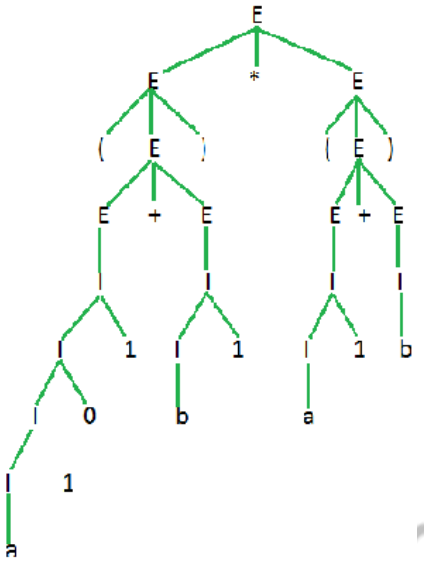


Figure 3.2: LMD for the string w= (a11+b0)*(b00+001).

RMD

E →E*E
E →E*(E)
E →E*(E+E)
E→ E*(E+I)
E→ E*(E+b)
E →E*(I+b)
E→ E*(I1+b)
E →E*(a1+b)
E →(E)*(a1+b)
E →(E+E)*(a1+b)
E →(E+I)*(a1+b)
E →(E+I1)*(a1+b)
E →(E+b1)*(a1+b)
E →(I+b1)*(a1+b)
E →(I1+b1)*(a1+b)
E →(I01+b1)*(a1+b)
E →(a101+b1)*(a1+b)

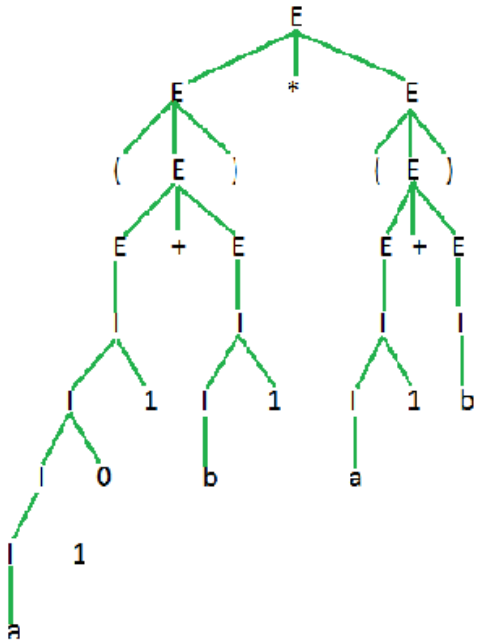


Figure 3.3: RMD for the string w= (a11+b0)*(b00+001) .

Ambiguity:

Two different trees for a LMD or RMD called ambiguity. For example, the production:

$S \rightarrow S+S|a$

for the string a+a+a, two LMD and RMD are shown in figure 3.4 and 3.5 respectively.

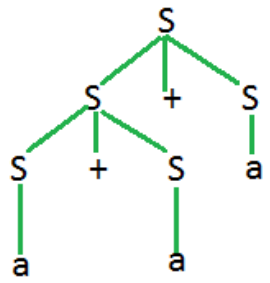
First LMD

$S \rightarrow S+S+S$

$S \rightarrow a+S+S$

$S \rightarrow a+a+S$

$S \rightarrow a+a+a$



Second LMD

$S \rightarrow S+S$

$S \rightarrow a+S$

$S \rightarrow a+S+S$

$S \quad a+a+S$

$S \quad a+a+a$

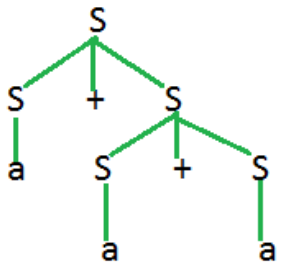


Figure 3.4: Two different LMD for the production $S \rightarrow S+S|a$ over the string a+a+a

First RMD

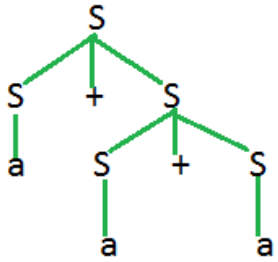
$S \rightarrow S+S$

$S \rightarrow S+S+S$

$S \rightarrow S+S+a$

$S \rightarrow S+a+a$

$S \rightarrow a+a+a$



Second RMD

$S \rightarrow S+S$

$S \rightarrow S+a$

$S \rightarrow S+S+a$

$S \rightarrow S+a+a$

$S \rightarrow a+a+a$

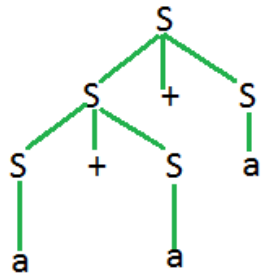


Figure 3.5: Two different RMD for the production $S \rightarrow S+S|a$

Example 3.11: Show that the given grammar is ambiguous by using LMD and RMD for the production $S \rightarrow SS|(S)|\varepsilon$ over a string (()()()).

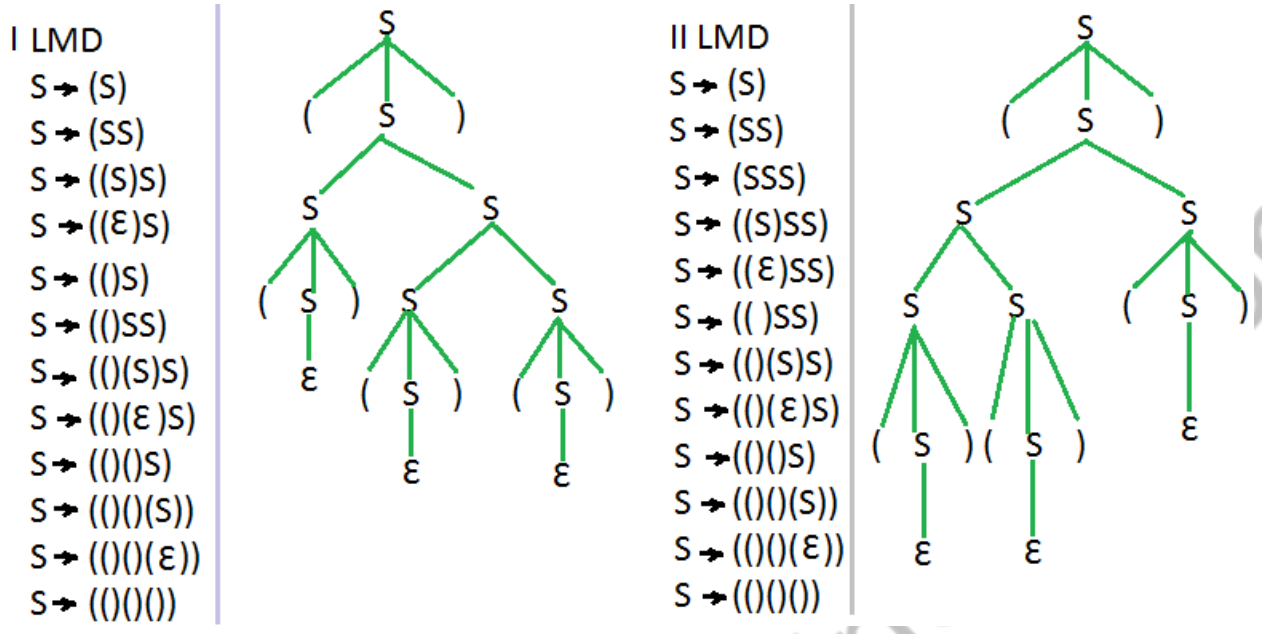


Figure 3.6: Two different LMD for the production $S \rightarrow SS \mid (S) \mid \varepsilon$ over a string (()()()).

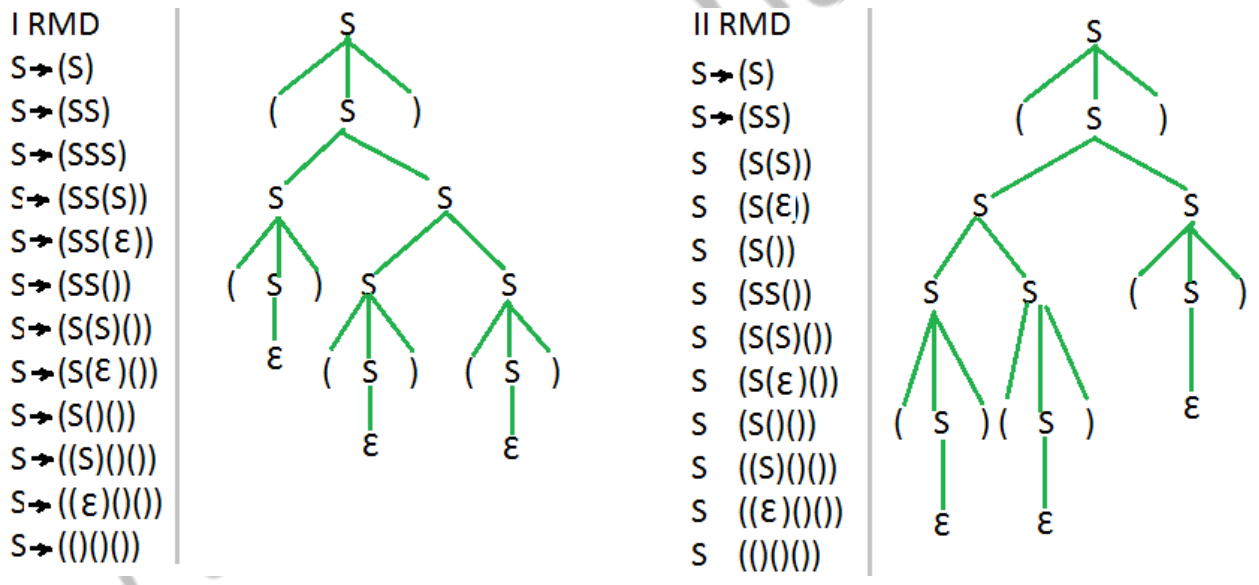


Figure 3.7: Two different RMD for the production $S \rightarrow SS \mid (S) \mid \varepsilon$ over a string (()()()).

For both RMD and LMD, the tree obtained for the same production has two different trees called Ambiguity.

Resolve Ambiguity:

There are two reasons for the ambiguity

i. Recursive use of variables.
ii. Similar operands used in the expression.

To resolve ambiguity, two variables are used

i. **Factor:** Factor can be identifier or parenthesis expression written as: $F \rightarrow I|(E)$.
ii. **Term:** Term is a factor$^+$ given as $T \rightarrow FT|F$

Example 3.12: Resolve the ambiguity in the given production $S \rightarrow aS|Sa|\varepsilon$

$S \rightarrow aS|Sa|\varepsilon$
$S \rightarrow aSa|Sa|\varepsilon$
$S \rightarrow aT|T|\varepsilon \quad T \rightarrow Sa|\varepsilon$
$F \rightarrow \varepsilon$

Example 3.13: Resolve the ambiguity in the given production $E \rightarrow E+E|E-E|E*E|E/E|(E)|a$

$E \rightarrow E+E|E-E|E*E|E/E|(E)|a$
$E \rightarrow E+E|E-E|E*E|E/(E)|(E)|a$
$E \rightarrow E+E|E-E|E*E|E/T|T|a \quad T \rightarrow (E)|a$
$E \rightarrow E+E|E-E|E*E|E/T|T|a$
$E \rightarrow E+E|E-E|E*E/T|E/T|T|a$
$E \rightarrow E+E|E-E|E*R|R|T|a \quad R \rightarrow E/T|a$
$E \rightarrow E+E|E-E|E*R|R|T|a$
$E \rightarrow E+E|E-E*R|E*R|R|T|a$
$E \rightarrow E+E|E-S|S|R|T|a \quad S \rightarrow E*R|a$
$E \rightarrow E+E|E-S|S|R|T|a$
$E \rightarrow E+E-S|E-S|S|R|T|a$
$E \rightarrow E+P|P|S|R|T|a \quad P \rightarrow E-S|a$
$F \rightarrow a$

Chapter 5

PUSHDOWN AUTOMATA (PDA)

To process input string PDA uses stack along with input, such an automata is called Push Down Automata (PDA). PDA is defined with seven tuples, (Q, ∑, δ, Z_0, ┌, IS, FS)

Q is number of states in PDA

∑ is input alphabet

δ is transition function defined as $Qx\sum x\varepsilon x\Gamma \rightarrow Q, \Gamma^*$

Z_0 is initial symbol of stack

┌ is stack symbols

IS is initial state

FS is final state

The PDA works on equal symbol, if the input string is 100001.

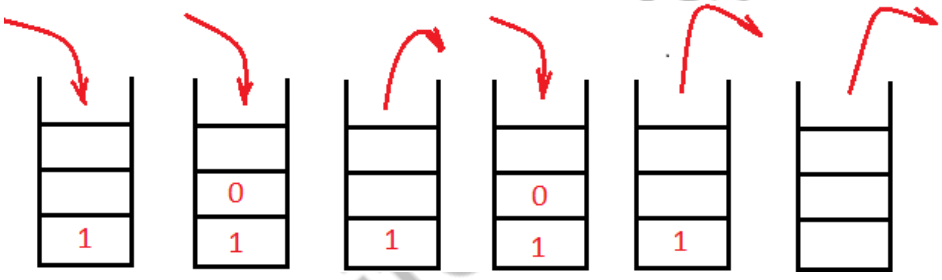


Figure 5.1: Different position of stack for the input string is 100001.

There are two types of PDA

i. Deterministic Push Down Automata (DPDA)
ii. Non-deterministic Push Down Automata (NPDA)

Deterministic Push Down Automata (DPDA):

In Deterministic Push Down Automata (DPDA), we are designing PDA. The design of PDA begins with Instantaneous Description (IDs), each IDs consist of state, input string to be processed and initial symbol of the stack (z_0).

Example 5.1: Design DPDA for the string 000111.

The Instantaneous Descriptions (IDs) for the given problem are as follows:

ID1 ID2 ID3 ID4

$q_0, 000111, z_0 \vdash q_0, 00111, 0z_0 \vdash q_0. 0111, 00z_0 \vdash q_0, 111, 000z_0 \vdash$

$q_0, 11, 00\ z_0 \vdash q_0, 1, 0\ z_0 \vdash q_0, \varepsilon, z_0 \vdash q_1, \varepsilon, z_0.$

ID5 ID6 ID7 ID8

By making use of the above IDs we can obtain the transition functions, which are required for the construction of PDA.

For first transition function, ID1 is used for input transition and ID2 is used as output.

$Qx\sum x\varepsilon x\Gamma \to Q, \Gamma^*$

$\delta(q_0, 0, z_0)=q_0, 0z_0$

Next ID2 is the input of transition and ID3 is used for output transition function.

$\delta(q_0, 0, 0)=q_0, 00$

ID3 and ID4 are same as ID2 and ID3

ID4 and ID5

$\delta(q_0, 1, 0)=q_0, \varepsilon$

ID7 and ID8

$\delta(q_0, \varepsilon, z_0)=q_1, z_0$

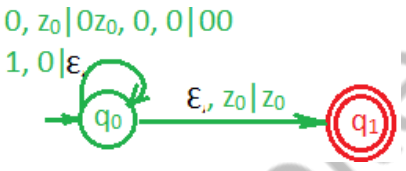


Figure 5.2: DPDA for the string 000111.

Example 5.2: Design DPDA for the language $L=\{a^n b^{2n} \mid n\geq 0\}$.

The Instantaneous Descriptions (IDs) for the given problem are as follows

ID1 ID2 ID3 ID4

$q_0, aabbbb, z_0 \vdash q_0, abbbb, aaz_0 \vdash q_0, bbbb, aaaaz_0 \vdash q_0, bbb, aaaz_0$

$q_0, bb, aaz_0 \vdash q_0, b, az_0 \vdash q_0, \varepsilon, z_0 \vdash q_1, \varepsilon, z_0$

ID5 ID6 ID7 ID8

By making use of the above IDs we can obtain the transition functions, which are required to construct the PDA.

For first transition function, ID1 is used for input transition and ID2 is used as output.

$Qx\sum x\varepsilon x\Gamma \to Q, \Gamma^*$

$\delta(q_0, a, z_0)=q_0, aaz_0$

Next ID2 is the input of transition and ID3 is used for output transition function.

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

ID3 and ID4

$\delta(q_0, b, a)=q_0, \varepsilon$

ID7 and ID8

$\delta(q_0, \varepsilon, z_0)=q_1, z_0$

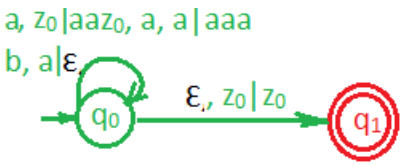


Figure 5.3 DPDA for the language $L=\{a^n b^{2n} \mid n\geq 0\}$.

Example 5.3: Design DPDA for the language $L=\{x \mid n_a(x)=N_b(x)\}$.

$L=\{\varepsilon, ab, ba, abab, baba, aabb, bbaa \ldots\}$

The Instantaneous Descriptions (IDs) for the given problem are as follows

ID1 ID2 ID3 ID4 ID5

$q_0, aabb, z_0 \vdash q_0, abb, az_0 \vdash q_0, bb, aaz_0 \vdash q_0, b, az_0 \vdash q_0, \varepsilon, z_0 \vdash$

$q_1, \varepsilon\ z_0$

ID6

By making use of the above IDs we can obtain the transition functions, which are required for the construction of a PDA.

For first transition function, ID1 is used for input transition and ID2 is used as output.

$Qx\sum x\varepsilon x\Gamma \rightarrow Q, \Gamma^*$

$\delta(q_0, a, z_0)=q_0, az_0$

Possible transition

$\delta(q_0, b, z_0)=q_0, bz_0$

Next ID2 is the input of transition and ID3 is used for output transition function.

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

Possible transition

$\delta(q_0, b, b)=q_0, bb$

ID3 and ID4

$\delta(q_0, b, a)=q_0, \varepsilon$

$\delta(q_0, a, b)=q_0, \varepsilon$

ID5 and ID6

$\delta(q_0, \varepsilon, z_0)=q_1, z_0$

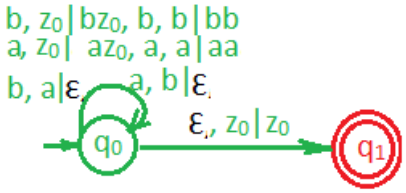


Figure 5.4: DPDA for the language $L=\{x \mid n_a(x)=N_b(x)\}$.

Example 5.4: Design DPDA for the language $L=\{a^n b^n c^n \mid n\geq 0\}$.

The Instantaneous Descriptions (IDs) for the given problem are as follows

$$\underset{ID1}{q_0, aabbcc, z_0} \vdash \underset{ID2}{q_0, abbcc, az_0} \vdash \underset{ID3}{q_0, bbcc, aaz_0} \vdash \underset{ID4}{q_0, bcc, az_0}$$

$$q_0, cc, z_0 \vdash q_0, c, cz_0 \vdash q_0, \varepsilon, ccz_0 \vdash q_0, \varepsilon, cz_0 \vdash q_0, \varepsilon, z_0 \vdash q_1, \varepsilon, z_0$$

ID5 ID6 ID7 ID8 ID8 ID9

By making use of the above IDs we can obtain the transition functions, which are required to construct the PDA.

For first transition function, ID1 is used for input transition and ID2 is used as output.

$Qx\sum x\varepsilon x\Gamma \rightarrow Q, \Gamma^*$

$\delta(q_0, a, z_0)=q_0, az_0$

Next ID2 is the input of transition and ID3 is used for output transition function.

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

ID3 and ID4

$\delta(q_0, b, a)=q_0, \varepsilon$

$\delta(q_0, c, z_0)=q_0, cz_0$

$\delta(q_0, c, c)=q_0, cc$

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

ID8 and ID9

$\delta(q_0, \varepsilon, z_0)=q_1, z_0$

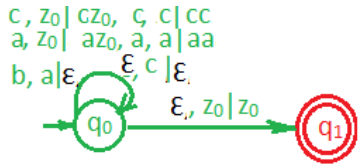


Figure 5.5: DPDA for the language $L=\{a^n b^n c^n \mid n\geq 0\}$.

i. Non-deterministic Push Down Automata:

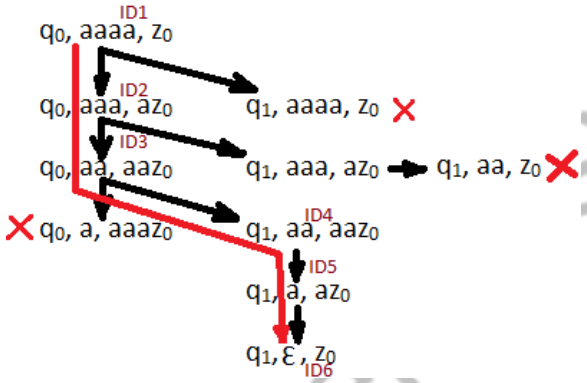
Example 5.5: Design NPDA for the language $L=\{ww^R \mid w=\{a, b\}^*\}$.

L={Ɛ, aa, bb, aaaa, bbbb, abba…}

In NPDA, it treats all the symbols into

i. Symbol of first half continues in the same state and push the symbol to top of the stack.
ii. Symbol of second half, it just changes the state and continues further.

The Instantaneous Descriptions (IDs) for the given problem are as follows



$\delta(q_0, a, z_0)=q_0, az_0$

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

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

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

$\delta(q_1, \varepsilon, z_0)=q_2, z_0$

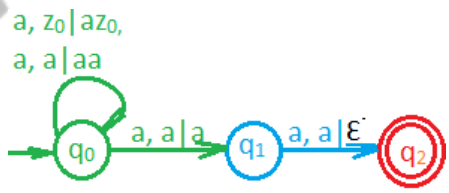


Figure 5.6: NPDA for the language $L=\{ww^R \mid w=\{a, b\}^*\}$.

Equivalence of CFG and PDA:

Each CFG has PDA and each PDA has CFG.

i. Converting PDA to CFG
ii. Converting CFG to PDA

i. Converting PDA to CFG

In this method we convert transition functions into variables. Each variable consist of state, stack symbol and state, all are represented in square bracket [Q, Stack Symbol, Q].

Example 5.6: Convert following transition function of PDA to productions.

$\delta(q_0, a, z_0)=q_0, XZ_0$; $\delta(q_0, a, z_0)=q_0, X$;

This conversion method is a blind, we do not have any specific rules

CFG begins with starting symbol. Therefore starting symbol in the form of transition function of PDA is derived as:

$S \rightarrow [q_0, z_0, Q]$

$S \rightarrow [q_0, z_0, q_0] \mid [q_0, z_0, q_1]$

$\delta(q_0, a, z_0)=q_0, XZ_0$

The production for the above transition function can be written as

$[q_0, z_0, q_0] \rightarrow a[q_0, X, Q][Q, Z_0, q_0]$

$\rightarrow a[q_0, X, q_0][q_0, Z_0, q_0] \mid a[q_0, X, q_0][q_0, Z_0, q_0]$

$\delta(q_0, a, z_0)=q_0, X$

The production for the above transition function can be written as

$[q_0, z_0, q_0] \rightarrow aX$

ii. Converting CFG to PDA

In this case productions of CFG's are given; we need to find out equivalent transition functions of PDA.

Example 5.7: Convert following productions of CFG into transition function of PDA.

$S \rightarrow aSb \mid bSa \mid SS \mid \varepsilon$

Before converting CFG to PDA, see that all productions are in the form of Greibach Normal form. In Greibach Normal form there should not be undesired production and all the right hand side production are only terminal or terminal followed by any number of variables.

$S \rightarrow aSb|bSa|SS|\varepsilon$

$\rightarrow aSb|ab|bSa|ba|SS|S$

$\rightarrow aSA_b|aA_b|bSA_a|bA_a|SS \qquad A_a \rightarrow a \qquad A_b \rightarrow b$

$\rightarrow aSA_b|aA_b|bSA_a|bA_a|aSA_bS|aA_bS|bSA_aS|bA_aS$

$\delta(q_0, a, S)=q_0, SA_b \qquad \delta(q_0, a, S)=q_0, A_b$
$\delta(q_0, b, S)=q_0, SA_a \qquad \delta(q_0, b, S)=q_0, A_b$
$\delta(q_0, a, S)=q_0, SA_bS \qquad \delta(q_0, a, S)=q_0, A_bS$
$\delta(q_0, b, S)=q_0, SA_aS \qquad \delta(q_0, b, S)=q_0, A_bS$
$\delta(q_0, a, A_a)=q_0, \varepsilon \qquad \delta(q_0, b, A_b)=q_0, \varepsilon$
$\delta(q_0, \varepsilon, S)=q_1, S$

Closure Properties of CFLs:

CFLs are closed under operator; there are different types of operators

i. Union
ii. Concatenation
iii. Star Closure
iv. Intersection of two CFLs
v. Intersection of CFL and regular language
vi. Compliment of CFLs
vii. Reversal.
viii. Substitution.
ix. Inverse Homomorphism,

i. Union:
$CFL_1=G_1=(V_1, T_1, P_1, S_1) \quad CFL_2=(V_2, T_2, P_2, S_2)$
$CFL_3=G_3=(V_1UV_2, T_1UT_2, P_1UP_2, S_1|S_2)$

ii. Concatenation:
$CFL_3=G_3=(V_1V_2, T_1T_2, P_1P_2, S_1S_2)$

iii. Star Closure:
$CFL_1=G_1=(V_1, T_1, P_1, S_1)$
$S_1=TS_1|\varepsilon;$

iv. Intersection of two CFLs:
$CFL_1=L_1=\{a^nb^nc^i : n\geq 0, i\geq 0\}$
$CFL_2=L_2=\{a^ib^nc^n : n\geq 0, i\geq 0\}$ is not a CFL

Proof:

L1={Ɛ, c, ab, abc, aabb, cc, aabbcc…}

L2={Ɛ, a, bc, abc, aa, bbcc, aabbcc…}

L3= L1∩L2={Ɛ, abc, aabbcc…}

$=\{a^nb^nc^n : n\geq 0\}$

The above language is neither belongs to L1 nor L2 therefore intersection of two CFLs are not CFL.

v. Intersection of CFL and regular language:

CFL =(Q1, ∑1, δ1, ┌1, IS1, FS1)

Regular language M=(Q2, ∑2, δ2, IS2, FS2)

CFL∩M=((Q1, Q2), (∑1, ∑2), (δ1, δ2), ┌1, (IS1, IS2), (FS1, FS2))

Intersection consist of seven tuples, it is a CFL.

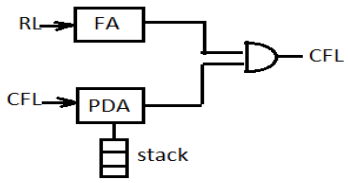


Figure 5.7: Intersection of CFL and regular language.

vi. Compliment of CFLs

CFL1, CFL2

According to De'morgan's theorem

$$\overline{CFL1} \cup \overline{CFL2} = \overline{CFL1 \cap CFL2}$$

vii. Reversal:

In this case the right hand side of production is reversed.

$S \rightarrow aB$ $S \rightarrow aSb$

$S^R \rightarrow Ba$ $S^R \rightarrow bSa$

viii. Substitution:

h(0)= aba; h(1)= ba

h(010)=ababaaba

S(0)= $\{a^n b^n : n \geq 0\}$; S(1)={aa, bb}

S(01)={$a^n b^n$aa, $a^n b^n$bb}

ix. Inverse Homomorphism:

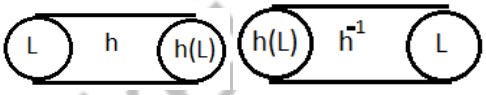


Figure 5.8: Homomorphism and inverse homomorphism

Figure 5.9: Substitution and inverse substitution.

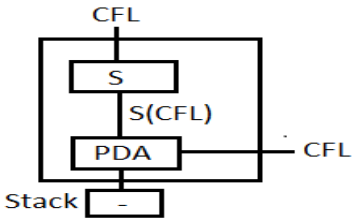


Figure 5.10: PDA for Substitution method.

Chapter 6

TURING MACHINE

Turing machine was invented by Alan Turing in 1936. It is a mathematical model of computer. Turing machines prove fundamental limitations on the power of mechanical computation. While they can express arbitrary computations, their minimalistic design makes them unsuitable for computation in practice: real-world computers are based on different designs that, unlike Turing machines, use random-access memory.

Turing completeness is the ability for a system of instructions to simulate a Turing machine. A programming language that is Turing complete is theoretically capable of expressing all tasks accomplishable by computers; nearly all programming languages are Turing complete if the limitations of finite memory are ignored.

The pictorial representation of Turing machine is shown in the figure 6.1. The machine consists of a finite control, which can be in any of a finite set of states. It has an input tape divided into finite cells; each can hold one symbol. Initially an input string over alphabet is stored in the tape. Default value of tape is blank symbol called B, it is a tape symbol it is not an input symbol. There is a tape head always pointing at the leftmost cell of the tape. A move of the Turing machine is a function of the state of the finite control and the tape symbol scanned. In one move Turing machine will:

i. Change state. The next state optionally may be the same as the current state.
ii. Write a tape symbol in the cell scanned. This tape symbol replaces whatever symbol was in that cell.

Turing machine can be defined with 5-tuples

$M_{T=}$ (Q, ∑, δ, B, ┌, IS, FS)

Q= Number of states used in the Turing machine.

∑= {a, b}

┌ = Tape symbol {a, b, B}

B = Blank symbol.

δ:

Q x ∑=Q, ┌, D where, D is direction

IS= Initial State of TM.

FS= Final state of TM.

For example if the input string is aabb, it is stored in the input tape shown in figure 6.1.

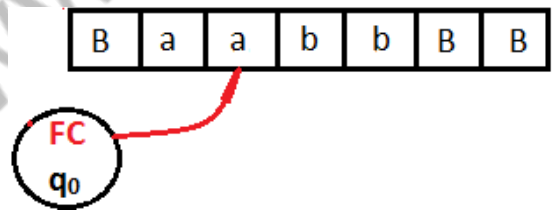


Figure 6.1: Turing Machine.

Turing machine uses IDs to process string; the IDs consist of state and input string. The IDs for the string aabb is shown below.

$q_0aabb \vdash Xq_1abb \vdash Xaq_1bb \vdash Xq_2aYb \vdash q_2XaYb \vdash Xq_0aYb \vdash$

$XXq_1Yb \vdash XXYq_1b \vdash XXq_2YY \vdash Xq_2XYY \vdash XXq_0YY \vdash$

$XXYq_0Y \vdash XXYYq_0B \vdash XXYYBq_3$

$\delta(q_0, a)=q_1, X, R;$

$\delta(q_1, a)=q_1, a, R;$

$\delta(q_1, b)=q_2, Y, L;$

$\delta(q_2, a)=q_2, a, L;$

$\delta(q_2, X)=q_0, X, R;$

$\delta(q_2, Y)=q_2, Y, L;$

$\delta(q_0, Y)=q_0, Y, R;$

$\delta(q_0, B)=q_3, B, R;$

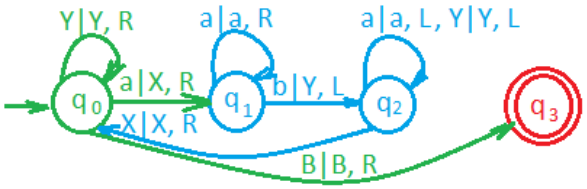


Figure 6.2: Turing machine for the string aabb.

$M_T=(Q, \sum, \delta, B, \Gamma, IS, FS)$

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

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

Γ = Tape symbol {a, b, c, B, X, Y}

B = Blank symbol.

δ: is defined above

$IS=q_0$

$FS=q_3$

Example 6.1: Design a Turing machine for the language $L=\{a^nb^nc^n : n\geq 1\}$

The processing of string aabbcc is shown below.

$q_0aabbcc \vdash Xq_1abbcc \vdash Xaq_1bbcc \vdash XaYq_2bcc \vdash XaYbq_2cc \vdash XaYq_3bZc \vdash$

$Xaq_3YbZc \vdash Xq_3aYbZc \vdash Xq_0aYbZc \vdash XXq_1YbZc \vdash XXYq_1bZc \vdash$

$XXYYq_2Zc \vdash XXYYZq_2c \vdash XXYYq_3ZZ \vdash XXYq_3YZZ \vdash XXq_3YYZZ \vdash$

$Xq_3XYYZZ \vdash XXq_0YYZZ \vdash XXYq_0YZZ \vdash XXYYq_0ZZ \vdash$

$XXYYZq_0Z \vdash XXYYZZq_0B \vdash XXYYZZBq_4$

$\delta(q_0, a)=q_1, X, R;$

$\delta(q_1, a)=q_1, a, R;$

$\delta(q_1, b)=q_2, Y, R;$

$\delta(q_2, b)=q_2, b, R;$

$\delta(q_2, c)=q_3, Z, L;$

$\delta(q_3, Z)=q_3, Z, L;$

$\delta(q_3, b)=q_3, b, L;$

$\delta(q_3, Y)=q_3, Y, L;$

$\delta(q_3, a)=q_3, a, L;$

$\delta(q_3, X)=q_0, X, R;$

$\delta(q_1, Y)=q_1, Y, R;$

$\delta(q_2, Z)=q_2, Z, R;$

$\delta(q_0, Y)=q_0, Y, R;$

$\delta(q_0, Z)=q_0, Z, R;$

$\delta(q_0, B)=q_4, B, R;$

$\delta(q_0, B)=q_3, B, R;$

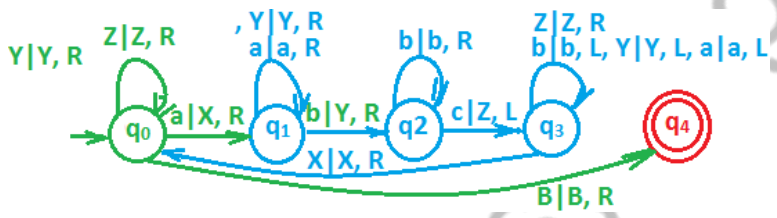


Figure 6.3: Turing machine for the language $L=\{a^n b^n c^n : n\geq 1\}$

$M_T=(Q, \sum, \delta, B, \Gamma, IS, FS)$

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

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

Γ = Tape symbol {a, b, c, B, X, Y, Z}

B = Blank symbol.

δ: is defined above

$IS=q_0$

$FS=q_4$

Example 6.2: Design a Turing machine for the language $L=\{ww^R : w=(0+1)^*\}$

The IDs for the string $\underbrace{001}_{w}\underbrace{100}_{w^R}$ is shown below.

$q_0 001100 \vdash Xq_1 01100 \vdash q_2 XY1100 \vdash Xq_0 Y1100 \vdash XYq_0 1100 \vdash$
$XYXq_1 100 \vdash XYq_2 XY00 \vdash XYXq_0 Y00 \vdash XYXYq_0 00 \vdash XYXYXq_1 0 \vdash$
$XYXYq_2 XY \vdash XYXYXq_0 Y \vdash XYXYXYq_0 B \vdash XYXYXYBq_3$

$\delta(q_0, 0)|\ \delta(q_0, 1)=q_1, X, R;$

$\delta(q_1, 0)|\ \delta(q_1, 1)=q_2, Y, L;$

$\delta(q_1, 1)=q_1, 1, R;$

$\delta(q_1, 0)=q_1, 0, R;$

$\delta(q_2, X)=q_0, X, R;$

$\delta(q_2, 0)=q_2, 0, L;$

$\delta(q_2, 1)=q_2, 1, L;$

$\delta(q_0, Y)=q_0, Y, R;$

$\delta(q_0, B)=q_3, B, R;$

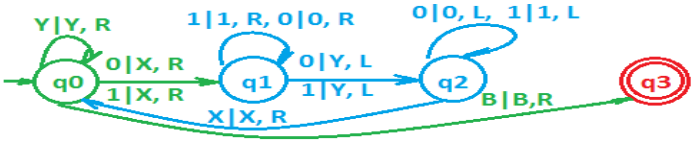


Figure 6.4: Turing machine for the language L={ww^R : w=(0+1)*}.

Example 6.3: Design a Turing machine for the language L= {ww : w=(0+1)*}.

The IDs for the string 001001 (w w) is shown below.

$q_0001001$ Xq_101001 $q_2XY1001$ Xq_0Y1001 XYq_01001
$XYXq_1001$ $XYX0q_101$ $XYX00q_11$ $XYX0q_20Y$ $XYXq_200Y$
XYq_2X00Y $XYXq_000Y$ $XYXXq_10Y$ $XYXq_2XYY$ $XYXXq_0YY$
$XYXXYq_0Y$ $XYXXYYq_0B$ $XYXXYYBq_3$

$\delta(q_0, 0)|\ \delta(q_0, 1)=q_1, X, R;$

$\delta(q_1, 0)|\ \delta(q_1, 1)=q_2, Y, L;$

$\delta(q_1, 1)=q_1, 1, R;$

$\delta(q_1, 0)=q_1, 0, R;$

$\delta(q_2, X)=q_0, X, R;$

$\delta(q_2, 0)=q_2, 0, L;$

$\delta(q_2, 1)=q_2, 1, L;$

$\delta(q_0, Y)=q_0, Y, R;$

$\delta(q_0, B)=q_3, B, R;$

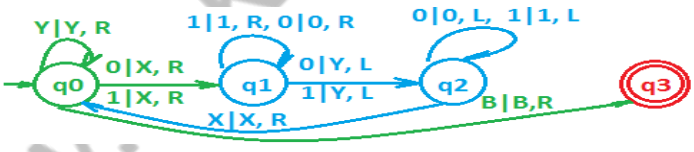


Figure 6.5: Turing machine for the language L= {ww : w=(0+1)*}.

Different types of Turing Machines

The Turing machine studied so far has simple one to understand the basic operation. By making use of this knowledge we can construct the complex Turing Machine. They are of

i. Multi-dimensional Turing Machines
ii. Multi-tape Turing Machines
iii. Non-deterministic Turing Machines

i. Multi-dimensional Turing Machines

In this case, Turing machine has multi-dimensional tape, with a read write head. It uses basic concept of Turing machine with modification of transition function as shown in figure 6.6.

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, U, D\}$,
Where *U* and *D* specifies movement of the read-write head up and down.

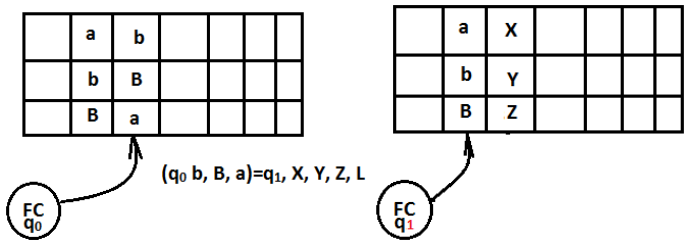


Figure 6.6: Multi-dimensional Turing Machine.

The basic Turing machine resembles like multi-dimensional Turing machine. It treats multiple symbols as single symbol, because all symbols are stored in a single cell as shown in figure 6.7.

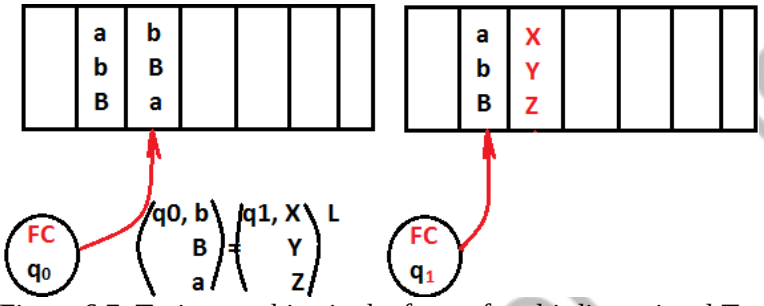


Figure 6.7: Turing machine in the form of multi-dimensional Turing Machines.

ii. **Multi-tape Turing Machines**

Turing machine has multiple tapes, each with its read write head. It uses basic concept of Turing machine with the modification of transition function as follows.

$Q \times \sum^n = Q, \Gamma^n, \{L, R\}^n$

For example, if n = 2, with a current configuration shown in Figure 6.8, then
$\delta(q_0, 0, 1) = (q_1, X, Y, L, R)$

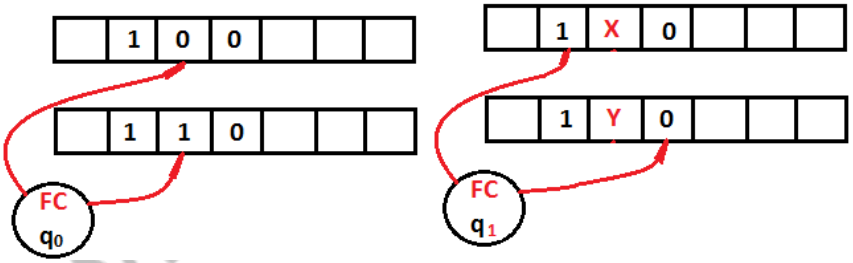


Figure 6.8: Multi-tape Turing Machines.

The transition rule can be applied only if the machine is in state q_0 and the first read-write head read an 0 and the second an 1. The symbol on the first tape will then be replaced with an X and its read-write head will move to the left. At the same time, the symbol on the second tape is rewritten as Y and the read-write head moves right. The control unit then changes its state to q_1 and the machine goes into the new configuration shown in Figure 6.8.

Multi-dimensional uses extra caps to work like a multi-tape. It scans first row from left to right, whenever it comes across symbol which has a cap, it stop scanning and changes the particular symbol to tape symbol and move right or left with the cap. Next it scans second row, same actions are repeated in the second row too as shown in figure 6.9.

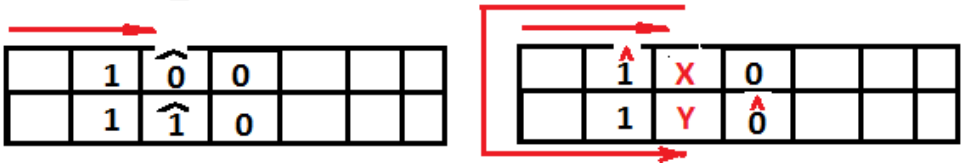


Figure 6.9: Multi-dimensional Turing machine in the form of multi-tape Turing machines.

iii. Non-deterministic Turing Machines

Non-deterministic Turing Machines is similar to deterministic Turing machine, except the transition function. The transition function is defined as

$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$

If a Turing machine has transitions specified by

$\delta (q_0, t) = \{(q_1, X, R), (q_2, Y, L)\}$,

in NPDA, the moves $q_0ttt \vdash Xq_1tt$ and $q_0ttt \vdash q_2BYtt$ are possible.