1

FINITE AUTOMATA

After going through this chapter, you should be able to understand :

- Alphabets, Strings and Languages
- Mathematical Induction
- Finite Automata
- Equivalence of NFA and DFA
- NFA with ϵ - moves

1.1 ALPHABETS, STRINGS & LANGUAGES

Alphabet

An alphabet, denoted by Σ , is a finite and nonempty set of symbols.

Example:

1. If Σ is an alphabet containing all the 26 characters used in English language, then Σ is finite and nonempty set, and $\Sigma = \{a, b, c, \ldots, z\}$.
2. $X = \{0,1\}$ is an alphabet.
3. $Y = \{1,2,3,\ldots\}$ is not an alphabet because it is infinite.
4. $Z = \{\ \}$ is not an alphabet because it is empty.

String

A string is a finite sequence of symbols from some alphabet.

Example :

"xyz" is a string over an alphabet $\Sigma = \{a, b, c, \ldots, z\}$. The empty string or null string is denoted by ϵ .

Length of a string

The length of a string is the number of symbols in that string. If w is a string then its length is denoted by $|w|$.

Example :

1. $w=abcd$, then length of w is $|w|= 4$
2. $n = 010$ is a string, then $|n| = 3$
3. ϵ is the empty string and has length zero.

The set of strings of length K (K ≥ 1)

Let Σ be an alphabet and $\Sigma = \{a, b\}$, then all strings of length K ($K \geq 1$) is denoted by Σ^K.

$\Sigma^K = \{w : w \text{ is a string of length } K, K \geq 1\}$

Example:

1. $\Sigma = \{a,b\}$, then

   $\Sigma^1 = \{a,b\}$,

   $\Sigma^2 = \{aa, ab, ba, bb\}$,

   $\Sigma^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

   $|\Sigma^1| = 2 = 2^1$ (Number of strings of length one),

   $|\Sigma^2| = 4 = 2^2$ (Number of strings of length two), and

   $|\Sigma^3| = 8 = 2^3$ (Number of strings of length three)
2. $S = \{0,1,2\}$, then $S^2 = \{00,01,02,11, 10,12,22,20,21\}$, and $|S^2| = 9 = 3^2$

Concatenation of strings

If w_1 and w_2 are two strings then concatenation of w_2 with w_1 is a string and it is denoted by w_1w_2. In other words, we can say that w_1 is followed by w_2 and $|w_1w_2| = |w_1| + |w_2|$.

Prefix of a string

A string obtained by removing zero or more trailing symbols is called prefix. For example, if a string $w = abc$, then a, ab, abc are prefixes of w.

Suffix of a string

A string obtained by removing zero or more leading symbols is called suffix. For example, if a string $w = abc$, then c, bc, abc are suffixes of w.
A string a is a proper prefix or suffix of a string w if and only if $a \neq w$.

Substrings of a string

A string obtained by removing a prefix and a suffix from string w is called substring of w. For example, if a string $w = abc$, then b is a substring of w. Every prefix and suffix of string w is a substring of w, but not every substring of w is a prefix or suffix of w. For every string w, both w and $\in$ are prefixes, suffixes, and substrings of w.
Substring of w = w – (one prefix) – (one suffix).

Language

A Language L over Σ, *is a subset of* Σ^*, *i. e., it is a collection of strings over the alphabet* Σ. ϕ, and $\{\in\}$ are languages. The language ϕ is undefined as similar to infinity and $\{\in\}$ is similar to an empty box i.e. a language without any string.

Example:

1. $L_1 = \{01, 0011, 000111\}$ is a language over alphabet $\{0,1\}$
2. $L_2 = \{\in, 0, 00, 000, ...\}$ is a language over alphabet $\{0\}$
3. $L_3 = \{0^n 1^n 2^n : n \geq 1\}$ is a language.

Kleene Closure of a Language

Let L be a language over some alphabet Σ. Then Kleene closure of L is denoted by L^* and it is also known as reflexive transitive closure, and defined as follows :

L^* = {*Set of all words over* Σ}
= {word of length zero, words of length one, words of length two, ...}

$$= \bigcup_{K=0}^{\infty} (\Sigma^K) = L^0 \cup L^1 \cup L^2 \cup \ldots.$$

Example:

1. $\Sigma = \{a, b\}$ and a language L over Σ. Then

   $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots.$

   $L^0 = \{\in\}$

   $L^1 = \{a, b\}$,

   $L^2 = \{aa, ab, ba, bb\}$ and so on.

   So, $L^* = \{\in, a, b, aa, ab, ba, bb ...\}$
2. $S = \{0\}$, then $S^* = \{\in, 0, 00, 000, 0000, 00000, ...\}$

Positive Closure

If Σ is an alphabet then positive closure of Σ is denoted by Σ^+ and defined as follows :

$\Sigma^+ = \Sigma^* - \{\in\}$ = {*Set of all words over* Σ *excluding empty string* $\in$}

Example :

if $\Sigma = \{0\}$, then $\Sigma^+ = \{0, 00, 000, 0000, 00000, ...\}$

1.2 MATHEMATICAL INDUCTION

Based on general observations specific truths can be identified by reasoning. This principle is called mathematical induction. The proof by mathematical induction involves four steps.

Basis : This is the starting point for an induction. Here, prove that the result is true for some n = 0 or 1.

Induction Hypothesis : Here, assume that the result is true for n = k .

Induction step : Prove that the result is true for some n = k + 1.

Proof of induction step : Actual proof.

Example : Prove the following series by principle of induction $1+2+3+.....+n=\frac{n(n+1)}{2}$

Solution :

Basis :

Let n = 1

L. H. S = 1 and R. H. S $=\frac{1(1+1)}{2}=1$

So the result is true for n = 1

Induction hypothesis :

By induction hypothesis we assume this result is true for n = k

i. e. $1+2+3+.....k=\frac{k(k+1)}{2}$

Inductive step :

We have to prove that the result is true for $n=k+1$

i. e. $1+2+3+.....+k+k+1=\frac{(k+1)(k+1+1)}{2}$

Proof of induction step :

L. H. S $=1+2+3+.....+k+k+1$

$=\frac{k\ (k+1)}{2}+k+1$

$=(k+1)\left(\frac{k}{2}+1\right)$

$=\frac{(k+1)\ (k+2)}{2}$

$=\frac{(k+1)\ (k+1+1)}{2}=R.H.S$

Hence the proof.

1.3 FINITE AUTOMATA (FA)

A finite automata consists of a finite memory called input tape, a finite - nonempty set of states, an input alphabet, a read - only head , a transition function which defines the change of configuration, an initial state, and a finite - non empty set of final states.

A model of finite automata is shown in figure 1.1.

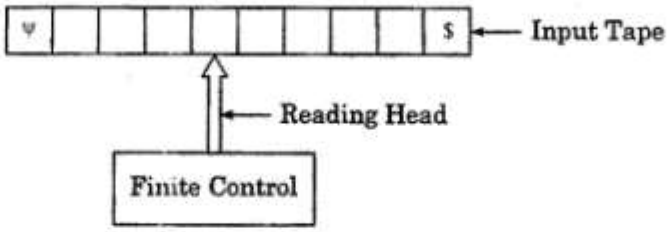


FIGURE 1.1 : Model of Finite Automata

The input tape is divided into cells and each cell contains one symbol from the input alphabet. The symbol 'ψ' is used at the leftmost cell and the symbol '$' is used at the rightmost cell to indicate the beginning and end of the input tape. The head reads one symbol on the input tape and finite control controls the next configuration. The head can read either from left - to - right or right - to -left one cell at a time. The head can't write and can't move backward. So , FA can't remember its previous read symbols. This is the major limitation of FA.

Deterministic Finite Automata (DFA)

A deterministic finite automata M can be described by 5 - tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is finite, nonempty set of states,
2. Σ is an input alphabet,
3. δ is transition function which maps $Q \times \Sigma \rightarrow Q$ i. e. the head reads a symbol in its present state and moves into next state.
4. $q_0 \in Q$, known as initial state
5. $F \subseteq Q$, known as set of final states.

Non - deterministic Finite Automata (NFA)

A non - deterministic finite automata M can be described by 5 - tuple $(Q, \Sigma, \delta, q_0, F)$, where

1 Q is finite, nonempty set of states,
2. Σ is an input alphabet,
3. δ is transition function which maps $Q \times \Sigma \rightarrow 2^Q$ i. e., the head reads a symbol in its present state and moves into the set of next state (s). 2^Q is power set of Q,
4. $q_0 \in Q$, known as initial state, and
5. $F \subseteq Q$, known as set of final states.

The difference between a DFA and a NFA is only in transition function. In DFA, transition function maps on at most one state and in NFA transition function maps on at least one state for a valid input symbol.

States of the FA

FA has following states :

1. **Initial state :** Initial state is an unique state ; from this state the processing starts.
2. **Final states :** These are special states in which if execution of input string is ended then execution is known as successful otherwise unsuccessful.
3. **Non - final states :** All states except final states are known as non - final states.
4. **Hang - states :** These are the states, which are not included into Q, and after reaching these states FA sits in idle situation. These have no outgoing edge. These states are generally denoted by ϕ. For example, consider a FA shown in figure1.2.

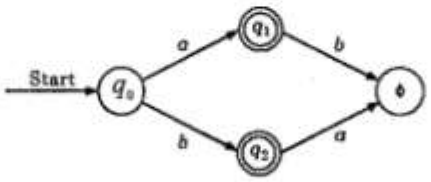


FIGURE 1.2 : Finite Automata

q_0 is the initial state, q_1, q_2 are final states, and ϕ is the hang state.

Notations used for representing FA

We represent a FA by describing all the five - terms $(Q, \Sigma, \delta, q_0, F)$. By using diagram to represent FA make things much clearer and readable. We use following notations for representing the FA :

1. The initial state is represented by a state within a circle and an arrow entering into circle as shown below :

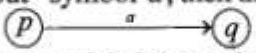
(Initial state q_0)

2. Final state is represented by final state within double circles :

(Final state q_f)

3. The hang state is represented by the symbol 'ϕ' within a circle as follows :

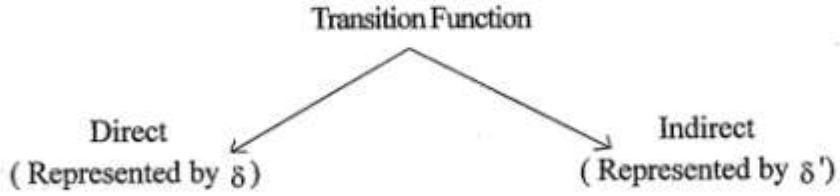
4. Other states are represented by the state name within a circle.
5. A directed edge with label shows the transition (or move). Suppose p is the present state and q is the next state on input - symbol 'a', then this is represented by



6. A directed edge with more than one label shows the transitions (or moves). Suppose p is the present state and q is the next state on input - symbols 'a_1' or 'a_2' or ... or 'a_n' then this is represented by

Transition Functions

We have two types of transition functions depending on the number of arguments.



Direct transition Function (δ)

When the input is a symbol, transition function is known as direct transition function.

Example : $\delta(p, a) = q$ (Where p is present state and q is the next state).
It is also known as one step transition.

Indirect transition function (δ')

When the input is a string, then transition function is known as indirect transition function.

Example : $\delta'(p, w) = q$, where p is the present state and q is the next state after | w | transitions. It is also known as one step or more than one step transition.

Properties of Transition Functions

1. If $\delta(p, a) = q$, then δ (p, ax) = δ(q, x) and if δ' (p, x) = q, then δ' (p, xa) = δ'(q, a)
2. For two strings x and y ; $\delta(p, xy) = \delta(\delta(p, x), y)$, and $\delta'(p, xy) = \delta'(\delta'(p, x), y)$

Example :1. A DFA $M = (\{q_0, q_1, q_2, q_f\}, \{0,1\}, \delta, q_0, \{q_f\})$ is shown in figure1.3 .

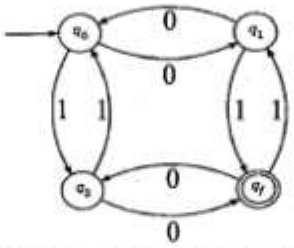


FIGURE 1.3 : Deterministic finite automata

Where δ is defined as follows :

	0	1
→ q_0	q_1	q_2
q_1	q_0	q_f
q_2	q_f	q_0
q_f	q_2	q_1

2. A NFA $M_1 = (\{q_0, q_1, q_2, q_f\}, \{0,1\}, \delta, q_0, \{q_f\})$ is shown in figure1.4.

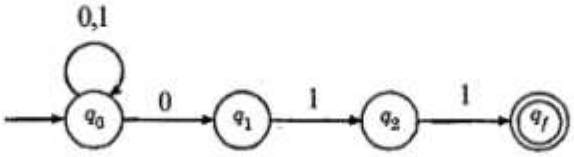


FIGURE 1.4 : Non - deterministic finite automata

Transition function δ is defined as follows :

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	-	$\{q_2\}$
q_2	-	$\{q_f\}$
q_f	-	$\{q_f\}$

Note : In first row of transition table, when present state is q_0 and input is '0', then there are two next states q_0 , and q_1 .

Acceptability of a string by DFA : Let a DFA $M = (Q, \Sigma, \delta, q_0, F)$ and an input string $w \in \Sigma^*$. The string w is accepted by M if and only if $\delta(q_0, w) = q_f$, where $q_f \in F$.

When w is accepted by M, then the execution of string w ends in a final state and this execution is known as successful otherwise unsuccessful .

Example : Consider the DFA shown in figure1.5.

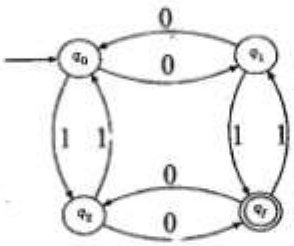


FIGURE 1.5 : Deterministic finite automata

Input strings are :

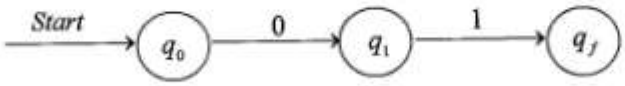
i) 01,

ii) 011

Check the acceptability of each string.

Solution :

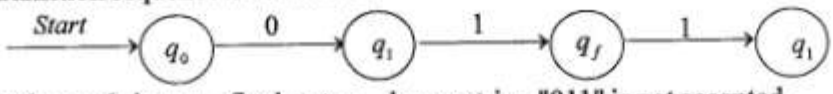
1. Let the input string $w_1 = 01$, the transition sequence is as follows :



Execution ends in final state q_f, hence string "01" is accepted.

2. Let input string $w_2 = 011$

The transition sequence is as follows :



Execution ends in non - final state q_1, hence string "011" is not accepted.

Acceptability of a string by NFA

Let a NFA $M = (Q, \Sigma, \delta, q_0, F)$ and an input string $w \in \Sigma^*$. The string w is accepted by M if and only if $\delta(q_0, w) = \{ q_i : q_i \in F$, for some i = 0, 1, ..., n $\}$.

When w is accepted by M, then the execution of string w ends in some final state and the execution is known as successful otherwise unsuccessful .

Example : Consider the NFA shown in figure1.6.

Check the acceptability of following strings : i) 011 ii) 010 iii) 011011

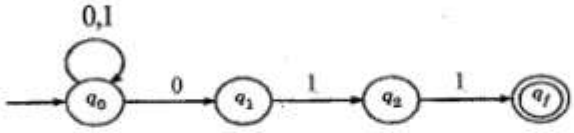
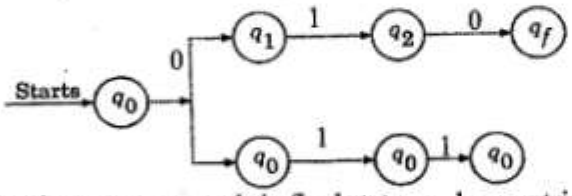


FIGURE 1.6 : Non - deterministic finite automata

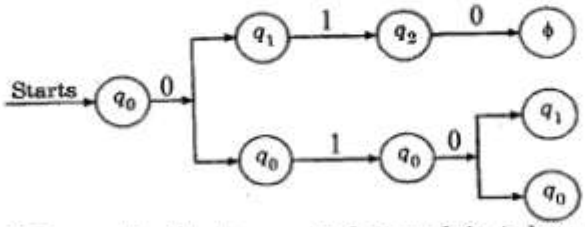
Solution :

1. Transition sequence for the string "011" is as follows :



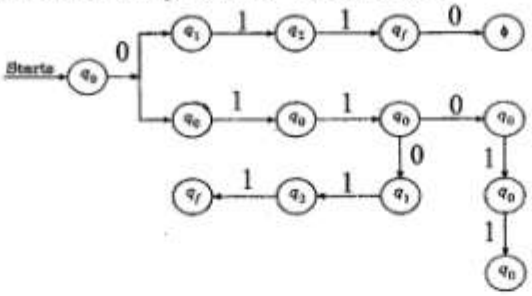
One execution sequence ends in final state q_f, hence string "011" is accepted.

2. Transition sequence for the string "010" is as follows :



The execution ends in non - final states q_0, q_1 and one ends in ϕ, hence string "010" is not accepted.

3. Transition sequence for the string "011011" is as follows :



One execution ends in hang state ϕ, second ends in non - final state q_0, and third ends in final state q_f hence string "011011" is accepted by third execution.

Difference between DFA and NFA

Strictly speaking the difference between DFA and NFA lies only in the definition of δ. Using this difference some more points can be derived and can be written as shown :

DFA	NFA
1. The DFA is 5 - tuple or quintuple $M=(Q,\Sigma,\delta,q_0,F)$ where Q is set of finite states Σ is set of input alphabets $\delta: Q\times\Sigma$ to Q q_0 is the initial state $F\subseteq Q$ is set of final states	The NFA is same as DFA except in the definition of δ. Here, δ is defined as follows : $\delta: Q\times(\Sigma\cup\in)$ to subset of 2^Q
2. There can be zero or one transition from a state on an input symbol	There can be zero, one or more transitions from a state on an input symbol
3. No $\in$ – transitions exist i.e., there should not be any transition or a transition if exist it should be on an input symbol	$\in$ – transitions can exist i. e., without any input there can be transition from one state to another state.
4. Difficult to construct	Easy to construct

Example 1 : Consider the FA shown in below figure. Check the acceptability of following strings:

(a) 0101 (b) 0111 (c) 001

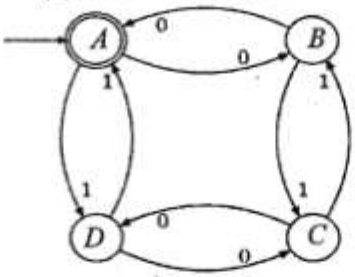
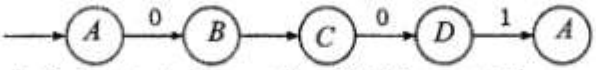


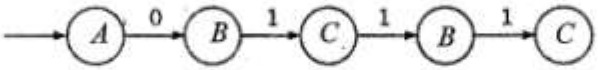
FIGURE : Finite automata

Solution : (a) The transition sequence for input string 0111 is following :



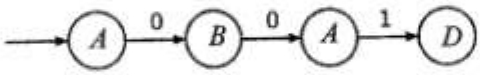
Execution ends in final state A, hence string 0101 is accepted.

(b) The transition sequence for input string 0111 is as follows :



Execution ends in non-final state C, hence string 0111 is not accepted.

(c) The transition sequence for input string 001 is as follows :



Execution ends in non-final state D, hence string 001 is not accepted

Example 2 : Let a DFA $M = (Q, \Sigma, \delta, q_0, F)$ is shown in below figure.

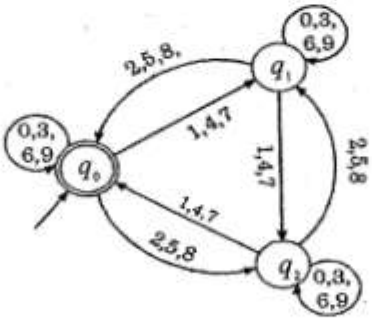
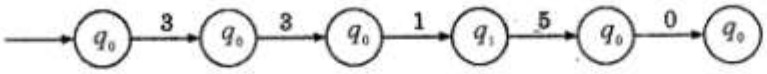


FIGURE : D F A

Check that string 33150 is recognized by above DFA or not ?

Solution :

For string 33150 the transition sequence is as follows :



Since, transition ends in final state, q_0, so string 33150 is recognized.

Example 3 : Consider below transition diagram and verify whether the following strings will be accepted or not ? Explain.

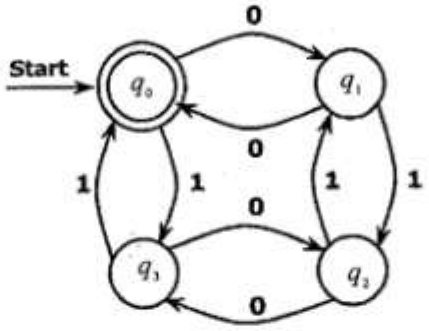


FIGURE : Given Transition Diagram

i) 0011 ii) 010101 iii) 111100 iv) 1011101 .

Solution : Transition table for the given diagram is ,

		0	1
→	(q_0)	q_1	q_3
	q_1	q_0	q_2
	q_2	q_3	q_1
	q_3	q_2	q_0

TABLE : Transition Table for the given Transition Diagram

i) 0011

$\delta(q_0,0011) \vdash \delta(q_1,011)$
$\vdash \delta(q_0,11)$
$\vdash \delta(q_3,1)$
$\vdash q_0$

$\therefore$ 0011 is accepted.

ii) 010101

$\delta(q_0,010101) \vdash \delta(q_1,10101)$
$\vdash \delta(q_2,0101)$
$\vdash \delta(q_3,101)$
$\vdash \delta(q_0,01)$
$\vdash \delta(q_1,1)$
$\vdash q_2$

$\therefore$ 010101 is not accepted.

iii) 111100

$\delta(q_0,111100) \vdash \delta(q_3,11100)$
$\vdash (q_0,1100)$
$\vdash \delta(q_3,100)$
$\vdash \delta(q_0,00)$
$\vdash \delta(q_1,0)$
$\vdash q_0$

$\therefore$ 111100 is accepted.

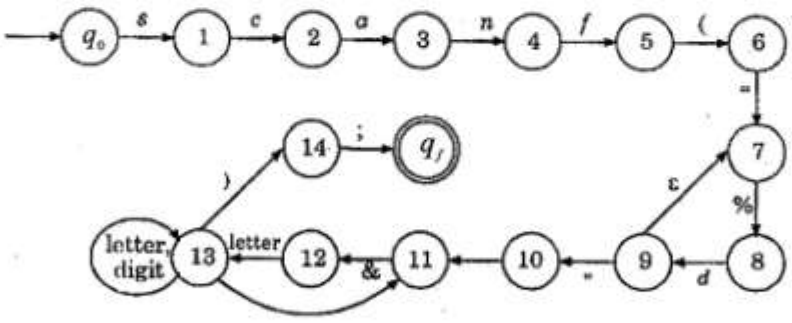
iv) 1011101

$\delta(q_0,1011101) \vdash \delta(q_3,011101)$
$\vdash \delta(q_2,11101)$
$\vdash \delta(q_1,1101)$
$\vdash \delta(q_2,101)$
$\vdash \delta(q_1,01)$
$\vdash \delta(q_0,1)$
$\vdash q_3$

$\therefore$ 1011101 is not accepted.

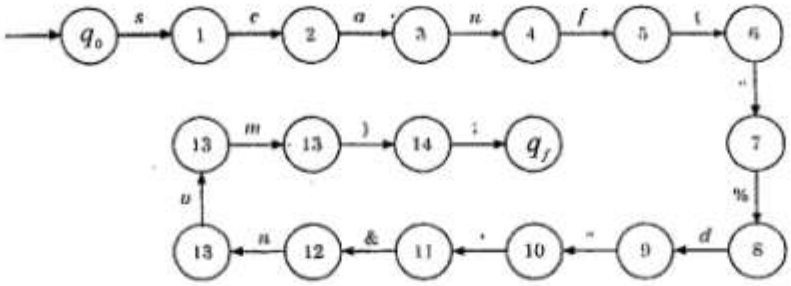
Example 4 : Consider the NFA shown in below figure. Check the acceptability of following string

scanf ("%d", & num) ;



Note : Letter stands for any symbol from { a, b, ... , z } and digit stands for any digit from { 0, 1, 2, ... 9 } .

Solution : The transition sequence for given string : **scanf("%d", & num) ;**

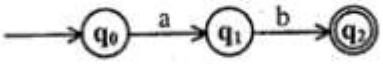


Since, execution of given string ends in final state q_f, so the string is recognized.

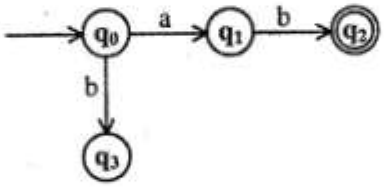
Example 5 : Obtain a DFA to accept strings of a's and b's starting with the string ab .

Solution :

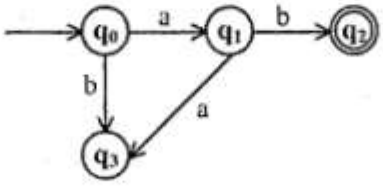
From the problem it is clear that the string should start with ab and so, the minimum string that can be accepted by the machine is ab. To accept the string ab, we need three states and the machine can be written as



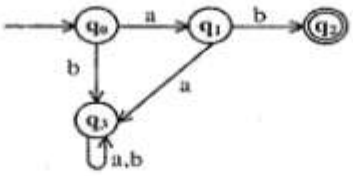
where q_2 is the final or accepting state. In state q_0, if the input symbol is b, the machine should reject b (note the string should start with a) . So, in state q_0, on input b, we enter into the rejecting state q_3. The machine for this can be of the form



The machine will be in state q_1, if the first input symbol is a. If this a is followed by another a, the string aa should be rejected by the machine . So, in state q_1, if the input symbol is a , we reject it and enter into q_3 which is the rejecting state. The machine for this can be of the form



Whenever the string is not starting with ab, the machine will be in state q_3 which is the rejecting state. So, in state q_3, if the input string consists of a's and b's of any length, the entire string can be rejected and can stay in state q_3 only. The resulting machine can be of the form



The machine will be in state q_2, if the input string starts with ab. After the string ab, the string containing any combination of a's and b's, can be accepted and so remain in state q_2 only. The complete machine to accept the strings of a's and b's starting with the string ab is shown in below figure. The state q_3 is called dead state or trap state or rejecting state.

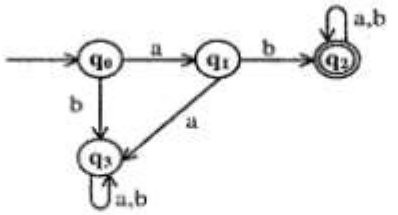


FIGURE : Transition diagram to accept string ab (a + b)*

So, the DFA which accepts strings of a's and b's starting with the string ab is given by $M = (Q, \Sigma, \delta, q_0, F)$

where $Q = \{q_0, q_1, q_2\}$; $\Sigma = \{a, b\}$;

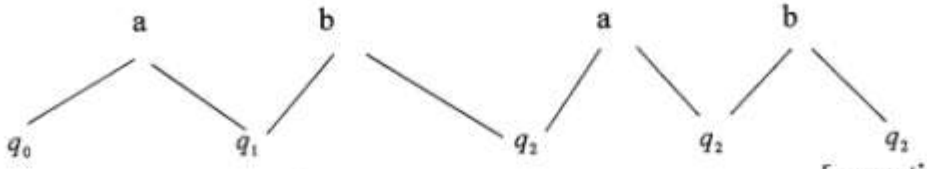
q_0 is the start state; $F = \{q_2\}$

δ is shown the transition table.

		← Σ →	
	δ	a	b
↑	$\rightarrow q_0$	q_1	q_3
States	q_1	q_3	q_2
	(q_2)	q_2	q_2
↓	q_3	q_3	q_3

TABLE : Transition table for DFA shown in above figure

To accept the string abab : This string is accepted by the machine and is evident from the below figure.

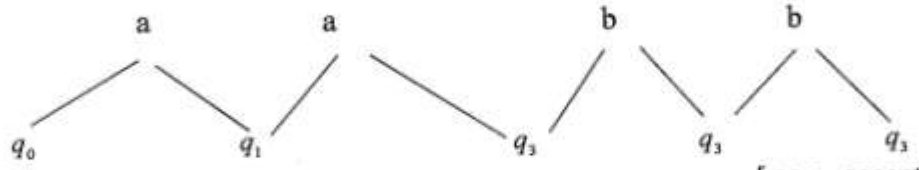


[accepting state]

FIGURE : To accept the string abab

Here, $\delta^*(q_0, abab) = q_2$ which is the final state. So, the string abab is accepted by the machine.

To reject the string aabb : The string is rejected by the machine and is evident from the below figure.



[non - accepting state]

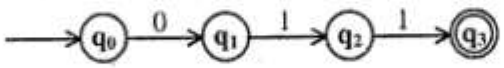
FIGURE : To reject the string aabb

Here, $\delta^*(q_0, aabb) = q_3$ which is not an accepting state. So, the string aabb is rejected by the machine.

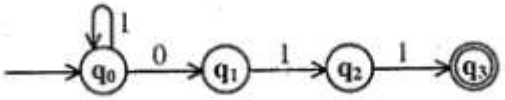
Example 6 : Draw a DFA to accept string of 0's and 1's ending with the string 011.

Solution :

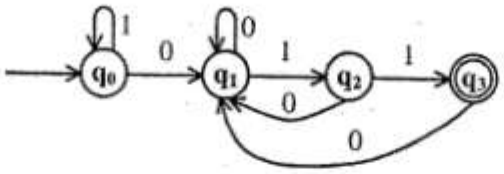
The minimum string that can be accepted by the machine is 011. It requires four states with q_0 as the start state and q_3 as the final state as shown below.



In state q_0, suppose we input the string 1111 ..... 011. Since the string ends with 011, the entire string has to be accepted by the machine. To accept the string 011 finally, the machine should be in state q_0. So, on any number of 1's the machine stays only in state q_0 and if the string ends with 011, the machine enters into the final state. The machine can be of the form



If the machine is in any of the states q_1, q_2 and q_3 and if the current input symbol is 0 and if the next input string is 11, the entire string should be accepted. This is because the string ends with 011. So, from all these states on the input symbol 0, there should be a transition to state q_1 so that if we enter the string 11 we can reach the final state. Now the machine can take the form as shown below.



In state q_3, if the input symbol is 1, enter into state q_0 so that if the next input string is 011, we can enter into the final state q_3. So, the final machine which accepts a string of 0's and 1's ending with the string 011 can take the following form.

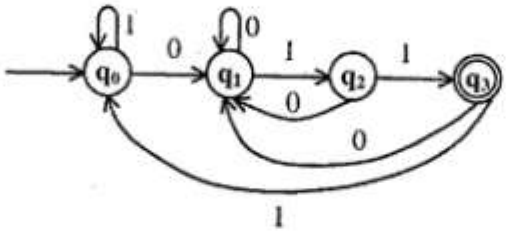


FIGURE : transition diagram to accept (0+1) *011

So, the DFA which accepts strings of 0's and 1's ending with the string 011 is given by $M = (Q, \Sigma, \delta, q_0, F)$ where

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

q_0 is the start state; $F = \{q_3\}$;

δ is shown using the transition table.

		← Σ →	
		0	1
↑ States ↓	→ q_0	q_1	q_0
	q_1	q_1	q_2
	q_2	q_1	q_3
	(q_3)	q_1	q_0

TABLE : Transition table for the machine shown in above figure

To accept the string 0011 : This string is accepted by the machine and is evident from the below figure . Here, $\delta^*(q_0,0011)= q_3$ which is the final state. So, the string 0011 is accepted by the machine.

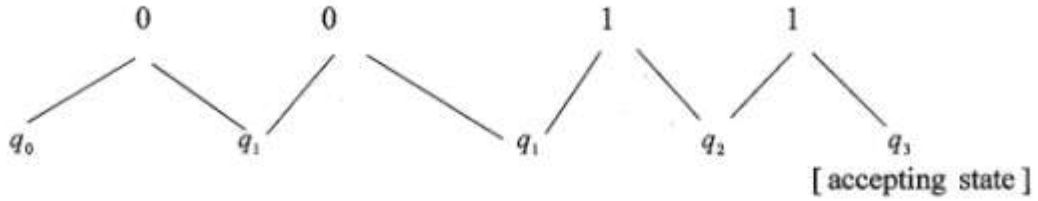


FIGURE : To accept the string 0011

To reject the string 0101 : The string is rejected by the machine and is evident from the below figure.

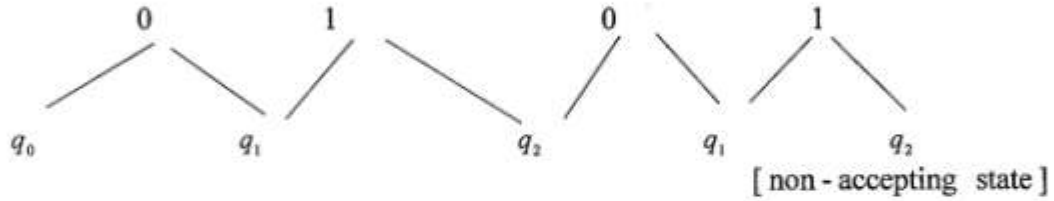


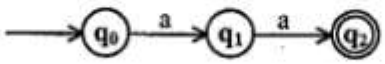
FIGURE : To reject the string 0101

Here, $\delta^*(q_0,0101)= q_2$ which is not an accepting state. So, the string 0101 is rejected by the machine.

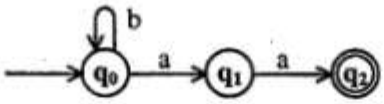
Example 7 : Obtain a DFA to accept strings of a's and b's having a substring aa .

Solution :

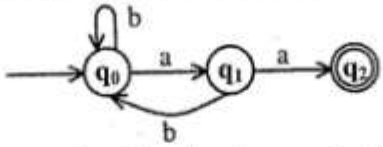
The minimum string that can be accepted by the machine is aa. To accept exactly two symbols, the DFA requires 3 states and the machine to accept the string aa can take the form



where q_0 is the start state and q_2 is the accepting state. In state q_0, if the input symbol is b, stay in q_0 so that when any number of b's ends with aa, the entire string is accepted. The machine for this can be of the form



There is a transition to state q_1 on input symbol a. In state q_1, if the input symbol is b, there will be a transition to state q_0 so that if this b is followed by aa, the machine enters into state q_2 so that the entire string is accepted by the machine. The transition diagram for this can be of the form



The machine enters into state q_2 when the string has a sub string aa. So, in this state even if we input any number of a's and b's the entire string has to be accepted. So, the machine should stay in q_2. The final machine which accepts strings of a's and b's having a sub string aa is shown in below figure

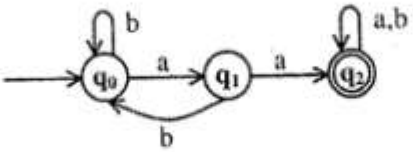


FIGURE : transition diagram to accept (a+b)* aa(a+b)*

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

$Q = \{q_0, q_1, q_2\}$; $\Sigma = \{a, b\}$

q_0 is the start state; $F = \{q_2\}$

δ is shown using the transition table.

	δ	$\leftarrow \Sigma \rightarrow$ a	b
↑ States ↓	$\rightarrow q_0$	q_1	q_0
	q_1	q_2	q_0
	(q_2)	q_2	q_2

TABLE : Transition table for the machine shown in above figure

To accept the string baab : This string is accepted by the machine and is evident from the below figure.

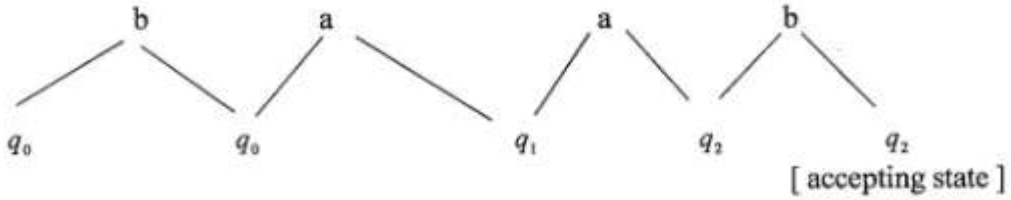


FIGURE : To accept the string baab

Here, $\delta^*(q_0, baab) = q_2$ which is the final state. So, the string baab is accepted by the machine. The string baba is rejected by the machine and is evident from the below figure.

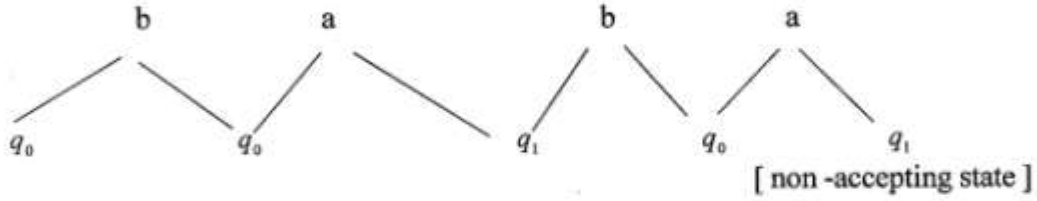


FIGURE : To reject the string baba

Here, $\delta^*(q_0, baba) = q_1$ which is not an accepting state. So, the string baba is rejected by the machine.

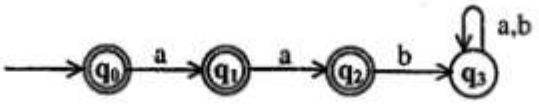
Example 8 : Obtain a DFA to accept strings of a's and b's except those containing the substring aab.

Solution :

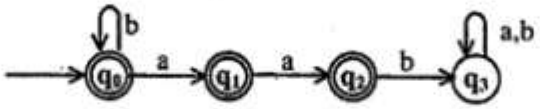
Note : This can be solved in two ways. The first method is similar to the previous problem i. e., draw a DFA to accept strings of a's and b's having a substring aab. Then change the final states to non - final states and non final states to final states. The resulting machine will accept the strings of a's and b's except those containing the sub - string aab.

Here, the second method is explained. The minimum string that can be rejected by the machine is aab. To reject this string we need four states q_0, q_1, q_2 and q_3. Since the string aab has to be rejected, q_3 can not be the final state and the rest of the states will be the final states as shown below.

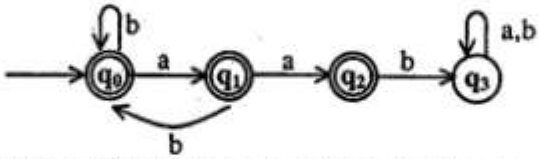
The machine enters into q_3 if the string has a sub string aab. In this state if we input any number of a's or / and b's, the entire string has to be rejected. So, stay in the state q_3 only. The machine for this is shown below.



In state q_0, if the input symbol is b, stay in q_0 so that if this b is followed by aab, the machine enters into state q_3 so that the string is rejected. The machine for this is shown below.



In state q_1, if the input symbol is b, enter into state q_0, so that if this b ends with the string aab, the entire string is rejected. The machine for this is shown below.



The machine will be in state q_2 if the string ends with aa. At this stage, if the input symbol is a, again the string ends with aa and so stay in state q_2 only. The complete machine to accept strings of a's and b's except those containing the sub string aab is shown below.

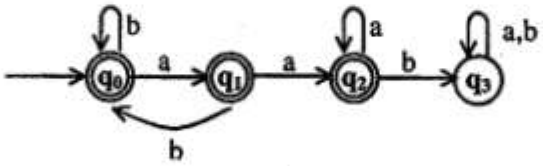


FIGURE : DFA to accept the string except the sub string aab.

So, the DFA $M = (Q, \Sigma, \delta, q_0, F)$ where

$Q = \{q_0, q_1, q_2, q_3\}$; $\Sigma = \{a, b\}$

q_0 is the start state; $F = \{q_0, q_1, q_2\}$

δ is shown using the transition table

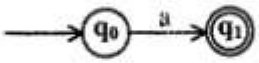
States	δ	a	b
↑	$\rightarrow q_0$	q_1	q_0
	q_1	q_2	q_0
	q_2	q_2	q_3
↓	q_3	q_3	q_3

TABLE : Transition table

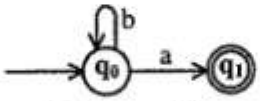
Example 9 : Obtain a DFA to accept strings of a's and b's having exactly one a, atleast one a, not more than three a's.

Solution :

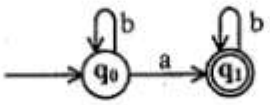
To accept exactly one a : To accept exactly one a, we need two states q_0 and q_1 and make q_1 as the final state. The machine to accept one a is shown below.



In q_0, on input symbol b, remain q_0 only so that any number of b's can end with one a. The machine for this can be of the form



In state q_1, on input symbol b remain in q_1 and the machine can take the form



But, in state q_1, if the input symbol is a, the string has to be rejected as the machine can have any number of b's but exactly one a. So, the string has to be rejected and we enter into a trap state q_2. Once the machine enters into trap state, there is no way to come out of the state and the string is rejected by the machine. The complete machine is shown in below figure.

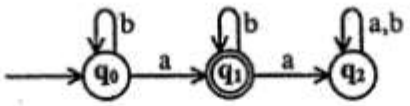


FIGURE : DFA to accept exactly one a

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

$Q = \{q_0, q_1, q_2\}$; $\Sigma = \{a, b\}$

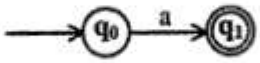
q_0 is the start state; $F = \{q_1\}$

δ is shown below using the transition table.

		$\leftarrow \Sigma \rightarrow$	
	δ	a	b
States $\uparrow$	$\rightarrow q_0$	q_1	q_0
	(q_1)	q_2	q_1
$\downarrow$	q_2	q_2	q_2

TABLE : Transition table

The machine to accept at least one a : The minimum string that can be accepted by the machine is a. For this, we need two states q_0 and q_1 where q_1 is the final state. The machine for this is shown below.



In state q_0, if the input symbol is b, remain in q_0. Once the final state q_1 is reached, whether the input symbol is a or b, the entire string has to be accepted. The machine to accept at least one a is shown in below figure.

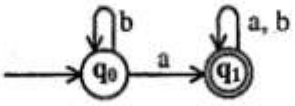


FIGURE : DFA to atleat one a

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

$Q = \{q_0, q_1\}$; $\Sigma = \{a, b\}$

q_0 is the start state ; $F = \{q_1\}$

δ is shown using the transition table .

	← Σ →	
δ	a	b
→ q_0	q_1	q_0
(q_1)	q_1	q_1

TABLE : Transition table

The machine to accept not more than three a's : The machine should accept not more than three a's means

- It can accept zero a's i. e., no a's
- It can accept one a
- It can accept two a's
- It can accept 3 a's
- But, it can not accept more than three a's.

In this machine maximum of three a's can be accepted i. e., the machine can accept zero a's, one a, two a's or three a's. So, we need maximum four states q_0, q_1, q_2 and q_3 where all these states are final states and q_0 is the start state. The machine can take the form

In state q_3, if the input symbol is a, the string has to be rejected and we enter into a trap state q_4. Once this trap state is reached, whether the input symbol is a or b, the entire string has to be rejected and remain in state q_4. Now, the machine can take the form as shown below.

In state q_0, q_1, q_2 and q_3, if the input symbol is b, stay in their respective states and the final transition diagram is shown in below figure.

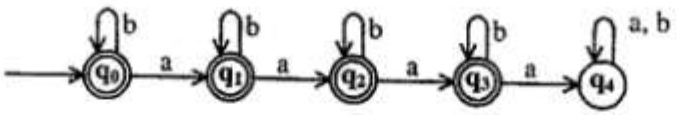


FIGURE : DFA to accept not more than 3 a's

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

$Q = \{q_0, q_1, q_2, q_3, q_4\}$; $\Sigma = \{a, b\}$

q_0 is the start state; $F = \{q_0, q_1, q_2, q_3\}$

δ is shown using the transition table.

	← Σ →	
δ	a	b
→q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_4	q_3
q_4	q_4	q_4

TABLE : Transition table for DFA shown in above figure

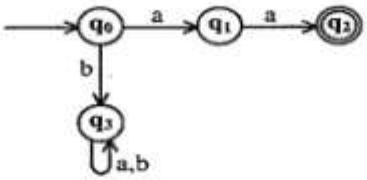
Example 10 : Obtain a DFA to accept the language $L = \{awa \mid w \in (a+b)^*\}$.

Solution :

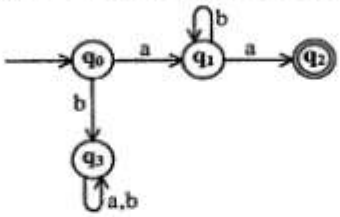
Here, $w \in (a+b)^*$ indicates the string consisting of a's and b's of any length including the null string. So, the language accepted by DFA is a string which starts with a, followed by a string of a's and b's (possibly including $\in$) of any length and followed by one a.

If w is $\in$ (null string), the minimum string that can be accepted by the machine is aa and so, we need three states q_0, q_1 and q_2 to accept the string. The machine can be of the form

where q_0 is the start state and q_2 is the final state. In state q_0, if the input symbol is b, the string has to be rejected and so, we enter into a trap state q_3. Once the machine enters into trap state, whether the input is either a or b, the string has to be rejected and the machine for this is shown below.



In state q_1, if the input symbol is b, remain in q_1 and the machine takes the form



In state q_2, if the input symbol is a, the string ends with a and so remain in q_2. In state q_2, if the input symbol is b, enter into state q_1 so that after inputting the symbol a, the machine enters into q_2. The complete machine is shown in below figure.

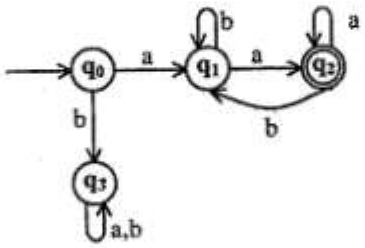


FIGURE : DFA to accept awa

So, the machine $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{q_0, q_1, q_2, q_3\}$; $\Sigma = \{a, b\}$

q_0 is the start state ; $F = \{q_2\}$

δ is shown using the transition table .

	← Σ →	
δ	a	b
$\rightarrow q_0$	q_1	q_3
q_1	q_2	q_1
(q_2)	q_2	q_1
q_3	q_3	q_3

TABLE : Transition table for DFA shown in above figure

Example 11 : Obtain a DFA to accept even number of a's, odd number of a's .

Solution :

The machine to accept even number of a's is shown in figure (a) and odd number of a's is shown in figure(b).

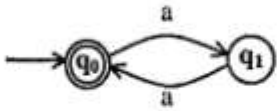


Figure : (a)

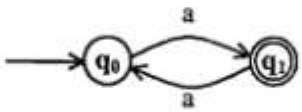


Figure : (b)

Example 12 : Obtain a DFA to accept strings of a's and b's having even number of a's and b's.

Solution :

The machine to accept even number of a's and b's is shown in figure 1.

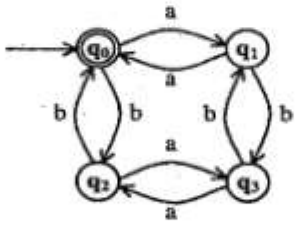


FIGURE 1 : DFA to accept even no. of a's and b's

Note : In the DFA shown in figure 1, instead of making q_0 as the final state, make q_2 as the final state. The DFA to accept even number of a's and odd number of b's is obtained and is shown in figure 2.

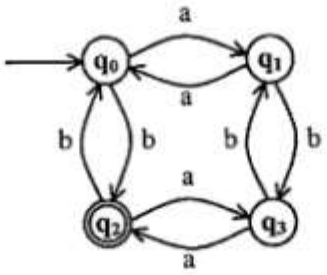


FIGURE 2 : DFA to accept even no. of a's and odd number of b's

Note : In the DFA shown in figure 1, instead of making q_0 as the final state, make q_1 as the final state. The DFA to accept odd number of a's and even number of b's is obtained and is shown in figure 3 .

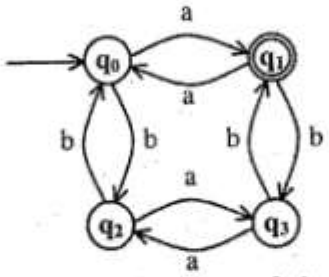


FIGURE 3 : DFA to accept odd no. of a's and even number of b's

Note : In the DFA shown in figure 1, instead of making q_0 as the final state, make q_3 as the final state. The DFA to accept odd number of a's and odd number of b's is obtained and is shown in figure 4.

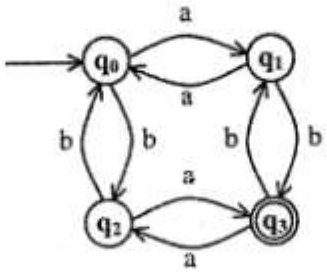


FIGURE 4 : DFA to accept odd no. of a's and odd number of b's

Example 13 : Design a DFA, M that accepts the language $L(M) = \{ w | w \in \{a, b\}^* \}$ and w does not contain 3 consecutive b's.

Solution :

We first consider a language $L_1(M) = \{ w | w \in \{a, b\}^* \}$ and w contain 3 consecutive b's.

Then DFA for L_1 is,

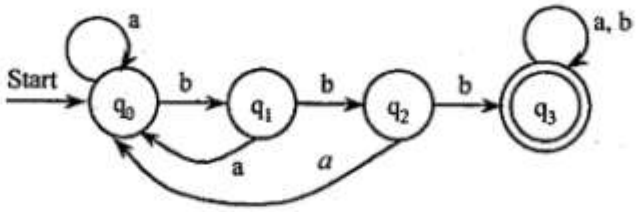


FIGURE : (A)

Now we can get language L(M) by converting non - final states to final states and final states to non - final states.

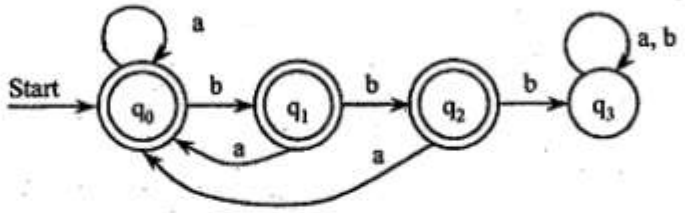


FIGURE : (B)

FIGURE : Construction of DFA from the language $L = \{ w | w \in \{a, b\}^* \}$

Example 14 : Design DFA which accepts language L = { 0, 000, 00000, ... } over { 0 }.

Solution : L = { 0, 000, 00000, ... } over { 0 } means L accepts the strings of odd number of 0's. So the DFA for L is ,

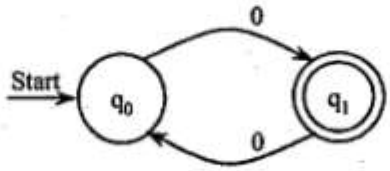
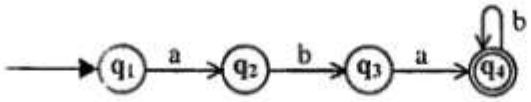


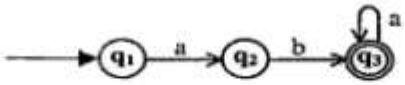
FIGURE : DFA for the given language L.

Example 15 : Obtain an NFA to accept the following language $L = \{ w | w \in abab^n$ or aba^n where $n \geq 0 \}$

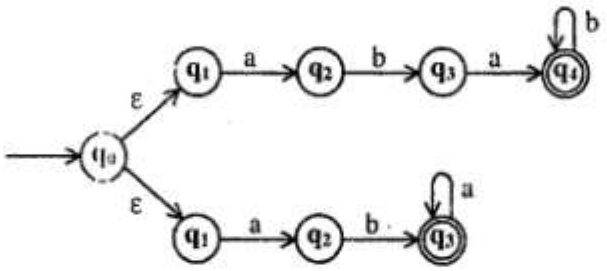
Solution : The machine to accept $abab^n$ where $n \geq 0$ is shown below :



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



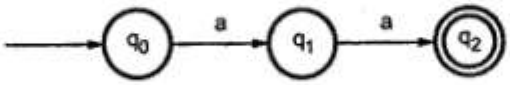
The machine to accept either $abab^n$ or aba^n where $n \geq 0$ is shown below :



Example 16 : Design NFA to accept strings with a's and b's such that the string end with 'aa'.

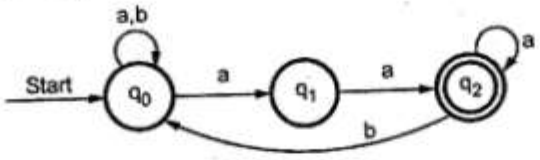
Solution :

Method - I : The simple FA which accepts a string with 'aa ' is



Now there can be a situation where in

Hence we can design a required NFA as



It can be denoted by ,

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

We can test some strings for above drawn NFA.

Consider

$$\begin{aligned}\delta(q_0, aaa) &\vdash \delta(q_0, aa)\\ &\vdash \delta(q_1, a)\\ &\vdash \delta(q_2, \in)\end{aligned}$$

i. e. we reach to final state.

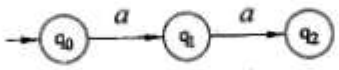
$$\begin{aligned}\delta(q_0, aaa) &\vdash \delta(q_0, aa)\\ &\vdash \delta(q_0, a)\\ &\vdash \delta(q_1, \in)\end{aligned}$$

i.e. we are not in final state.

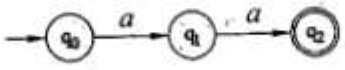
Thus there are two possibilities by which we move with string 'aaa' in above given NFA.

Method - II

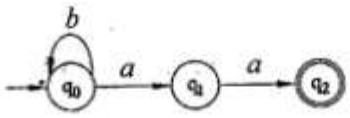
Start with two consecutive a's initially. It requires three states q_0, q_1 and q_2 respectively. Consider q_0 as the initial state



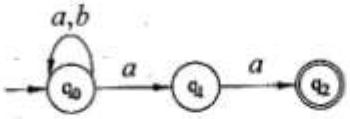
Assign q_2 as final state so that it accepts two consecutive a's



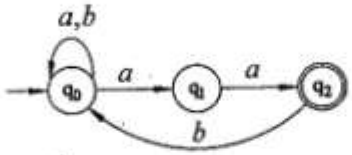
Design such a way if any number of b's preceeds first a it should be in the same state i.e., in the state q_0.



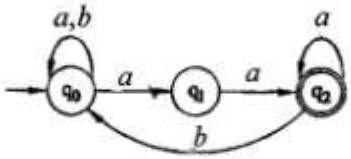
Design such a way if a's preceed by first a it should move from q_0 to q_0 only i. e., it will be in the state.



After the second a, b comes it has to move from q_2 to q_0.



Any number of a's followed by second a then it will be in the same state q_2.



The transition table is

	a	b
q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	ϕ
q_2	q_2	q_1

Test for the strings which ends with two consecutive a's.

String baa :

$\delta(q_0,baa) \vdash \delta(q_0,aa)$

$\vdash \delta(q_1,a)$

$\vdash \delta(q_2,\in)$

$\vdash q_2 \in F$

String baa :

$\delta(q_0,baa) \vdash \delta(q_0,aa)$

$\vdash \delta(q_0,a)$

$\vdash \delta(q_1,\in)$

$\vdash q_1 \notin F$

$\therefore$ NFA and two possibilities for the same input also shown.

String aab :

$$\begin{aligned}\delta(q_0, aab) &\vdash \delta(q_1, ab)\\ &\vdash \delta(q_2, b)\\ &\vdash \delta(q_1, \in)\\ &\vdash q_1 \notin F\end{aligned}$$

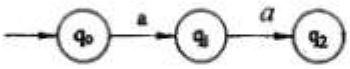
$\therefore$ If the string is not ending with two consecutive a's it will not be accepted.

String aaa :

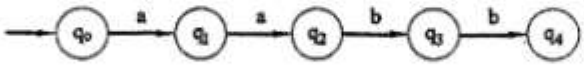
$$\begin{aligned}\delta(q_0, aaa) &\vdash \delta(q_0, aa)\\ &\vdash \delta(q_1, a)\\ &\vdash \delta(q_2, \in)\\ &\vdash q_2 \in F\end{aligned}$$

Example 17 : Design an NFA to accept a language of all strings with double 'a' followed by double 'b'.

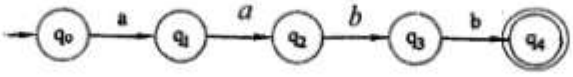
Solution : First design an NFA with three states q_0, q_1, q_2 and in which q_0 is the initial state to accept the string with two a's.



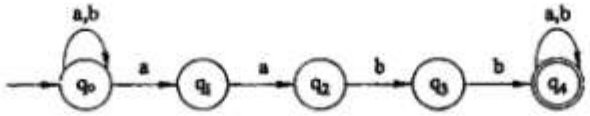
In second step we have to add another two states for the following two b's as shown below. Those states are q_3 and q_4



In the third step we assign q_4 as final state.



It can accept any number of a's or b's before first two successive a's. In the same way after the two successive b's also it can accept any number of a's or b's.



The NFA is defined as below :

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

where $Q = \{q_0, q_1, q_2, q_3, q_4\}$; $\Sigma = \{a, b\}$

$F = \{q_4\}$ and the transition table is given below :

	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
q_1	q_2	ϕ
q_2	ϕ	q_3
q_3	ϕ	q_4
(q_4)	q_4	q_4

Consider the string aaa bb :

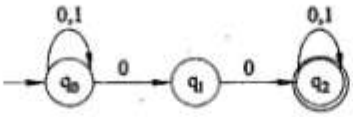
$$\begin{aligned}\delta(q_0, aaa\ bb) &\vdash \delta(q_0, aa\ bb)\\ &\vdash \delta(q_1, a\ bb)\\ &\vdash \delta(q_2, bb)\\ &\vdash \delta(q_3, b)\\ &\vdash \delta(q_4, \in)\\ &\vdash q_4 \in F\end{aligned}$$

$\therefore$ $aaa\ bb \in L(M)$

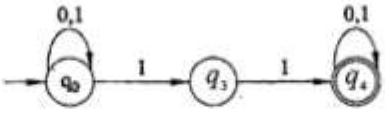
Example 18 : Design an NFA to accept strings with 0's and 1's such that string contains two consecutive 0's or two consecutive 1's.

Solution : First we design NFA to accept two consecutive 0's . This

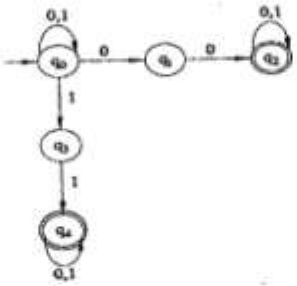
Next we can have any number of 0's and 1's before and after two consecutive zeros. i.e.,



then similarly NFA for accepting two consecutive 1's is



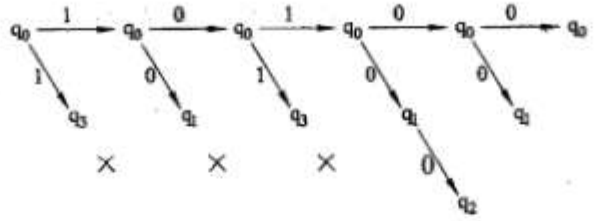
Combining above two designs



Transition table is

δ	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	q_2	ϕ
(q_2)	q_2	q_2
q_3	ϕ	q_4
(q_4)	q_4	q_4

Checking 10100 string with NFA.

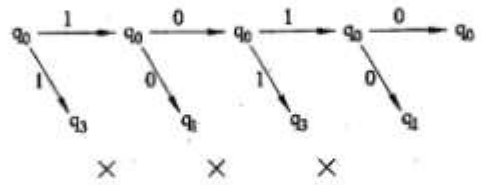


Observing above graph there are three completed paths for the string 10100. They are

$$q_0\,^1\,q_0\,^0\,q_0\,^1\,q_0\,^0\,q_0\,^0\,q_0$$
$$q_0\,^1\,q_0\,^0\,q_0\,^1\,q_0\,^0\,q_0\,^0\,q_1$$
$$q_0\,^1\,q_0\,^0\,q_0\,^1\,q_0\,^0\,q_1\,^0\,q_2$$

In all these three couple paths one path is ending with final state (q_2 *or* q_4). So, the string 10100 is accepted (It contains two consecutive 0's).

Now considering another stirng 1010, then graph becomes



There are two completed paths. But no path is ending with final state (q_2 *or* q_4). So, the string 1010 is not accepted (because it does n't contain two consecutive 0's or 1's).

1.3 EQUIVALENCE OF NFA AND DFA

As we have discussed in comparison of NFA and DFA that the power of NFA and DFA is equal. It means that if a NFA M_1 accepts language L, then some DFA M_2 also accepts it and vice - versa.

In this section, we will discuss about the equivalence of NFA and DFA. It is obvious that all DFA are NFA from NFA definition. We will see this in the following theorem.

Theorem 1.3 .1 : All DFA are NFA.

Proof : While discussing the proof, we will concentrate on two things :

1. How to construct the target NFA ? And
2. The acceptability should be same for both.

Step 1 : Construction of the target NFA from given DFA

Let $M = (Q, \Sigma, \delta, q_0, F)$ be the given DFA and $M_1 = (Q_1, \Sigma, \delta_1, s, F_1)$ be the target NFA, then

1. $Q_1 = Q$ (States of DFA are same for NFA),
2. Σ is same for both,
3. $\delta_1 = \delta$, it means, whatever transition function given for DFA M is same for the target NFA M_1.

We also see that
For DFA M : Transition function is defined as $Q \times \Sigma \to Q$, and

For NFA M_1 : Transition function is defined as $Q_1 \times \Sigma \to 2^{Q_1}$

So, $(Q \times \Sigma \to Q) \subseteq (Q_1 \times \Sigma \to 2^{Q_1})$ or $Q \subseteq 2^{Q_1}$

4. $s = q_0$ (Same starting point or initial state)
5. $F_1 = F$ (Same terminating points or final states)

Step 2 : The acceptability of DFA and NFA : Let w be an input string and accepted by DFA M and $w \in \Sigma^*$ if and only if $\delta'(q_0, w) = q_f, q_f \in F$ (δ' is indirect transition function)

For equivalent NFA M_1

$$\delta'_1(s, w) = \delta'(q_0, w) = q_f, q_f \in F$$

(By construction definition $\delta_1 = \delta$, $s = q_0$, $F_1 = F$ and δ'_1 is indirect transition function for NFA).
Thus, NFA M_1 also accepts w.

It means, $L(M_1) \subseteq L(M)$ (1)

Now, let w is accepted by NFA M_1 if and only if $\delta'_1(s, w) = \delta'_1(q_0, w) = q_f, q_f \in F_1$ and by construction definition $\delta_1 = \delta$, $s = q_0$, $F_1 = F$ and δ'_1 is indirect transition function for NFA.

So, for DFA M $\delta'(q_0, w) = q_f, q_f \in F$ (δ' is indirect transition function)

Thus, DFA M also accepts w.

Hence, $M(L) \subseteq L(M_1)$ (2)

Therefore, all DFA are NFA. (From (1) and (2))

Example : Let a DFA $M = (Q, \Sigma, \delta, q_0, F)$ as shown in below figure . Find an equivalent NFA.

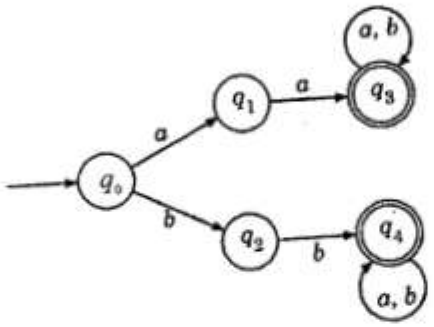


FIGURE : D F A

Solution :

Let equivalent NFA $M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ where $Q_1 = \{q_0, q_1, q_2, q_3, q_4\}, \Sigma = \{a, b\}$, $F_1 = \{q_3, q_4\}$, δ_1 is defined below.

	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_3	-
q_2	-	q_4
(q_3)	q_3	q_3
(q_4)	q_4	q_4

Theorem 1.3.2 : If there is a NFA M, then there exists equivalence DFA M_1 that has equal string recognizing power.

Proof : While discussing the proof, we will concentrate on two things :
1. How to construct the equivalent DFA ? And
2. The acceptability should be same for both.

Step 1 : Construction of the equivalent DFA M_1 from given NFA M

In NFA, zero, one or more next states are possible on a particular input. When we have more than one next state then we group all next states into one as $[q_1, q_2, q_3]$ and we call it one next state for equivalent DFA.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be the given NFA and $M_1 = (Q_1, \Sigma, \delta_1, s, F_1)$ be the equivalent DFA, then

1. $Q_1 \subseteq 2^Q$ (2^Q is the power set of the set Q.),
2. Σ is same for both,
3. $s = [q_0]$ is initial state for M_1,
4. $F_1 \subseteq 2^Q$ such that each member of F_1 has at least one final state from F.
5. δ_1 is constructed as follows :

   Let $w = a \in \Sigma$ and

   If for given NFA M : $\delta(q_0, a) = \{q_1, q_2, \ldots\ldots\ldots q_n\}$, then

   For equivalent DFA M_1 : $\delta_1([q_0], a) = [q_1, q_2, \ldots\ldots\ldots q_n]$

   And

   If for NFA M : $\delta(\{q_1, q_2, \ldots\ldots, q_n\}, a) = \{q_1, q_2, \ldots, q_m\}$, then

   For equivalent DFA M_1 : $\delta_1([q_1, q_2, \ldots\ldots, q_n], a) = [q_1, q_2, \ldots, q_m]$

Note : $[q_1, q_2, \ldots\ldots, q_i]$ denotes a single state for equivalent DFA.

Step 2 : The acceptability of DFA and NFA

We use the mathematical induction method to prove that $L(M) \subseteq L(M_1)$ and $L(M_1) \subseteq L(M)$ for all input strings $w \in \Sigma^*$.

Case 1 : Let $|w| = 0$, it means, $w = \in$ (Null string)

Let w is accepted by NFA M if and only if

$\delta(q_0, \in) = q_0$, and $q_0 \in F$ (Starting state is final state).

So, the initial state of DFA will be the final state, hence $w = \in$ is accepted by DFA also.

Case 2 : Let | w | = 1 and $w = a \in \Sigma$ is accepted by NFA M, then for NFA $M : \delta(q_0, a) = \{q_1, q_2, ..., q_n\}$, and $\{q_1, q_2, ..., q_n\}$ has at least one final state, then by constructive proof of equivalent DFA M_1 :

$\delta_1([q_0], a) = [q_1, q_2, ..., q_n]$ and $[q_1, q_2, ..., q_n]$ has at least one final state, so $[q_1, q_2, ..., q_n]$ is a final state for equivalent DFA M_1.

Therefore, the equivalent DFA M_1 also accepts $w = a$.

Case 3 : Suppose | w | = n and $w = a_1 a_2 .... a_n$ is accepted by both M and M_1 and

For NFA $M : \delta'(q_0, a_1 a_2 ... a_n) = \{q_1, q_2, ..., q_m\}$, and

For equivalent DFA M_1: $\delta'_1 = ([q_0], a_1 a_2 ... a_n) = [q_1, q_2, ..., q_m]$

Case 4 : Let | w | = n + 1 and w = yb

Where $|y| = n$, $y = a_1 a_2 ... a_n$ and $y, b \in \Sigma^*$ is accepted by NFA M If and only if

For NFA $M : \delta'(q_0, a_1 a_2 ..... a_n b) = \delta(\{q_1, q_2, ..., q_m\}, b) = \{q_1, q_2, ..., q_p\}$,

$(\{q_1, q_2, ..., q_p\}$ has at least one final state from the set F).

By constructive proof of equivalent DFA M_1

$\delta'_1([q_0], a_1 a_2 ..... a_n b) = \delta_1([q_1, q_2, ..., q_m], b) = [q_1, q_2, ..., q_p]$

$[q_1, q_2, ..., q_p]$ contains one final state from F, thus it is a final state for equivalent DFA M_1.

Therefore, M_1 also accepts the string $w = yb$.

(δ', δ'_1 are indirect transition functions for NFA M and DFA M_1 respectively.)

It has been proved that if NFA M accepts w then DFA M_1 also accepts w for any arbitrary string w.

Thus, $L(M_1) \subseteq L(M)$. (1)

Similarly, we can prove that if equivalent DFA M_1 accepts any string $w \in \Sigma^*$, then NFA also accepts it.

Thus, $M(L) \subseteq L(M_1)$. (2)

Hence, the statement of Theorem 1.3.2 is true. (From (1) and (2))

Example 1 : Consider a NFA shown in below figure. Find equivalent DFA.

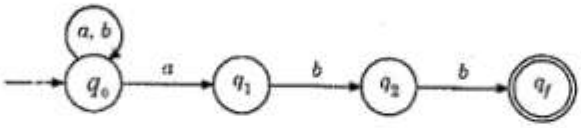


FIGURE : Non - deterministic finite Automata

Solution : Let given NFA $M = (Q, \Sigma, \delta, q_0, F)$ and equivalent DFA $M_1 = (Q_1, \Sigma, \delta_1, [q_0], F_1)$, where $Q = \{q_0, q_1, q_2, q_f\}, \Sigma = \{a, b\}$, s is starting state, $F = \{q_f\}$, and δ is defined as follows :

	q_0	b
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	-	$\{q_2\}$
q_2	-	$\{q_f\}$
(q_f)	-	-

δ_1 is defined as follows :

1. Keep the first row of NFA as it is with square bracket as follows :

	a	b
$[q_0]$	$[q_0,q_1]$	$[q_0]$

2. Now, we have two states : $[q_0],[q_0,q_1]$. We select the one next state that is not a present state till now and define the transition for it. We have only one next state $[q_0,q_1]$, which is not a present state .

	a	b
$\rightarrow[q_0]$	$[q_0,q_1]$	$[q_0]$
$[q_0,q_1]$	$[q_0,q_1]$	$[q_0,q_2]$

Since, $\delta_1([q_0,q_1],a)=[\delta(q_0,a)\cup\delta(q_1,a)]=[\{q_0,q_1\}\cup\phi]=[q_0,q_1]$, and
$\delta_1([q_0,q_1],b)=[\delta(q_0,b)\cup\delta(q_1,b)]=[\{q_0\}\cup\{q_2\}]=[q_0,q_2]$)

3. Now $[q_0,q_2]$ is the next selected state, because $[q_0,q_1]$ is defined already

	a	b
$\rightarrow[q_0]$	$[q_0,q_1]$	$[q_0]$
$[q_0,q_1]$	$[q_0,q_1]$	$[q_0,q_2]$
$[q_0,q_2]$	$[q_0,q_1]$	$[q_0,q_f]$

4. Now, the state $[q_0,q_f]$ is the next selected state.

	a	b
$\rightarrow[q_0]$	$[q_0,q_1]$	$[q_0]$
$[q_0,q_1]$	$[q_0,q_1]$	$[q_0,q_2]$
$[q_0,q_2]$	$[q_0,q_1]$	$[q_0,q_f]$
$[q_0,q_f]$	$[q_0,q_1]$	$[q_0]$

5. Now, we have no new choice of the next state to be considered as present state. This is the completion of transition table. We have

$Q_1=\{[q_0],[q_0,q_1],[q_0,q_2],[q_0,q_f],\}$ (All selected states in transition),

and $F_1 = \{[q_0, q_f]\}$ (Only one final state)

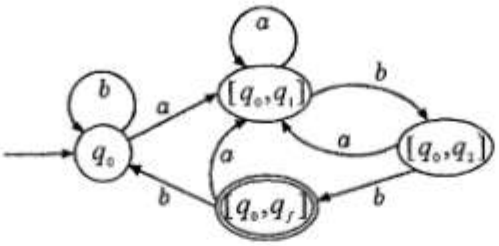


FIGURE : Transition Diagram of equivalent DFA

We see one thing here that not all states of 2^Q are selected for transition. We have selected those states, which are reachable from the initial state only and other remaining states of 2^Q are neglected.

So, finally we conclude that only those states of 2^Q are considered in transitions, which are reachable from the initial state.

Example 2 : Construct equivalent DFA for NFA M = ({ p, q, r, s }, { 0, 1 }, δ, p,{ q, s}), where δ is given below .

	0	1
p	{ q, s }	{ q }
(q)	{ r }	{ q, r }
r	{ s }	{ q, r }
(s)	-	{ p }

Solution : Let equivalent DFA is M_1 and $M_1 = (Q, \Sigma, \delta, [p], F)$

Construction of Transition table for equivalent DFA

	0	1
$\rightarrow [p]$	$[q, s]$	$[q]$
$[q]$	$[r]$	$[q, r]$
$[q, s]$	$[r]$	$[p, q, r]$
$[r]$	[s]	$[q, r]$
$[q, r]$	$[r, s]$	$[q, r]$
$[p, q, r]$	$[q, r, s]$	$[q, r]$

[s]	ϕ	[p]
$[r, s]$	[s]	$[p, q, r]$
$[q, r, s]$	$[r, s]$	$[p, q, r]$

Q = { [p], [q], [r], [s], [q,r], [r, s], [q, s], [p, q, r], [q, r, s] },
Σ = { 0, 1 }, [p] is the starting state,
and F = { [q], [s], [q, r] , [r, s] , [q, s], [p, q, r], [q, r, s].

Example 3 : Find a DFA equivalent to NFA $M = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_2\})$, where δ is defined as follows .

PS	NS	
	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_2\}$
q_1	$\{q_0\}$	$\{q_1\}$
(q_2)	-	$\{q_0, q_1\}$

Solution : Let $M' = (Q, \Sigma, \delta, [q_0], F)$ be the equivalent DFA, where $\Sigma = \{a, b\}$, and $[q_0]$ is the initial state.

Transition table :

	NS	
PS	0	1
$\rightarrow [q_0]$	$[q_0, q_1]$	$[q_2]$
$[q_2]$	ϕ	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_0]$	$[q_0, q_1]$

$Q = \{[q_0], [q_2], [q_0, q_1], [q_1, q_2]\}$, and $F = \{[q_2], [q_1, q_2]\}$

Transition diagram :

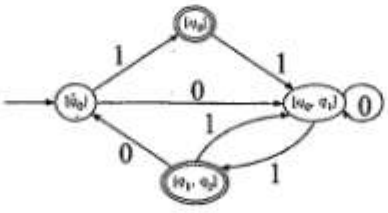


FIGURE : Equivalent DFA

Example 4 : A NFA which accepts set of strings over { 0, 1 } such that some two zero's are separated by a string over { 0, 1 } whose length is $4n$ $(n \geq 0)$ is shown in below figure . Construct equivalent DFA.

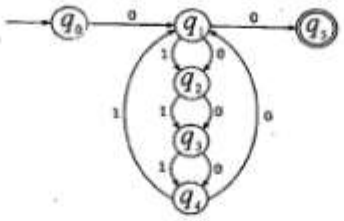


FIGURE : N F A

Solution : Let equivalent DFA $M = (Q, \Sigma, \delta, [q_0], F)$. constructing transition table for given NFA :

	(NS)	
(PS)	0	1
$\rightarrow q_0$	$\{ q_1 \}$	-
q_1	$\{ q_2, q_5 \}$	$\{ q_2 \}$
q_2	$\{ q_3 \}$	$\{ q_3 \}$
q_3	$\{ q_4 \}$	$\{ q_4 \}$
q_4	$\{ q_1 \}$	$\{ q_1 \}$
(q_5)	-	-

Constructing transition table for equivalent DFA :

	(NS)	
(PS)	0	1
$\rightarrow [q_0]$	$[q_1]$	ϕ
$[q_1]$	$[q_2, q_5]$	$[q_2]$
$[q_2]$	$[q_3]$	$[q_3]$
$[q_3]$	$[q_4]$	$[q_4]$
$[q_4]$	$[q_1]$	$[q_1]$
$([q_2, q_5])$	$[q_3]$	$[q_3]$

Where, $Q = \{[q_0],[q_1],[q_2],[q_3],[q_4],[q_2,q_5]\}, \Sigma = \{0,1\}, [q_0]$ is starting state, $F = \{[q_2,q_5]\}$, and transition function is defined above.

Transition diagram :

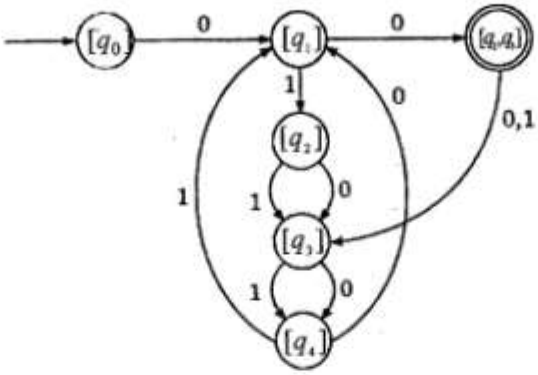


FIGURE : Equivalent DFA

1.5 NFA WITH $\in$ - MOVES

1.5.1 Finite automata With $\in$ - Transitions

This is same as NFA except we are using a special input symbol called epsilon ($\in$). Using this symbol path we can jump to one state to other state without reading any input symbol.

This also analytically indicated as 5 - tuple notation.

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

$Q \rightarrow$ set of states in design

$\Sigma \rightarrow$ input alphabet

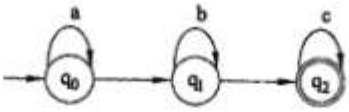
$q_0 \rightarrow$ initial state

$F \rightarrow$ final states ($\subseteq Q$)

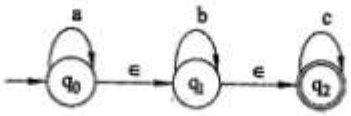
$\delta \rightarrow$ mapping function indicates $Q \times (\Sigma \cup \{\in\}) \rightarrow 2^Q$

Example : Draw a transition diagram of NFA which include transitions on the empty input $\in$ and accepts a language consisting of any number a's followed by any number of b's and which in turn followed by any number of c's.

Solution : It requires three states q_0, q_1 and q_2 and they accept any number of a's, b's and c's respectively. Assign q_2 as final state.



To reach from q_0 to q_1 and q_1 *to* q_2 no input will be given i. e., they treat $\in$ as their input and do the transition.



Normally these $\in$'s do not appear explicitly in the string.
The transition function for the NFA is shown below :

	a	b	c	$\in$
$\rightarrow q_0$	$\{q_0\}$	ϕ	ϕ	$\{q_1\}$
q_1	ϕ	$\{q_1\}$	ϕ	$\{q_2\}$
(q_2)	ϕ	ϕ	$\{q_2\}$	ϕ

For example consider the string $\omega = ab\,c$

String $\omega = ab\,c$ (i. e. , string in actual form is $a \in b \in c$ i. e., included along with epsilons).

$$\begin{aligned}\delta(q_0, abc) &\vdash \delta(q_0, bc)\\ &\vdash \delta(q_0, \in bc)\\ &\vdash \delta(q_1, bc)\\ &\vdash \delta(q_1, \in c)\\ &\vdash \delta(q_2, c) \quad \vdash q_2 \in F\end{aligned}$$

The path is shown below :

$$q_0 \overset{a}{\rightarrow} q_0 \overset{\in}{\rightarrow} q_1 \overset{b}{\rightarrow} q_1 \overset{\in}{\rightarrow} q_2 \overset{c}{\rightarrow} q_2$$

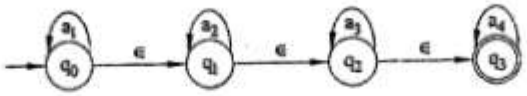
with arcs labeled $a, \in, b, \in, c$

Extension of Transition Function From δ to $\hat{\delta}$

The extended transition function $\hat{\delta}$ maps $Q \times \Sigma^*$ to 2^Q. It is important to compute the set of states reachable from a given state q_0 using $\in$ transitions only for constructing $\hat{\delta}$.

The $\in$ – closure (q_0) is used to denote the set of all vertices q_n such that there is a path from q_0 to q_n labeled $\in$.

Consider the problem



Here $\in$ - closure $(q_0) = \{q_0, q_1, q_2, q_3\}$

$\therefore \quad \hat{\delta}(q_0, \in) = \in\text{-}closure = \{q_0, q_1, q_2, q_3\}$

$\in$ – closure (q) is used to denote the set of all states s such that there is a path from q to s for string ω, includes edges labeled $\in$.

Note : The transition on $\in$ doesnot allow the NFA to accept Non - regular sets.

Definition : The extended transition function $\hat{\delta}$ is defined as follows :

(i) $\hat{\delta}(q, \in) = \in$ – closure (q)

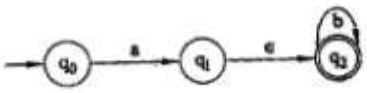
(ii) For ω in Σ^* and x in $\Sigma, \hat{\delta}(q, \omega x) = \in$ - closure (s), where s = { s | for some r in $\hat{\delta}(q, \omega), s \in \delta(r, x)$} δ can be extended $\hat{\delta}$ by extension to set of states.

(iii) $\delta(R, x) = \bigcup_{q \in R} \delta(q, x)$ and

(iv) $\hat{\delta}(R, \omega) = \bigcup_{q \in R} \hat{\delta}(q, \omega)$.

Note : $\hat{\delta}(q, a)$ is not necessarily equal to $\delta(q, a)$.

Example : The following NFA with $\in$ transitions accepts input strings with (a's and b's) single a or a followed by any number of b's.



The NFA accepts strings a, ab, abbb etc. by using $\in$ path between q_1 and q_2 we can move from q_1 state to q_2 without reading any input symbol. To accept ab first we are moving from q_0 to q_1 reading a and we can jump to q_2 state without reading any symbol there we accept b and we are ending with final state so it is accepted.

Equivalence of NFA with $\in$– Transitions and NFA without $\in$– Transitions

Theorem : If the language L is accepted by an NFA with $\in$– transitions, then the language L_1 is accepted by an NFA without $\in$– transitions.

Proof : Consider an NFA 'N' with $\in$– transitions where $N=(Q, \Sigma, \delta, q_0, F)$

Construct an NFA N_1 without $\in$– transitions $N_1=(Q_1, \Sigma, \delta_1, q_0, F_1)$

where $Q_1 = Q$ and

$$F_1 = \begin{cases} F \cup \{q_0\} & \text{if } \in\text{– closure}(q_0) \text{ contains a state of } F \\ F & \text{otherwise} \end{cases}$$

and $\delta_1(q,a)$ is $\hat{\delta}(q,a)$ for q in Q and a in Σ.

Consider a non - empty string ω. To show by induction $|\omega|$ that $\delta_1(q_0, \omega) = \hat{\delta}(q_0,\omega)$

For $\omega = \in$, the above statement is not true. Because

$$\delta_1(q_0, \in) = \{q_0\},$$

while $$\hat{\delta}(q_0, \in) = \in\text{–closure}\ (q_0)$$

Basis :

Start induction with string length one.

i. e., $|\omega| = 1$

Then w is a symbol a, and $\delta_1(q_0,a) = \hat{\delta}(q_0,a)$ by definition of δ_1.

Induction :

$|\omega| > 1$

Let $\omega = xy$ for symbol a in Σ.

Then $\delta_1(q_0, xy) = \delta_1(\delta_1(q_0, x), y)$

By inductive hypothesis

$$\delta_1(q_0, x) = \hat{\delta}(q_0, x)$$

Let $\hat{\delta}(q_0, x) = s$

We have to show that $\delta_1(s, y) = \hat{\delta}(q_0, xy)$

But $$\delta_1(s,y) = \bigcup_{q \in s} \delta_1(q,y) = \bigcup_{q \in s} \hat{\delta}(q,y)$$

then $$s = \hat{\delta}(q_0, x)$$

$\therefore$ $$\bigcup_{q \in s} \hat{\delta}(q,y) = \hat{\delta}(q_0, x)$$

By rule (Rule : For $\omega \in \Sigma^*$ and $x \in \Sigma$, $\hat{\delta}(q, \omega x) = \epsilon-$ closure (s),

where $s = \{ s \mid$ for some r in $\hat{\delta}(q, \omega)$, $s \in \delta(r, x)\}$ in the definition of $\hat{\delta}$).

Thus $\delta_1(q_0, xy) = \hat{\delta}(q_0, xy)$.

To complete the proof we shall show that $\delta'(q_0, w)$ contain a state of F' if and only if $\hat{\delta}(q_0, x)$ contain a state of F. For this two cases arises.

Case I : If $\omega = \epsilon$, this statement is true from the definition of F_1.

i. e., $\delta_1(q_0, \epsilon) = \{q_0\}$

$\Rightarrow$ $q_0 \in F'_1$

Whenever $\hat{\delta}(q_0, \epsilon)$ is $\epsilon-$ closure (q_0), contains a state in F (possibly is q_0).

Case II : If $\omega \neq \epsilon$ then W = xy for some symbol y.

If $\hat{\delta}(q_0, \omega)$ contains a state of F, $\Rightarrow \delta_1(q_0, \omega)$ contains some state in F'

Conversely, if $\delta_1(q_0, \omega) \in F'_1$ other than q_0, $\Rightarrow \hat{\delta}(q_0, \omega) \in F$.

If $\delta_1(q_0, \omega) \in q_0$ and $q_0 \notin F$, then

$\hat{\delta}(q_0, \omega) = \epsilon-$ closure $(\delta_1(\hat{\delta}(q_0, \omega), y))$,

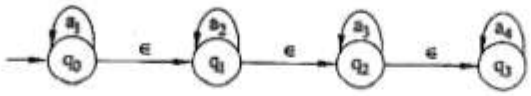
The state in ϵ - closure (q_0) and in F must be in $\hat{\delta}(q_0, \omega)$.

Calculation of ϵ - closure :

ϵ - closure of state (ϵ-closure (q)) defined as it is a set of all vertices p such that there is a path from q to p labelled ϵ (including itself).

Example :

Consider the NFA with ϵ - moves



ϵ – closure $(q_0) = \{ q_0, q_1, q_2, q_3 \}$

ϵ – closure $(q_1) = \{ q_1, q_2, q_3 \}$

ϵ – closure $(q_2) = \{ q_2, q_3 \}$

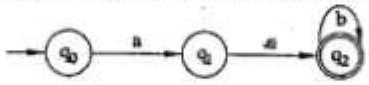
ϵ – closure $(q_3) = \{ q_3 \}$

Procedure to convert NFA with ϵ moves to NFA without ϵ moves

Let $N = (Q, \Sigma, \delta, q_0, F)$ is a NFA with ϵ moves then there exists $N' = (Q, \epsilon, \hat{\delta}, q_0, F')$ without ϵ moves

1. First find ϵ – closure of all states in the design.
2. Calculate extended transition function using following conversion formulae.
   (i) $\hat{\delta}(q, x) = \epsilon$ – closure $(\delta(\hat{\delta}(q, \epsilon), x))$
   (ii) $\hat{\delta}(q, \epsilon) = \epsilon$ – closure (q)
3. F' is a set of all states whose ϵ closure contains a final state in F.

Example 1 : Convert following NFA with ϵ moves to NFA without ϵ moves.



Solution : Transition table for given NFA is

δ	a	b	ϵ
$\rightarrow q_0$	q_1	ϕ	ϕ
q_1	ϕ	ϕ	q_2
(q_2)	ϕ	q_2	ϕ

(i) Finding $\in$ closure :

$\in$ – closure $(q_0) = \{q_0\}$

$\in$ – closure $(q_1) = \{q_1, q_2\}$

$\in$ – closure $(q_2) = \{q_2\}$

(ii) Extended Transition function :

$\hat{\delta}$	a	b
$\rightarrow q_0$	$\{q_1, q_2\}$	ϕ
(q_1)	ϕ	$\{q_2\}$
(q_2)	ϕ	$\{q_2\}$

$\hat{\delta}(q_0, a) = \in$ –*closure* $(\delta(\hat{\delta}(q_0, \in), a))$

$= \in$ – *closure* $(\delta(\in$ – *closure* $(q_0), a))$

$= \in$ – *closure* $(\delta(q_0, a))$

$= \in$ – *closure* (q_1)

$= \{q_1, q_2\}$

$\hat{\delta}(q_0, b) = \in$ – *closure* $(\delta(\hat{\delta}(q_0, \in), b))$

$= \in$ – *closure* $(\delta(\in$ – *closure* $(q_0), b))$

$= \in$ – *closure* $(\delta(q_0, b))$

$= \in$ – *closure* (ϕ)

$= \phi$

$\hat{\delta}(q_1, a) = \in$ – *closure* $(\delta(\hat{\delta}(q_1, \in), a))$

$= \in$ – *closure* $(\delta(\in$ – *closure* $(q_1), a))$

$= \in$ – *closure* $(\delta((q_1, q_2), a))$

$= \in$ – *closure* $(\delta(q_1, a) \cup \delta(q_2, a))$

$= \in$ – *closure* (ϕ)

$= \phi$

$\hat{\delta}(q_1, b)$ $= \in - closure\ (\delta(\hat{\delta}(q_1, \in), b))$

$= \in - closure\ (\delta(\in - closure(q_1), b))$

$= \in - closure\ (\delta((q_1, q_2), b))$

$= \in - closure\ (\delta(q_1, b) \cup \delta(q_2, b))$

$= \in - closure\ (q_2)$

$= \{q_2\}$

$\hat{\delta}(q_2, a)$ $= \in - closure\ (\delta(\hat{\delta}(q_2, \in), a))$

$= \in - closure\ (\delta(\in - closure(q_2), a))$

$= \in - closure\ (\delta(q_2, a))$

$= \in - closure\ (\phi)$

$= \phi$

$\hat{\delta}(q_2, b)$ $= \in - closure\ (\delta(\hat{\delta}(q_2, \in), b))$

$= \in - closure\ (\delta(\in - closure(q_2), b))$

$= \in - closure\ (\delta(q_2, b))$

$= \in - closure\ (q_2)$

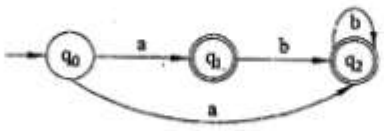
$= \{q_2\}$

(iii) Final states are q_1, q_2, because

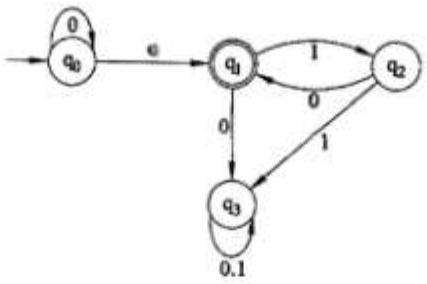
$\in - closure\ (q_1)$ contains final state

$\in - closure\ (q_2)$ contains final state

(iv) NFA without $\in$ moves is



Example 2 : Convert the following NFA with $\in -$ moves into equivalent NFA without $\in -$ moves.



Solution : Transition table is

	0	1	$\in$
$\rightarrow q_0$	q_0	ϕ	q_1
(q_1)	q_3	q_2	ϕ
q_2	q_1	q_3	ϕ
q_3	q_3	q_3	ϕ

(i) Finding $\in$- closure :

$\in -$ *closure* (q) is a set of states having paths on epsilon symbol from state q.

$\in -$ *closure* $(q_0) = \{ q_0, q_1 \}$

$\in -$ *closure* $(q_1) = \{ q_1 \}$

$\in -$ *closure* $(q_2) = \{ q_2 \}$

$\in -$ *closure* $(q_3) = \{ q_3 \}$

(ii) Extended Transition function :

$$\hat{\delta}(q_0, 0) = \in - closure\ (\delta\ (\hat{\delta}\ (q_0, \in), 0))$$

$$= \in - closure\ (\delta\ (\in - closure\ (q_0), 0))$$

$$= \in - closure\ (\delta\ ((q_0, q_1), 0))$$

$$= \in - closure(\delta(q_0,0) \cup \delta(q_1,0))$$

$= \in -closure\ (q_0, q_3)$

$= \in -closure\ (q_0) \cup \in -$ closure (q_3)

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

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

$\hat{\delta}(q_0, 1)$ $= \in -closure\ (\delta(\hat{\delta}(q_0, \in), 1))$

$= \in -closure\ (\delta(\in -closure\ (q_0), 1))$

$= \in -closure\ (\delta\ ((q_0, q_1), 1))$

$= \in -closure(\delta(q_0,1) \cup \delta(q_1,1)) = \in -closure(\phi \cup q_2)$

$= \in -closure\ (q_2)$

$=\{q_2\}$

Continuing like this the table is generalised as follows.

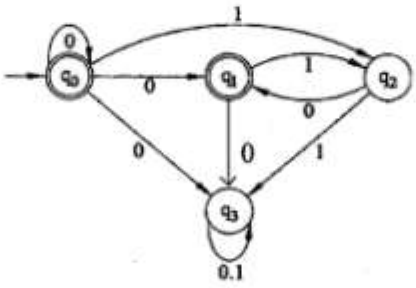
	0	1
$\rightarrow q_0$	$\{q_0,q_1,q_3\}$	q_2
q_1	q_3	q_2
q_2	q_1	q_3
q_3	q_3	q_3

(iii) Final states are q_0, q_1, because

$\in$- closure $(q_0) = \{q_0,q_1\}$ it contains final state

$\in$- closure $(q_1) = q_1$ is also final state

(iv) NFA without $\in$ moves is :



Example 3 : Find an equivalent NFA without $\epsilon-$ transitions for NFA with $\epsilon-$ transitions shown in below figure.

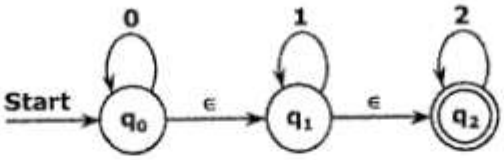


FIGURE : NFA with $\epsilon-$ transitions

Solution : The transition table is,

	Inputs			
States	0	1	2	ϵ
$\rightarrow q_0$	$\{q_0\}$	ϕ	ϕ	$\{q_1\}$
q_1	ϕ	$\{q_1\}$	ϕ	$\{q_2\}$
(q_2)	ϕ	ϕ	$\{q_2\}$	ϕ

TABLE : Transition Table for the NFA in above figure.

Given NFA $M = (\{q_0, q_1, q_2\}, \{0, 1, 2, \epsilon\}, \delta, q_0, \{q_2\})$.

Now NFA without $\epsilon-$ moves.

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

(i) Finding $\epsilon-$ closure:

$\epsilon-closure\ (q_0) = \{q_0, q_1, q_2\}$

$\epsilon-closure\ (q_1) = \{q_1, q_2\}$

$\epsilon-closure\ (q_2) = \{q_2\}$

(ii) Extended Transition function :

$$\hat{\delta}(q_0, 0) = \epsilon-closure\ (\delta\ (\hat{\delta}(q_0, \epsilon), 0))$$

$$= \epsilon-closure\ (\delta\ \{q_0, q_1, q_2\}, 0)$$

$$= \epsilon-closure(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))$$

$$= \in\text{-}closure\ (\{q_0\} \cup \phi \cup \phi)$$

$$= \in\text{-}closure\ (q_0)$$

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

$\hat{\delta}(q_0, 1)$ $= \in\text{-}closure\ (\delta(\hat{\delta}(q_0, \in), 1))$

$$= \in\text{-}closure\ (\delta(\{q_0, q_1, q_2\}, 1))$$

$$= \in\text{-}closure\ [\delta(q_0,1) \cup \delta(q_1,1) \cup \delta(q_2, 1)]$$

$$= \in\text{-}closure\ (\phi \cup q_1 \cup \phi)$$

$$= \in\text{-}closure\ (q_1)$$

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

Similarly for other transitions gives, transition table $\hat{\delta}(q, a)$

	Inputs		
States	0	1	2
→ q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

TABLE : Modified Transition Table for the NFA in above figure

(iii) F' contains q_0, q_1, q_2 because $\in$– closure (q_0), $\in$– closure (q_1) and $\in$– closure (q_2) contains q_2.

(iv) $M' = (Q, \Sigma, \hat{\delta}, q_0, F')$ NFA without $\in$– transitions is,

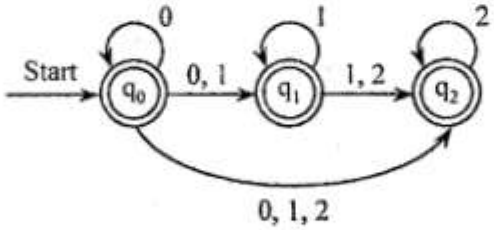


FIGURE : NFA without $\in$– transitions

Example 4 : For the following NFA with $\in-$ moves convert it into an NFA without $\in-$ moves.

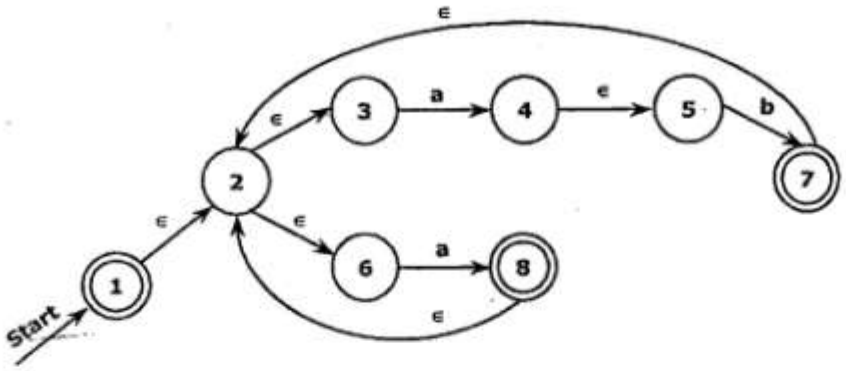


FIGURE : NFA with $\in-$ moves

Solution :

Let given NFA with $\in-$ moves be,

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

$Q = \{1,2,3,4,5,6,7,8\}$; $\Sigma = \{a, b\}$

$q_0 = 1$; $F = \{1, 7, 8\}$

(i) Finding $\in-$ closure :

First we need to find $\in-$ closure of all states of M.

$\hat{\delta}(q,\in)=\in-closure(q)$

$\hat{\delta}(1,\in)=\in-closure(1)=\{1,2, 3,6\}$

$\hat{\delta}(2,\in)=\in-closure(2)=\{2,3,6\}$

$\hat{\delta}(3,\in)=\in-closure(3)=\{3\}$

$\hat{\delta}(4,\in)=\in-closure(4)=\{4,5\}$

$\hat{\delta}(5,\in)=\in-closure(5)=\{5\}$

$\hat{\delta}(6,\in)=\in-closure(6)=\{6\}$

$\hat{\delta}(7,\in)=\in-closure(7)=\{2,3,6,7\}$

$\hat{\delta}(8,\in)=\in-closure(8)=\{2,3, 6,8\}$

(ii) Extended Transition function:

$\hat{\delta}(1,a) = \in -closure\ (\delta(\hat{\delta}(1,\in),a))$

$= \in - closure\ (\delta(\{1,2,3,6\}, a))$

$= \in - closure\ (\{4, 8\})$

$= \{2, 4,5, 6, 8\}$

$\hat{\delta}(1,b) = \in -closure(\delta(\hat{\delta}(1,\in),b))$

$= \in - closure\ (\delta(\{1,2,3,6\}, b))$

$= \in - closure\ (\phi)$

$= \{\phi\}$

$\hat{\delta}(2,a) = \in -closure(\delta(\hat{\delta}(2,\in),a))$

$= \{2, 4,5, 6, 8\}$

$\hat{\delta}(2,b) = \in -closure(\delta(\hat{\delta}(2,\in),b))$

$= \{\phi\}$

$\hat{\delta}(3,a) = \in -closure(\delta(\hat{\delta}(3,\in),a))$

$= \{4,5\}$

$\hat{\delta}(3,b) = \in -closure(\delta(\hat{\delta}(3,\in),a))$

$= \{\phi\}$

$\hat{\delta}(4,a) = \in -closure\ (\delta(\hat{\delta}(4,\in),a))$

$= \{\phi\}$

$\hat{\delta}(4,b) = \in -closure\ (\delta(\hat{\delta}(4,\in),b))$

$= \{7\}$

$\hat{\delta}(5,a) = \in -closure\ (\delta(\hat{\delta}(5,\in),a))$

$= \{\phi\}$

$\hat{\delta}(5,b) = \in -closure\ (\delta(\hat{\delta}(5,\in),b))$

$= \{7\}$

$\hat{\delta}(6,a) = \in -closure\ (\delta(\hat{\delta}(6,\in),a))$

$= \{8\}$

$$\hat{\delta}(6,b) = \in\text{-}closure\ (\delta(\hat{\delta}(6,\in),b))$$
$$=\{\phi\}$$

$$\hat{\delta}(7,a) = \in\text{-}closure\ (\delta(\hat{\delta}(7,\in),a))$$
$$=\{4,8\}$$

$$\hat{\delta}(7,b) = \in\text{-}closure\ (\delta(\hat{\delta}(7,\in),b))$$
$$=\{\phi\}$$

$$\hat{\delta}(8,a) = \in\text{-}closure\ (\delta(\hat{\delta}(8,\in),a))$$
$$=\{8\}$$

$$\hat{\delta}(8,b) = \in\text{-}closure\ (\delta(\hat{\delta}(8,\in),b))$$
$$=\{\phi\}$$

Final states of M' includes all states whose $\in$– closure contains a final state of M.

$\therefore$ $F = \{1, 7, 8\}$

Transition table is,

		a	b
→	①	{ 2, 4, 5, 6, 8 }	ϕ
	2	{ 2, 4, 5, 6, 8 }	ϕ
	3	{ 4, 5 }	ϕ
	4	ϕ	{ 7 }
	5	ϕ	{ 7 }
	6	{ 8 }	ϕ
	⑦	{ 4, 8 }	ϕ
	⑧	{ 8 }	ϕ

FIGURE : Transition Table for the NFA in above figure.

Transition diagram of NFA without $\in$ – transitions is,

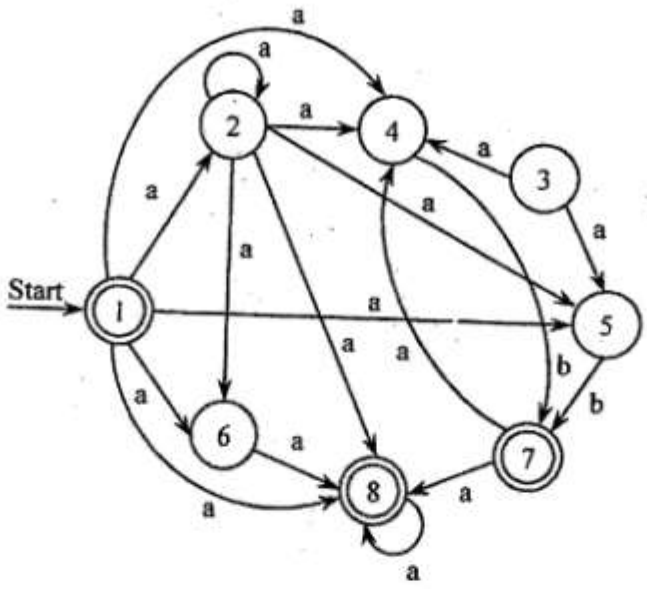


FIGURE :NFA without $\in$ – transitions

REVIEW QUESTIONS

Q1. Explain difference between DFA and NFA.

Answer :

For Answer refer to Page No : 1.12.

Q2. Consider the FA shown in below figure. Check the acceptability of following strings:

(a) 0101 (b) 0111 (c) 001

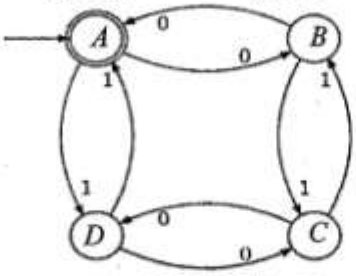


FIGURE : Finite automata

Answer :

For Answer refer to example - 1 , Page No : 1.13.

Q3. Let a DFA $M = (Q, \Sigma, \delta, q_0, F)$ is shown in below figure.

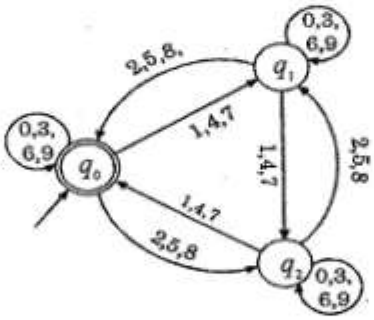


FIGURE : D F A

Check that string 33150 is recognized by above DFA or not ?

Answer :

For Answer refer to example - 2 , Page No : 1.13.

Q4. Consider below transition diagram and verify whether the following strings will be accepted or not ? Explain.

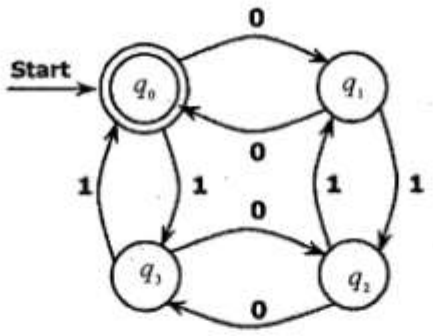


FIGURE : Given Transition Diagram

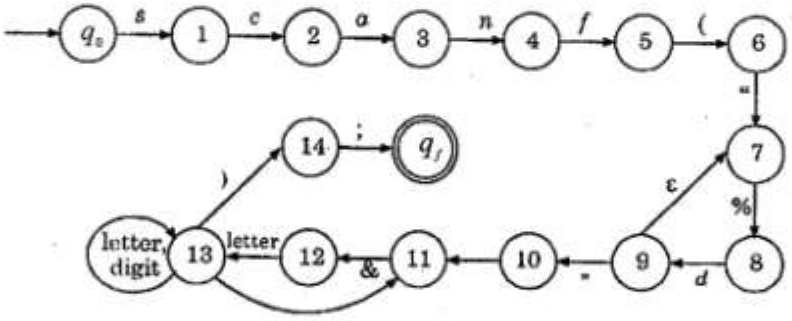
i) 0011 ii) 010101 iii) 111100 iv) 1011101 .

Answer :

For Answer refer to example - 3 , Page No : 1.14.

Q5. Consider the NFA shown in below figure. Check the acceptability of following string

scanf ("%d", & num) ;



Note : Letter stands for any symbol from { a, b, ... , z } and digit stands for any digit from { 0, 1, 2, ... 9 } .

Answer :

For Answer refer to example - 4 , Page No : 1.15

Q6. Obtain a DFA to accept strings of a's and b's starting with the string ab .

Answer :

For Answer refer to example - 5 , Page No : 1.16.

Q7. Draw a DFA to accept string of 0's and 1's ending with the string 011.

Answer :

For Answer refer to example - 6 , Page No : 1.18.

Q8. Obtain a DFA to accept strings of a's and b's having a substring aa .

Answer :

For Answer refer to example - 7 , Page No : 1.20.

Q9. Obtain a DFA to accept strings of a's and b's except those containing the substring aab.

Answer :

For Answer refer to example - 8 , Page No : 1.22.

Q10. Obtain a DFA to accept strings of a's and b's having exactly one a, atleast one a, not more than three a's.

Answer :

For Answer refer to example - 9 , Page No : 1.24.

Q11. Obtain a DFA to accept the language $L = \{ awa \mid w \in (a+b)^* \}$.

Answer :

For Answer refer to example - 10 , Page No : 1.27.

Q12. Obtain a DFA to accept even number of a's, odd number of a's .

Answer :

For Answer refer to example - 11, Page No : 1.29.

Q13. Obtain a DFA to accept strings of a's and b's having even number of a's and b's.

Answer :

For Answer refer to example - 12 , Page No : 1.29.

Q14. Design a DFA, M that accepts the language $L(M) = \{ w | w \in \{a, b\}^* \}$ and w does not contain 3 consecutive b's.

Answer :

For Answer refer to example - 13 , Page No : 1.31.

Q15. Design DFA which accepts language L = { 0, 000, 00000, ... } over { 0 }.

Answer :

For Answer refer to example - 14, Page No : 1.31.

Q16. Obtain an NFA to accept the following language $L = \{ w | w \in abab^n$ or aba^n where $n \geq 0 \}$

Answer :

For Answer refer to example - 15 , Page No : 1.32.

Q17. Design NFA to accept strings with a's and b's such that the string end with 'aa'.

Answer :

For Answer refer to example - 16, Page No : 1.32.

Q18. Design an NFA to accept a language of all strings with double 'a' followed by double 'b'.

Answer :

For Answer refer to example - 17 , Page No : 1.35.

Q19. Design an NFA to accept strings with 0's and 1's such that string contains two consecutive 0's or two consecutive 1's.

Answer :

For Answer refer to example - 18 , Page No : 1.36.

Q20. Consider a NFA shown in below figure. Find equivalent DFA.

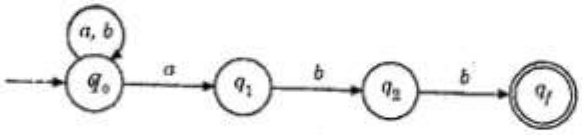


FIGURE : Non - deterministic finite Automata

Answer :

For Answer refer to example - 1 , Page No : 1.43.

Q21. Construct equivalent DFA for NFA M = ({ p, q, r, s }, { 0, 1 }, δ, p,{ q, s}), where δ is given below.

	0	1
p	{q, s}	{q}
(q)	{r}	{q, r}
r	{s}	{q, r}
(s)	-	{p}

Answer :

For Answer refer to example - 2 , Page No : 1.45.

Q22. Find a DFA equivalent to NFA $M = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_2\})$, where δ is defined as follows .

PS	NS	
	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_2\}$
q_1	$\{q_0\}$	$\{q_1\}$
(q_2)	-	$\{q_0, q_1\}$

Answer :

For Answer refer to example - 3 , Page No : 1.46.

Q23. A NFA which accepts set of strings over { 0, 1 } such that some two zero's are separated by a string over { 0, 1 } whose length is $4n$ $(n \geq 0)$ is shown in below figure . Construct equivalent DFA.

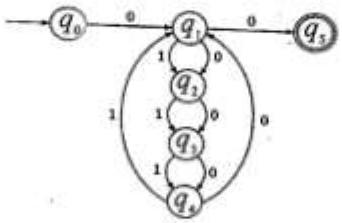
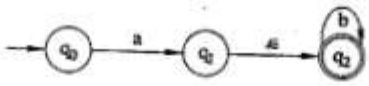


FIGURE : N F A

Answer :

For Answer refer to example - 4 , Page No : 1.47.

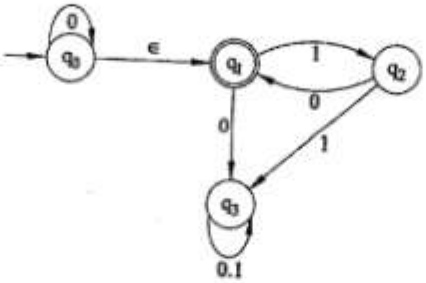
Q24. Convert following NFA with $\in$ moves to NFA without $\in$ moves.



Answer :

For Answer refer to example - 1 , Page No : 1.53.

Q25. Convert the following NFA with $\in-$ moves into equivalent NFA without $\in-$ moves.



Answer :

For Answer refer to example - 2 , Page No : 1.56.

Q26. Find an equivalent NFA without $\in-$ transitions for NFA with $\in-$ transitions shown in below figure.

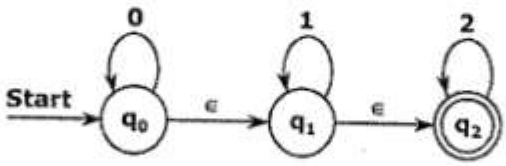


FIGURE : NFA with $\in-$ transitions

Answer :

For Answer refer to example - 3 , Page No : 1.58.

Q27. For the following NFA with $\epsilon-$ moves convert it into an NFA without $\epsilon-$ moves.

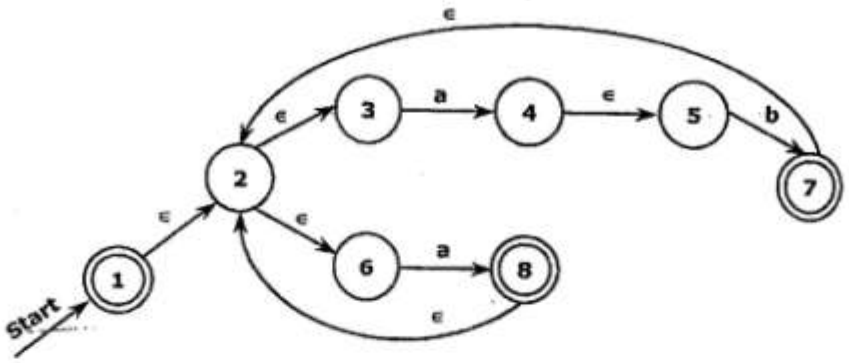


FIGURE : NFA with $\epsilon-$ moves

Answer :

For Answer refer to example - 4 , Page No : 1.60.

OBJECTIVE TYPE QUESTIONS

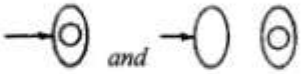
1. Which of the following is there is an FA?
   (a) State Transition (b) Input
   (c) State d) All of the above.
2. The basic limitations of Finite state machine is that
   (a) it sometimes recognizes non regular language
   (b) it sometimes does not recognizes regular language.
   (c) it can't remember arbitrary large information
   (d) all of the above.
3. Given a dfa $A = \langle S, \Sigma, s_0, \delta, F \rangle$, A accepts a word $w \in \Sigma^*$ iff
   (a) $\delta(s,w)$ *doesn't* $\in F$, *Where* $s \neq s_0$ (b) $\delta(s,w) \in F$, *Where* $s \neq s_0$
   (c) $\delta(s,w)$ *doesn't* $\in F$, *Where* $s = s_0$ (d) $\delta(s,w) \in F$, *Where* $s = s_0$
4. dfa can recognize
   (a) Only regular language (b) Only unambiguous grammar
   (c) Only CFG (d) Any grammar
5. dfa has:
   (a) Unique path(for a set of inputs) to the final state
   (b) Single final state
   (c) More than one initial states (d) All of the above.
6. The language generated by a deterministic finite automata is,
   (a) Informal language. (b) Context sensitive language
   (c) Context free language (d) Regular Language
7. It is given that $\delta(q,x) = \delta(q,y)$, then $\delta(q,xz) = \delta(q,yz)$ for All strings z in:
   (a) $\sim \Sigma$ (b) Σ^+ (c) Σ (d) Σ^*
8. Find the false statement for finite automata,
   (a) *if* $\delta(q,y) = \delta(q,x)$ *then* $\delta(q,xz) = \delta(q,yz)$. (b) $\delta(q,\in) = q$
   (c) $\delta(q,xw) = \delta(q,wx)$ (d) $\delta(q,xw) = \delta((q,x),w)$
9. Consider the FA for a switch with ON/OFF facilities. The automata can be designed with minimum no of states _____?
   (a) 4 (b) 3 (c) 2 (d) 1

10. Application of finite automata cannot be found in:
    (a) String matching (b) Lexical analyzers
    (c) Spelling checkers (d) Storage purpose
11. Find the false statement : An instantaneous description, of the finite-state automation is a singleton *uqv*, where:
    (a) the configuration is said to be a final configuration if $v = \in$ *and* q is the initial state.
    (b) the configuration is said to be an initial configuration if $u = \in$ *and* q is the initial state
    (c) uv is a string in Σ^*
    (d) q is a state in S
12. L is a nonempty language such that any w in L has length n, then any dfa accepting L must have
    (a) exactly $(n+1)$ states. (b) .atmost $(n+1)$ states
    (c) atleast $(n+1)$ states (d) exactly n states
13. Find the false statement for finite automata.
    (a) *if* $\delta(q,y) = \delta(q,x)$ *then* $\delta(q,xz) = \delta(q,yz)$
    (b) $\delta(q, \in) = q$
    (c) $\delta(q,xw) = \delta(q,wx)$
    (d) $\delta(q,xw) = \delta(delta(q,x),w)$
14. If, in a dfa, $\delta(q_1,x) = q_2$ *and* $\delta(q_2,y) = q_1$, *then* $\delta(q_1,xy)$ is
    (a) some state q_3 (b) q_1 (c) q_2 (d) None of the above.
15. For a deterministic finite automata, $M = (S,\Sigma,\delta,q_0,F);\delta$, the transition function is defined as:
    (a) $\delta : S \times \Sigma \to S^+$ (b) $\delta : S \times S \to \Sigma$
    (c) $\delta : s \times \Sigma \to \Sigma$ (d) $\delta : s \times \Sigma u\{\in\} \to Q$
16. The rules for nfa state that .....
    (a) every state of a nfa may have zero, one, or many exiting transition arrow for each symbol in the alphabet,plus".
    (b) every state of a nfa may have zero, one, many exiting transition arrow for each symbol in the alphabet,
    (c) every state of a nfa must always have exactly one exiting transition arrow for each symbol in the alphabet.
    (d) every state of a nfa must always have at most one exiting transition arrow for each symbol in alphabet.

17. The rules for dfa state that .....
(a) every state of a dfa must always have exactly one exiting transition arrow for each symbol in the alphabet.
(b) every state of dfa must always have at most one exiting transition arrow for each symbol in the alphabet.
(c) every state of a dfa may have zero, one, or many exiting transition arrow for each symbol in the alphabet,plus".
(d) every state of a dfa may have zero, one, or many exiting transition arrow for each symbol in the alphabet.

18. A nfa computes by reading in an input symbol from a string, and splits into multiple copies of itself, one for each possible transition. If the next input symbol doesn't appear on any of the arrows existing for the current state of a copy of the machine, that copy dics. A nfa accepts an input string when all the input symbols have been read and .....
(a) any one of the alive copies of the machine are in an accept state.
(b) all copies of the machine that died were in a reject state
(c) any one copy of the machine that died was in a reject state.
(d) all of the alive copies of the machine are in an accept state

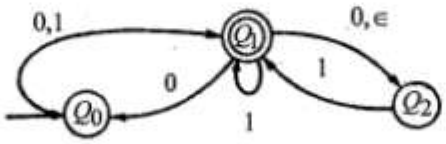
19. Consider the following two finite state machine in Figure.
(a) The first finite state machine accepts nothing
(b) Both are equivalent
(c) The second finite state machine accepts e-only
(d) none of the above.



20. For text searching applications which of them is used:
(a) npda (b) pda (c) dfa (d) nfa

21. If S is the number of states in ndfa then equivalent dfa can have maximum of
(a) $2^s - 1$ states (b) 2^s states (c) S-1 states (d) S states

22. How many of 00, 01001, 10010, 000, 0000 are accepted by the following nfa :



(a) 5 (b) 4 (c) 1 (d) 2

23. For a non-deterministic finite accepter, $M = (S, \Sigma, \delta, q_0, F); \delta$, the transition function is defined as:

(a) $\delta : S \times (\Sigma \times \{\in\}) \to 2^S$ (b) $\delta : S \times \Sigma \to S$

(c) $\delta : S \times \Sigma \to 2^S$ (d) None of these.

24. Find the true statement,

(a) There is nothing like non-determinism in finite-state automata.

(b) It depends from case to case.

(c) Non-determinism does not add to the recognition power of finite-state automata.

(d) Non-determinism adds to the recognition power of finite-state automata.

25. Given an arbitrary non-deterministic finite automation(nfa) with N states, the maximum number of states is an equivalent minimized dfa is at least

(a) $N!$ (b) 2^N (c) 2^N (d) N^2

26. $M = \langle \{q_1, q_2, q_3\}, \{0,1\}, \delta, q_1, \{q_1\} \rangle$ is a nondeterministic finite automation, where delta is given by

$\delta(q_1,0) = \{q_2, q_3\}$ $\delta(q_1,1) = \{q_1\}$

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

$\delta(q_3,0) = \{q_2\}$, $\delta(q_3,1) = \{q_1, q_2\}$

An equivalent dfa is given by which one of the following :

(a)

	0	1
q_0	q_1	q_0
q_1	q_2	q_2
q_2	q_0	q_3
q_3	q_2	q_3

(b)

	0	1
q_0	q_1	q_0
q_1	q_2	q_2
q_2	q_3	q_0
q_3	q_3	q_2

(c)

	0	1
q_0	q_1	q_0
q_1	q_2	q_2
q_2	q_0	q_3
q_3	q_3	q_2

(d)

	0	1
q_0	q_1	q_0
q_1	q_2	q_2
q_2	q_3	q_0
q_3	q_3	q_2

27. The recognizable property of dfa and ndfa
    (a) Must be same (b) May be different
    (c) Must be different (d) None of the above.

ANSWER KEY

1 (d) 2 (c) 3 (d) 4(a) 5 (a) 6 (d) 7 (b) 8 (c) 9 (c) 10 (d)

11 (a) 12 (c) 13(c) 14(b) 15 (c) 16 (a) 17 (a) 18 (a) 19 (d) 20 (d)
21 (b) 22(d) 23(a) 24 (c) 25 (c) 26 (c) 27(a)

2

FINITE STATE MACHINES

After going through this chapter, you should be able to understand :

- Finite State Machines
- Moore & Mealy Machines
- Equivalence of Moore & Mealy Machines
- Equivalence of two FSMs
- Minimization of FSM

2.1 FINITE STATE MACHINES (FSMs)

A finite state machine is similar to finite automata having additional capability of outputs.

A model of finite state machine is shown in below figure.

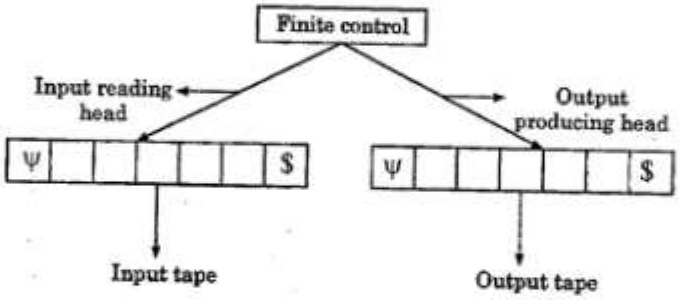


FIGURE : Model of FSM

2.1.1 Description of FSM

A finite state machine is represented by 6 - tuple $(Q,\Sigma,\Delta,\delta,\lambda,q_0)$, where

1. Q is finite and non - empty set of states,
2. Σ is input alphabet,
3. Δ is output alphabet,

4. δ is transition function which maps present state and input symbol on to the next state or $Q \times \Sigma \rightarrow Q$,
5. λ is the output function, and
6. $q_0 \in Q$, is the initial state .

2.1.2 Representation of FSM

We represent a finite state machine in two ways ; one is by transition table, and another is by transition diagram . In transition diagram , edges are labeled with Input / output.

Suppose , in transition table the entry is defined by a function F, so for input a_i and state q_i

$F(q_i, a_i) = (\delta(q_i, a_i), \lambda(q_i, a_i))$ (where δ is transition function, λ is output function.)

Example 1 : Consider a finite state machine, which changes 1's into 0's and 0's into 1's (1's complement) as shown in below figure .

Transition diagram :

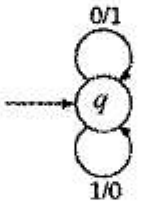


FIGURE : Finite state machine

Transition table :

Inputs				
	0		1	
Present State(PS)	Next State (NS)	Output	Next State (NS)	Output
q	q	1	q	0

Example 2 : Consider the finite state machine shown in below figure, which outputs the 2's complement of input binary number reading from least significant bit (LSB).

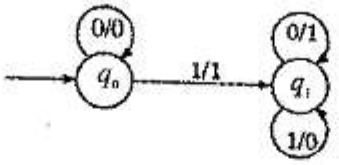


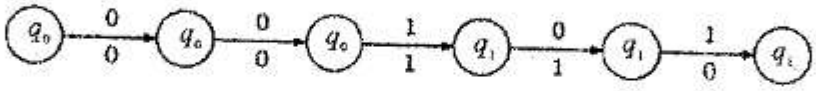
FIGURE : Finite State machine

Suppose, input is 10100. What is the output ?

Solution : The finite state machine reads the input from right side (LSB).

Transition sequence for input 10100 :

Inputs ⟶



Outputs ⟶

So, the output is 01100.

2.2 MOORE MACHINE

If the *output of finite state machine is dependent on present state only,* then this model of finite state machine is known as Moore machine.

A Moore machine is represented by 6-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where

1. Q is finite and non-empty set of states,
2. Σ is input alphabet,
3. Δ is output alphabet,
4. δ is transition function which maps present state and input symbol on to the next state or $Q \times \Sigma \rightarrow Q$,
5. λ is the output function which maps $Q \rightarrow \Delta$, (Present state $\rightarrow$ Output), and
6. $q_0 \in Q$, is the initial state.

If $Z(t)$, $q(t)$ are output and present state respectively at time t then

$$Z(t) = \lambda(q(t)).$$

For input $\in$ (null string), $Z(t) = \lambda$ (initial state)

Example 1 : Consider the Moore machine shown in below figure.Construct the transition table. What is the output for input 01010 ?

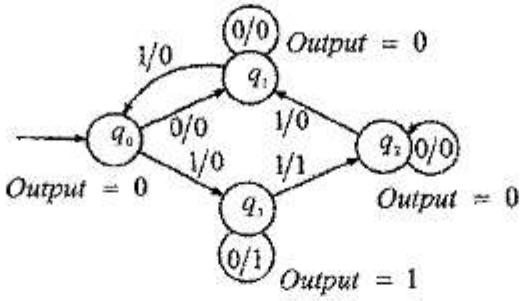
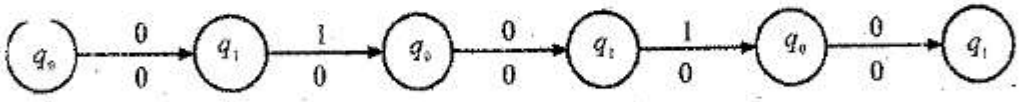


FIGURE: Moore machine

Solution : Transition table is as follows :

	Inputs		
	0	1	
Present State (PS)	Next State State (NS)	Next State State (NS)	Output
q_0	q_1	q_3	0
q_1	q_2	q_0	0
q_2	q_2	q_1	0
q_3	q_3	q_2	1

Transition sequence for string 01010 :



So, the output is 00000.

Note : Since, the output of Moore machine does not depend on input. So, the first output symbol is additional from the initial state without reading the input i.e., null input and output length is one greater than the input length, but not included in the above output.

Example 2 : Design a Moore machine, which outputs residue mod 3 for each binary input string treated as a binary integer.

Solution : Let Moore machine $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where $\Sigma = \{0, 1\}$

$\Delta = \{0, 1, 2\}$ (outputs after mod 3),

Let three states $\{q_0, q_1, q_2\}$ are there and

State q_0 outputs 0,

State q_1 outputs 1, and

State q_2 outputs 2.

If input is binary string X, then

X is followed by a 0 is equivalent to twice of X

X is followed by a 1 is equivalent to twice of X plus 1.

$X0 = (2 * X)_{10}$ (in decimal system), and

$X1 = (2 * X)_{10} + 1$ (in decimal system)

If X mod 3 = r, for r = 0 or 1 or 2, then

X0 mod 3 = 2* r mod 3 **(For input 0)**

= 0 or 2 or 1

For transition :

$q_r \rightarrow q_{2*r \bmod 3}$ for r = 0, 1, 2

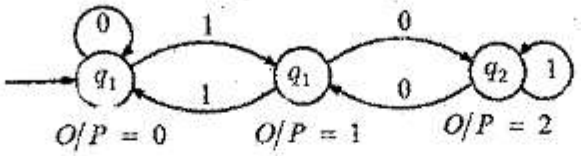
$X1 \bmod 3 = (2 * r + 1) \bmod 3$ **(For input 1)**

= 1, 0, 2

For transition :

$q_r \rightarrow q_{(2*r+1) \bmod 3}$ for r = 0, 1, 2

Transition diagram :



Example 3 : Design a Moore machine which reads input from (0+1+2)* and outputs residue mod 5 of the input. Input is considered at base 3 and it is treated as ternary integer.

Solution :

Let Moore machine $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ produces output residue mod 5 for each input string written in base 3.

$\Sigma = \{0, 1, 2\}, \Delta = \{0, 1, 2, 3, 4\}$

Let five states $\{q_0, q_1, q_2, q_3, q_4\}$ are there and

State q_0 outputs 0,

State q_1 outputs 1,

State q_2 outputs 2,

State q_3 outputs 3, and

State q_4 outputs 4.

If input is binary string W, then
W is followed by a 0 is equivalent to thrice of W,
W is followed by a 1 is equivalent to thrice of W plus 1,
W is followed by a 2 is equivalent to thrice of W plus 2.

Or

$W0 \equiv (3 * W)_{10}$ (in decimal system),

$W1 \equiv (3 * W)_{10} + 1$ (in decimal system),

$W2 \equiv (3 * W)_{10} + 2$ (in decimal system)

If $W \bmod 5 = r$, for $r = \{0,1,2,3,4\}$ (in the order of the elements), then

$W0 \bmod 5 = 3 * r \bmod 5$ **(For input 0)**

$= \{0,3,1,4,2\}$ (In the order of elements)

For transition :

$Q_r \rightarrow Q_{3*r \bmod 5}$ for $r = \{0,1,2,3,4\}$ (in the order of the elements)

$W1 \bmod 5 = (3 * r + 1) \bmod 5$ **(for input 1)**

$= \{1,4,2,0,3\}$ (In the order of elements)

For transition :

$Q_r \rightarrow Q_{(3*r+1) \bmod 5}$ for $r = \{0,1,2,3,4\}$ (in the order of the elements)

$W2 \bmod 5 = (3 * r + 2) \bmod 5$ **(for input 2)**

$= \{2,0,3,1,4\}$ (In the order of elements)

For transition :

$Q_r \rightarrow Q_{(3*r+2) \bmod 5}$ for $r = \{0,1,2,3,4\}$ (in the order of the elements)

Transition table

	Inputs			
	0	1	2	
PS	NS	NS	NS	Output
q_0	q_0	q_1	q_2	0
q_1	q_3	q_4	q_0	1
q_2	q_1	q_2	q_3	2
q_3	q_4	q_0	q_1	3
q_4	q_2	q_3	q_4	4

2.3 MEALY MACHINE

If the *output of finite state machine is dependent on present state and present input,* then this model of finite state machine is known as Mealy machine.

A Mealy machine is described by 6 - tuple $(Q,\Sigma,\Lambda,\delta,\lambda,q_0)$,

where

1. Q is finite and non-empty set of states,
2. Σ is input alphabet,
3. Δ is output alphabet,

4. δ is transition function which maps present state and input symbol on to the next state or $Q \times \Sigma \rightarrow Q$,
5. λ is the output function which maps $Q \times \Sigma \rightarrow \Delta$,((Present state, present input symbol) $\rightarrow$ Output), and
6. $q_0 \in Q$, is the initial state.

If $Z(t)$, $q(t)$, and $x(t)$ are output, present state, and present input respectively at time t, Then, $Z(t) = \lambda(q(t), x(t))$

For input $\in$ (null string), $Z(t) = \in$

Example 1: Consider the Mealy machine shown in below figure. Construct the transition table and find the output for input 01010.

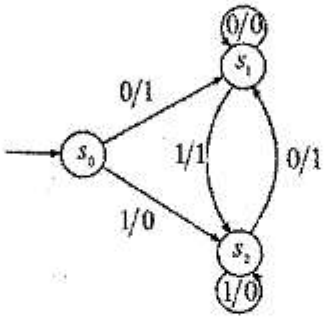
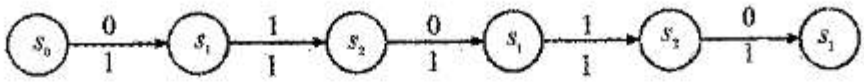


FIGURE : Mealy Machine

Solution : Transition table is constructed below.

	Inputs			
	0		1	
PS	NS	Output	NS	Output
s_0	s_1	1	s_2	0
s_1	s_1	0	s_2	1
s_2	s_1	1	s_2	0

Transition sequence for input 01010



(So, the output is 11111.)
(Note : The output length is equal to the input length).

Example 2 : Construct a Mealy machine which reads input from {0, 1} and outputs EVEN or ODD according to total number of 1's even or odd.

Solution :

We consider two states q_0, which outputs EVEN and q_1 which outputs ODD.

Suppose, $a \in (0 + 1)^*$ has even number of 1's, then a11 also has even number of 1's.

Suppose, $b \in (0 + 1)^*$ has odd number of 1's then b1 also has odd number of 1's.

Transition diagram :

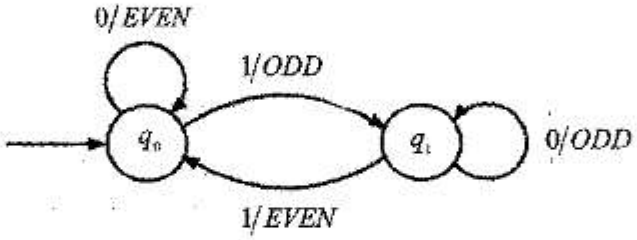


FIGURE : Mealy Machine

Example 3 : Design a Mealy machine which reads the input from (0+1)* and produces the following outputs.

(i) If input ends in 101, output is A,
(ii) If input ends 110, the output is B, and
(iii) For other inputs, output is C.

Solution : Suppose, Mealy machine $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ which reads the inputs from $(0 + 1)^*$, starting from the least significant bit (LSB).

Consider three LSBs of	Input	Output
	...000 (X)	C
	...001 (X)	C
	...010 (X)	C
	...011 (X)	C
	...100 (X)	C
	...101	A
	...110	B
	...111 (X)	C

Transition diagram :

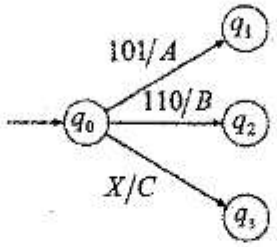


FIGURE : Moore Machine

2.4 EQUIVALENCE OF MOORE AND MEALY MACHINES

We can construct equivalent Mealy machine for a Moore machine and vice-versa. Let M_1 and M_2 be equivalent Moore and Mealy machines respectively. The two outputs $T_1(w)$ and $T_2(w)$ are produced by the machines M_1 and M_2 respectively for input string w. Then the length of $T_1(w)$ is one greater than the length of $T_2(w)$, i.e.

$$|T_1(w)| = |T_2(w)| + 1$$

The additional length is due to the output produced by initial state of Moore machine. Let output symbol x is the additional output produced by the initial state of Moore machine, then $T_1(w) = x\,T_2(w)$.

It means that if we neglect the one initial output produced by the initial state of Moore machine, then outputs produced by both machines are equivalent. *The additional output is produced by the initial state* of (for input $\in$) Moore machine without reading the input.

Conversion of Moore Machine to Mealy Machine

Theorem : If $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is a Moore machine then there exists a Mealy machine M_2 equivalent to M_1.

Proof : We will discuss proof in two steps.

Step 1 : Construction of equivalent Mealy machine M_2, and

Step 2 : Outputs produced by both machines are equivalent.

Step 1(Construction of equivalent Mealy machine M_2)

Let $M_2 = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$ where all terms $Q, \Sigma, \Delta, \delta, q_0$ are same as for Moore machine and λ' is defined as following :

$$\lambda'(q, a) = \lambda(\delta(q, a)) \text{ for all } q \in Q \text{ and } a \in \Sigma$$

The first output produced by initial state of Moore machine is neglected and transition sequences remain unchanged.

Step 2 : If x is the output symbol produced by initial state of Moore machine M_1, and $T_1(w)$, $T_2(w)$ are outputs produced by Moore machine M_1 and equivalent Mealy machine M_2 respectively for input string w, then

$$T_1(w) = x T_2(w)$$

Or Output of Moore machine $= x \mid\mid$ Output of Mealy machine

(The notation $\mid\mid$ represents concatenation).

If we delete the output symbol x from $T_1(w)$ and suppose it is $T_1'(w)$ which is equivalent to the output of Mealy machine. So we have,

$$T_1'(w) = T_2(w)$$

Hence, Moore machine M_1 and Mealy machine M_2 are equivalent.

Example 1 : Construct a Mealy machine equivalent to Moore machine M_1 given in following transition table.

	Inputs		
	0	1	
Present State (PS)	Next State (NS)	Next State (NS)	Output
q_0	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

Solution : Let equivalent Mealy machine $M_2 = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$

where

1. $Q = \{q_0, q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. $\Delta = \{0, 1\}$
4. λ' is defined as following :

For state q_0: $\lambda'(q_0, 0) = \lambda(\delta(q_0, 0)) = \lambda(q_1) = 0$

$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1)) = \lambda(q_2) = 1$

For state q_1 : $\lambda'(q_1, 0) = \lambda(\delta(q_1, 0)) = \lambda(q_3) = 1$

$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1)) = \lambda(q_2) = 1$

For state q_2 : $\lambda'(q_2, 0) = \lambda(\delta(q_2, 0)) = \lambda(q_2) = 1$

$\lambda'(q_2, 1) = \lambda(\delta(q_2, 1)) = \lambda(q_1) = 0$

For state q_3: $\lambda'(q_3, 0) = \lambda(\delta(q_3, 0)) = \lambda(q_0) = 1$

$\lambda'(q_3, 1) = \lambda(\delta(q_3, 1)) = \lambda(q_3) = 1$

Transition table :

	Inputs			
	0		1	
PS	NS	Output	NS	Output
$\rightarrow q_0$	q_1	0	q_2	1
q_1	q_3	1	q_2	1
q_2	q_2	1	q_1	0
q_3	q_0	1	q_3	1

Transition diagram :

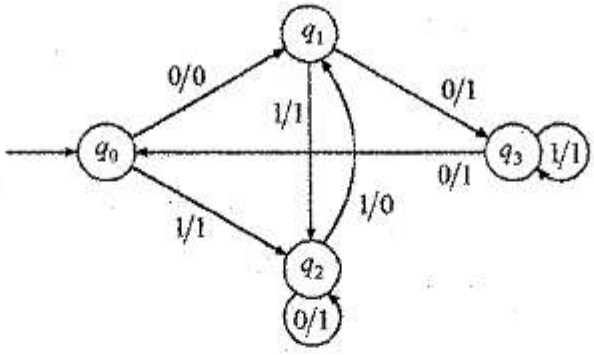


FIGURE : Mealy Machine

Example 2 : Construct a Mealy machine equivalent to Moore machine $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ described in following transition table.

	Inputs		
	0	1	
Present State (PS)	Next State (NS)	Next State (NS)	Output
q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

Solution : Let equivalent Mealy machine $M_2 = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$, where

1. $Q = \{q_0, q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. $\Delta = \{0, 1\}$
4. λ' is defined as following :

For state q_0: $\lambda'(q_0, 0) = \lambda(\delta(q_0, 0)) = \lambda(q_3) = 0$

$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1)) = \lambda(q_1) = 1$

For state $q_1 : \lambda'(q_1, 0) = \lambda(\delta(q_1, 0)) = \lambda(q_1) = 1$

$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1)) = \lambda(q_2) = 0$

For state $q_2 : \lambda'(q_2, 0) = \lambda(\delta(q_2, 0)) = \lambda(q_2) = 0$

$\lambda'(q_2, 1) = \lambda(\delta(q_2, 1)) = \lambda(q_3) = 0$

For state $q_3 : \lambda'(q_3, 0) = \lambda(\delta(q_3, 0)) = \lambda(q_3) = 0$

$\lambda'(q_3, 1) = \lambda(\delta(q_3, 1)) = \lambda(q_0) = 0$

5. Transition is same for both machines, and
6. q_0 is the initial state.

Transition table :

	Inputs 0		1	
PS	NS	Output	NS	Output
q_0	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_0	0

Conversion of Mealy Machine to Moore Machine

Theorem : If $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is a Mealy machine then there exists a Moore machine M_2 equivalent to M_1.

Proof : We will discuss proof in two steps.

Step 1 : Construction of equivalent Moore machine M_2, and

Step 2 : Outputs produced by both machines are equivalent.

Step 1 : Construction of equivalent Moore machine M_2

We define the set of states as ordered pair over Q and Δ. There is also a change in transition function and output function.

Let equivalent Moore machine $M_2 = (Q', \Sigma, \Delta, \delta', \lambda', q_0')$,

where

1. $Q' \subseteq Q \times \Delta$ is the set of states formed with ordered pair over Q and Δ,
2. Σ remains unchanged,

3. Δ remains unchanged.
4. λ' is defined as follows :

   $\delta'([q, b], a) = [\delta(q, a), \lambda(q, a)]$, where δ and λ are transition function and output function of Mealy machine.
5. λ' is the output function of equivalent Moore machine which is dependent on present state only and defined as follows :

$$\lambda'([q, b]) = b$$

6. q_0' is the initial state and defined as $[q_0, b_0]$, where q_0 is the initial state of Mealy machine and b_0 is any arbitrary symbol selected from output alphabet Δ.

Step 2 : Outputs of Mealy and Moore Machines

Suppose, Mealy machine M_1 enters states $q_0, q_1, q_2, \ldots q_n$ on input $a_1, a_2, a_3, \ldots a_n$ and produces outputs $b_1, b_2, b_3, \ldots b_n$, then M_2 enters the states $[q_0, b_0], [q_1, b_1], [q_2, b_2], \ldots, [q_n, b_n]$ and produces outputs $b_0, b_1, b_2, \ldots b_n$ as discussed in Step 1. Hence, outputs produced by both machines are equivalent.

Therefore, Mealy machine M_1 and Moore machine M_2 are equivalent.

Example 1 : Consider the Mealy machine shown in below figure. Construct an equivalent Moore machine.

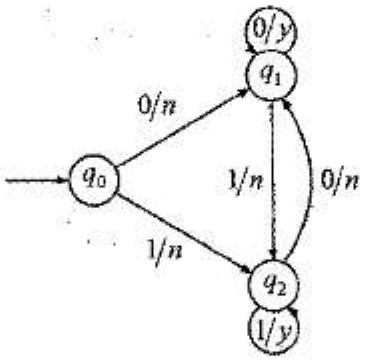


FIGURE : Mealy Machine

Solution : Let $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is a given Mealy machine and $M_2 = (Q', \Sigma, \Delta, \delta', \lambda', q_0')$ be the equivalent Moore machine,

where

1. $Q' \subseteq \{[q_0, n], [q_0, y], [q_1, n], [q_1, y], [q_2, n], [q_2, y]\}$ (Since, $Q' \subseteq Q \times \Delta$)
2. $\Sigma = \{0, 1\}$

3. $\Delta = \{n, y\}$,
4. $q_0' = [q_0, y]$, where q_0 is the initial state and y is the output symbol of Mealy machine,
5. δ' is defined as following :

For initial state $[q_0, y]$:

$$\delta'([q_0, y], 0) = [\delta(q_0, 0), \lambda(q_0, 0)] = [q_1, n]$$

$$\delta'([q_0, y], 1) = [\delta(q_0, 1), \lambda(q_0, 1)] = [q_2, n]$$

For state $[q_1, n]$:

$$\delta'([q_1, n], 0) = [\delta(q_1, 0), \lambda(q_1, 0)] = [q_1, y]$$

$$\delta'([q_1, n], 1) = [\delta(q_1, 1), \lambda(q_1, 1)] = [q_2, n]$$

For state $[q_2, n]$:

$$\delta'([q_2, n], 0) = [\delta(q_2, 0), \lambda(q_2, 0)] = [q_1, n]$$

$$\delta'([q_2, n], 1) = [\delta(q_2, 1), \lambda(q_2, 1)] = [q_2, y]$$

For state $[q_1, y]$:

$$\delta'([q_1, y], 0) = [\delta(q_1, 0), \lambda(q_1, 0)] = [q_1, y]$$

$$\delta'([q_1, y], 1) = [\delta(q_1, 1), \lambda(q_1, 1)] = [q_2, n]$$

For state $[q_2, y]$:

$$\delta'([q_2, y], 0) = [\delta(q_2, 0), \lambda(q_2, 0)] = [q_1, n]$$

$$\delta'([q_2, y], 1) = [\delta(q_2, 1), \lambda(q_2, 1)] = [q_2, y]$$

(Note : We have considered only those states, which are reachable from initial state)

6. λ' is defined as follows :

$$\lambda'[q_0, y] = y$$

$$\lambda'[q_1, n] = n$$

$$\lambda'[q_2, n] = n$$

$$\lambda'[q_1, y] = y$$

$$\lambda'[q_2, y] = y$$

Transition diagram :

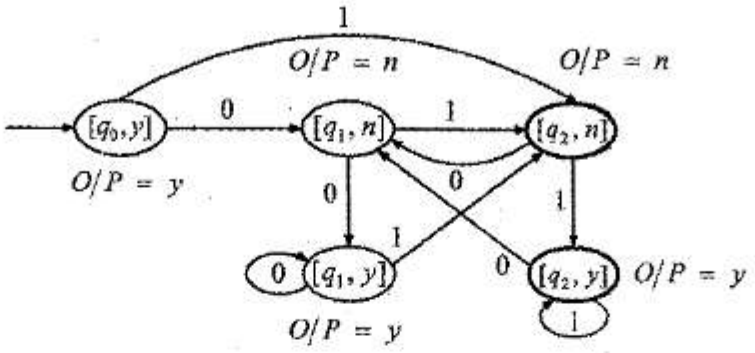


FIGURE : Moore machine

Example 2 : Construct a Moore machine equivalent to Mealy machine $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ described in following transition table

	Inputs			
	0		1	
PS	NS	Output	NS	Output
q_0	q_1	z_1	q_2	z_1
q_1	q_1	z_2	q_2	z_1
q_2	q_1	z_1	q_2	z_2

Solution :

Let $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is given Mealy machine and $M_2 = (Q', \Sigma, \Delta', \delta', \lambda', q_0')$ be the equivalent Moore machine, where

1. $Q' \subseteq \{[q_0, z_1], [q_0, z_2], [q_1, z_1], [q_1, z_2], [q_2, z_1], [q_2, z_2]\}$ (Since, $Q' \subseteq Q \times \Delta$)
2. $\Sigma = \{0, 1\}$
3. $\Delta' = \{z_1, z_2\}$
4. Let starting state $q_0' = [q_0, z_1]$ where q_0 is the initial state and z_1 is the output symbol of Mealy machine,

5. δ' is defined as follows :

For initial state $[q_0, z_1]$: $\delta'([q_0, z_1], 0) = [\delta(q_0, 0), \lambda(q_0, 0)] = [q_1, z_1]$

$\delta'([q_0, z_1], 1) = [\delta(q_0, 1), \lambda(q_0, 1)] = [q_2, z_1]$

(**Note :** Both states $[q_1, z_1]$ and $[q_2, z_1]$ are reachable from initial state.)

For state $[q_1, z_1]$: $\delta'([q_1, z_1], 0) = [\delta(q_1, 0), \lambda(q_1, 0)] = [q_1, z_2]$

$\delta'([q_1, z_1], 1) = [\delta(q_1, 1), \lambda(q_1, 1)] = [q_2, z_1]$

For state $[q_2, z_1]$: $\delta'([q_2, z_1], 0) = [\delta(q_2, 0), \lambda(q_2, 0)] = [q_1, z_1]$

$\delta'([q_2, z_1], 1) = [\delta(q_2, 1), \lambda(q_2, 1)] = [q_2, z_2]$

(**Note :** Both states $[q_1, z_2]$ and $[q_2, z_2]$ are reachable states.)

For state $[q_1, z_2]$: $\delta'([q_1, z_2], 0) = [\delta(q_1, 0), \lambda(q_1, 0)] = [q_1, z_2]$

$\delta'([q_1, z_2], 1) = [\delta(q_1, 1), \lambda(q_1, 1)] = [q_2, z_1]$

For state $[q_2, z_2]$: $\delta'([q_2, z_2], 0) = [\delta(q_2, 0), \lambda(q_2, 0)] = [q_1, z_1]$

$\delta'([q_2, z_2], 1) = [\delta(q_2, 1), \lambda(q_2, 1)] = [q_2, z_2]$

(**Note :** We have considered only those states, which are reachable from initial state.)

6. λ' is defined as follows :

$\lambda'[q_0, z_1] = z_1$

$\lambda'[q_1, z_1] = z_1$

$\lambda'[q_2, z_1] = z_1$

$\lambda'[q_1, z_2] = z_2$

$\lambda'[q_2, z_2] = z_2$

Transition Table

	Inputs		
	0	1	
PS	NS	NS	Output
$[q_0, z_1]$	$[q_1, z_1]$	$[q_2, z_1]$	z_1
$[q_1, z_1]$	$[q_1, z_2]$	$[q_2, z_1]$	z_1
$[q_2, z_1]$	$[q_1, z_1]$	$[q_2, z_2]$	z_1
$[q_1, z_2]$	$[q_1, z_2]$	$[q_2, z_1]$	z_2
$[q_2, z_2]$	$[q_1, z_1]$	$[q_2, z_2]$	z_2

2.5 EQUIVALENCE OF FSMs

Two finite machines are said to be equivalent if and only if every input sequence yields identical output sequence.

Example :

Consider the FSM M_1 shown in figure (a) and FSM M_2 shown in figure (b).

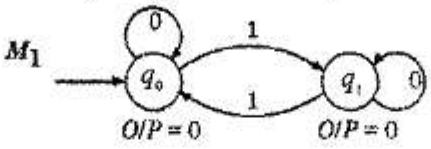


Figure (a)

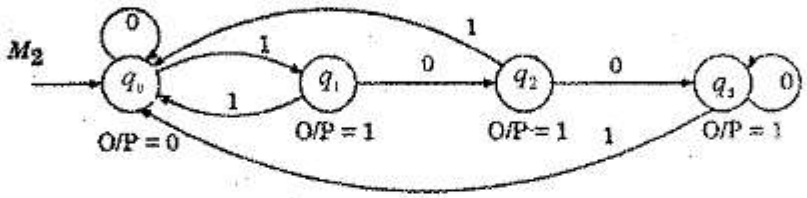


Figure (b)

Are these two FSMs equivalent ?

Solution :

We check this. Consider the input strings and corresponding outputs as given following :

Input string	Output by M_1	Output by M_2
(1) 01	00	00
(2) 010	001	001
(3) 0101	0011	0011
(4) 1000	0111	0111
(5) 10001	01111	01111

Now, we come to this conclusion that for each input sequence, outputs produced by both machines are identical. So, these machines are equivalent. In other words, both machines do the same task. But, M_1 has two states and M_2 has four states. So, some states of M_2 are doing the same

task i. e., producing identical outputs on certain input. Such states are known as equivalent states and require extra resources when implemented.
Thus, our goal is to find the simplest and equivalent FSM with minimum number of states.

2.5.1 FSM Minimization

We minimize a FSM using the following method, which finds the equivalent states, and merges these into one state and finally construct the equivalent FSM by minimizing the number of states.

Method : Initially we assume that all pairs (q_0, q_1) over states are non - equivalent states

Step 1 : Construct the transition table.

Step 2 : Repeat for each pair of non - equivalent states (q_0, q_1) :

(a) Do q_0 and q_1 produce same output ?

(b) Do q_0 and q_1 reach the same states for each input $a \in \Sigma$?

(c) If answers of (a) and (b) are YES, then q_0 and q_1 are equivalent states and merge these two states into one state $[q_0, q_1]$ and replace the all occurrences of q_0 and q_1 by $[q_0, q_1]$ and mark these equivalent states.

Step 3 : Check the all - present states, if any redundancy is found, remove that.

Step 4 : Exit.

Example 1 : Consider the following transition table for FSM. Construct minimum state FSM.

	Inputs		
	0	1	
Present State(PS)	**Next State (NS)**	**Next State (NS)**	**Output**
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_3	q_0	1
q_3	q_3	q_0	1

Solution :

Pairs formed over $\{q_0, q_1, q_2, q_3\}$ are $(q_0,q_1), (q_0,q_2), (q_0,q_3), (q_1,q_2), (q_1,q_3), (q_2,q_3)$.

Consider the pair (q_0,q_1) :

$$\lambda(q_0) = 0$$
$$\lambda(q_1) = 1$$

Hence, q_0 and q_1 are not equivalent.

Consider the pair (q_0,q_2) :

$$\lambda(q_0) = 0$$
$$\lambda(q_2) = 1$$

Hence, q_0 and q_2 are not equivalent

Consider the pair (q_0,q_3) :

$$\lambda(q_0) = 0$$
$$\lambda(q_3) = 1$$

Hence, q_0 and q_3 are not equivalent

Consider the pair (q_1,q_2) :

$$\lambda(q_1) = 1$$
$$\lambda(q_2) = 1$$

Outputs are identical.

Now, consider the transition :

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

So, transitions from q_1 and q_2 are not on the same state for 0 input.

Hence, q_1 and q_2 are not equivalent

Consider the pair (q_1,q_3) :

$$\lambda(q_1) = 1$$
$$\lambda(q_3) = 1$$

Outputs are identical.

Now, consider the transition :

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

So, transitions from q_1 and q_3 are not on the same state for 0 input.

Hence, q_1 and q_3 are not equivalent.

Consider the pair (q_2,q_3) :

$$\lambda(q_2)= 1$$
$$\lambda(q_3)= 1$$

Outputs are identical.

Now, consider the transition :

$$\delta(q_2,0)= q_3, \quad \delta(q_2,1)= q_0$$
$$\delta(q_3,0)= q_3, \quad \delta(q_3,1)= q_0$$

So, transitions from q_2 and q_3 are identical for inputs 0 and 1.

Hence, q_2 and q_3 are equivalent states.

So, merging q_2 and q_3 into $[q_2,q_3]$ to represent one state and replacing q_2 and q_3 by $[q_2,q_3]$, we have following intermediate transition table 1.

Intermediate transition table 1

Present State (PS)	Inputs 0 Next State (NS)	Inputs 1 Next State (NS)	Output
$\rightarrow q_0$	q_0	q_1	0
q_1	$[q_2,q_3]$	q_0	1
$[q_2,q_3]$	$[q_2,q_3]$	q_0	1
$[q_2,q_3]$	$[q_2,q_3]$	q_0	1

Applying Step 2 further on intermediate transition table we see that q_1,$[q_2,q_3]$ are equivalent states.

So, replacing q_1 and $[q_2,q_3]$ by $[q_1,q_2,q_3]$, we have intermediate transition table 2.

Intermediate transition table 2

Present State (PS)	Inputs 0 Next State (NS)	Inputs 1 Next State (NS)	Output
$\rightarrow q_0$	q_0	$[q_1,q_2,q_3]$	0
$[q_1,q_2,q_3]$	$[q_1,q_2,q_3]$	q_0	1
$[q_1,q_2,q_3]$	$[q_1,q_2,q_3]$	q_0	1
$[q_1,q_2,q_3]$	$[q_1,q_2,q_3]$	q_0	1

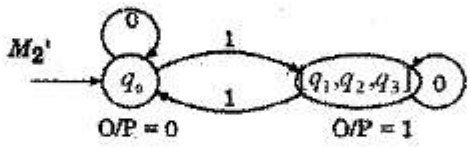
Applying Step 3 and removing redundancy, we have to delete two rows.

Now, we have the following final transition table 3 :

Transition table 3

Present State (PS)	Inputs 0 Next State (NS)	Inputs 1 Next State (NS)	Output
$\rightarrow q_0$	q_0	$[q_1,q_2,q_3]$	0
$[q_1,q_2,q_3]$	$[q_1,q_2,q_3]$	q_0	1

Transition diagram :



Example 2 : Consider the following transition table of a Mealy machine. Construct minimum state Mealy machine.

PS	Inputs 0 N S	Output	1 N S	Output
$\rightarrow q_0$	q_0	0	q_1	0
q_1	q_0	0	q_2	1
q_2	q_0	0	q_2	1

Solution : Last two rows of transition table show that states q_1 and q_2 are equivalent states. So, replacing these states by $\{q_1,q_2\}$, we have the following intermediate transition table.

P S	Inputs 0		1	
	N S	Output	N S	Output
$\rightarrow q_0$	q_0	0	$[q_1,q_2]$	0
$[q_1,q_2]$	q_0	0	$[q_1,q_2]$	1
$[q_1,q_2]$	q_0	0	$[q_1,q_2]$	1

Deleting the last row, we have the following final transition table.

P S	Inputs 0		1	
	N S	Output	N S	Output
$\rightarrow q_0$	q_0	0	$[q_1,q_2]$	0
$[q_1,q_2]$	q_0	0	$[q_1,q_2]$	1

REVIEW QUESTIONS

Q1. Define and explain about Moore Machine.

Answer :

For Answer refer to Topic : 2.2, Page No : 2.3.

Q2. Consider the Moore machine shown in below figure.Construct the transition table. What is the output for input 01010 ?

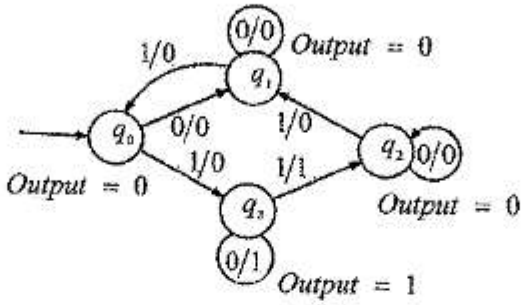


FIGURE: Moore machine

Answer :

For Answer refer to example - 1 , Page No : 1.2.4.

Q3. Design a Moore machine, which outputs residue mod 3 for each binary input string treated as a binary integer.

Answer :

For Answer refer to example - 2 , Page No : 2.5.

Q4. Design a Moore machine which reads input from (0+1+2)* and outputs residue mod 5 of the input. Input is considered at base 3 and it is treated as ternary integer.

Answer :

For Answer refer to example - 3 , Page No : 2.6.

Q5. Define and explain about Mealy Machine .

Answer :

For Answer refer to Topic : 2.3 , Page No : 2.7.

Q6. Consider the Mealy machine shown in below figure. Construct the transition table and find the output for input 01010.

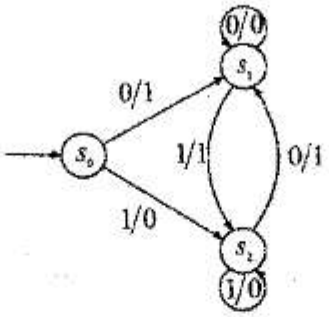


FIGURE : Mealy Machine

Answer :

For Answer refer to example - 1 , Page No : 2.8.

Q7. Construct a Mealy machine which reads input from {0, 1} and outputs EVEN or ODD according to total number of 1's even or odd.

Answer :

For Answer refer to example - 2 , Page No : 2.9.

Q8. Design a Mealy machine which reads the input from (0+1)* and produces the following outputs.

(i) If input ends in 101, output is A,

(ii) If input ends 110, the output is B, and

(iii) For other inputs, output is C.

Answer :

For Answer refer to example - 3 , Page No : 2.9.

Q9. Explain conversion of Moore Machine to Mealy Machine.

Answer :

For Answer refer to Theorem, Page No : 2.11.

Q10. Construct a Mealy machine equivalent to Moore machine M_1 given in following transition table.

	Inputs		
	0	1	
Present State (PS)	Next State (NS)	Next State (NS)	Output
q_0	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

Answer :

For Answer refer to example - 1 , Page No : 2.11.

Q11. Construct a Mealy machine equivalent to Moore machine $M_1 = (Q, \Sigma, \Lambda, \delta, \lambda, q_0)$ described in following transition table.

	Inputs		
	0	1	
Present State (PS)	Next State (NS)	Next State (NS)	Output
q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

Answer :

For Answer refer to example - 2 , Page No : 2.13.

Q12. Explain conversion of mealy machine to moore machine.

Answer :

For Answer refer to Theorem , Page No : 2.14.

Q13. Consider the Mealy machine shown in below figure. Construct an equivalent Moore machine.

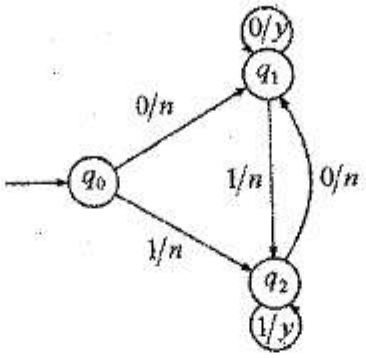


FIGURE : Mealy Machine

Answer :

For Answer refer to example - 1 , Page No : 2.15.

Q14. Construct a Moore machine equivalent to Mealy machine $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ described in following transition table

	Inputs			
	0		1	
PS	NS	Output	NS	Output
q_0	q_1	z_1	q_2	z_1
q_1	q_3	z_2	q_3	z_3
q_2	q_1	z_1	q_2	z_2

Answer :

For Answer refer to example - 2 , Page No : 2.17.

Q15. Explain about equivalence of two FSMs with an example.

Answer :

For Answer refer to Topic : 2.5, Page No : 2.19.

Q16. Explain procedure for FSM minimization.

Answer :

For Answer refer to Topic : 2.5.1, Page No : 2.20.

Q17. Consider the following transition table for FSM. Construct minimum state FSM.

	Inputs		
	0	1	
Present State(PS)	**Next State (NS)**	**Next State (NS)**	**Output**
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_3	q_0	1
q_3	q_3	q_0	1

Answer :

For Answer refer to example - 1 , Page No : 2.20.

Q18. Consider the following transition table of a Mealy machine. Construct minimum state Mealy machine.

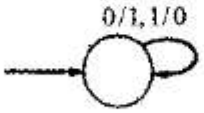
	Inputs			
	0		1	
P S	**N S**	**Output**	**N S**	**Output**
$\rightarrow q_0$	q_0	0	q_1	0
q_1	q_0	0	q_2	1
q_2	q_0	0	q_2	1

Answer :

For Answer refer to example - 2 , Page No : 2.23.

OBJECTIVE TYPE QUESTIONS

1. The automata in which the output depends only on the status of the machine is called:
   (a) Moore machine (b) Mealy machine
   (c) Any finite automata (d) both (a) &(c)
2. Choose the correct statement
   (a) A Mealy machine has no terminal state
   (b) A Mealy machine generates no language as scuh
   (c) A Moore machine generates no language as such (d) All.
3. Choose the correct statement
   (a) A Mealy machine has no terminal state
   (b) A Mealy machine generates no language as such
   (c) A Moore machine generates no language as such
   (d) All
4. The major difference between a Moore and Mealy machine is
   (a) Output of Moore depends on output only
   (b) Output of Moore depends on present state and input
   (c) Output of Moore depends on state only
   (d) None.
5. Choose the correct statements
   (a) Any given Mealy machine has an equivalent Moore machine
   (b) Moore and Mealy machines are finite state machines with output capability
   (c) Any given Moore machine has an equivalent Mealy machine
   (d) All.
6. The finite state machine in below figure is a



   (a) Kleene machine (b) Mealy machine
   (c) Moore machine (d) none of the above

7. Moore machine is
(a) Automaton in which the output depends only on the state and the input.
(b) Automaton in which the output depends only on the states
(c) Automaton in which the output depends only on the input
(d) None of the above.

ANSWER KEY

1(b) 2 (a) 3(a) 4(c) 5(d) 6(b) 7 (b)

Formal Languages And Automata Theory

UNIT 2

COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF AERONAUTICAL ENGINEERING

DUNDIGAL, HYDERABAD - 500 043

3

REGULAR LANGUAGES AND FINITE AUTOMATA

After going through this chapter, you should be able to understand :

- Regular sets and Regular Expressions
- Identity Rules
- Constructing FA for a given REs
- Conversion of FA to REs
- Pumping Lemma of Regular sets
- Closure properties of Regular sets

3.1 REGULAR SETS

A special class of sets of words over S, called regular sets, is defined recursively as follows. (Kleene proves that any set recognized by an FSM is regular. Conversely, every regular set can be recognized by some FSM.)

1. Every finite set of words over S (including ϵ, the empty set) is a regular set.
2. If A and B are regular sets over S, then $A \cup B$ and AB are also regular.
3. If S is a regular set over S, then so is its closure S*.
4. No set is regular unless it is obtained by a finite number of applications of definitions (1) to (3).

i.e., the class of regular sets over S is the smallest class containing all finite sets of words over S and closed under union, concatenation and star operation.

Examples:

i) Let $\Sigma = \{a,b\}$ then the set of strings that contain both odd number of a's and b's is a regular set.

ii) Let $\Sigma = \{0\}$ then the set of strings $\{0,00,000,.....\}$ is a regular set.

iii) Let $\Sigma = \{0,1\}$ then the set of strings $\{01,10\}$ is a regular set.

3.2 REGULAR EXPRESSIONS

The languages accepted by FA are regular languages and these languages are easily described by simple expressions called regular expressions. We have some algebraic notations to represent the regular expressions.

Regular expressions are means to represent certain sets of strings in some algebraic manner and regular expressions describe the language accepted by FA.

If Σ is an alphabet then regular expression(s) over this can be described by following rules.

1. Any symbol from $\Sigma, \in$ *and* ϕ are regular expressions.
2. If r_1 and r_2 are two regular expressions then *union* of these represented as $r_1 \cup r_2$ or $r_1 + r_2$ is also a regular expression
3. If r_1 and r_2 are two regular expressions then *concatenation* of these represented as $r_1 r_2$ is also a regular expression.
4. The Kleene closure of a regular expression r is denoted by r^* is also a regular expression.
5. If r is a regular expression then (r) is also a regular expression.
6. The regular expressions obtained by applying rules 1 to 5 once or more than once are also regular expressions.

Examples :

(1) If $\Sigma = \{a, b\}$, then

(a) a is a regular expression (Using rule 1)
(b) b is a regular expression (Using rule 1)
(c) $a + b$ is a regular expression (Using rule 2)
(d) b^* is a regular expression (Using rule 4)
(e) ab is a regular expression (Using rule 3)
(f) $ab + b^*$ is a regular expression (Using rule 6)

(2) Find regular expression for the following

(a) A language consists of all the words over $\{a, b\}$ ending in b.
(b) A language consists of all the words over $\{a, b\}$ ending in bb.
(c) A language consists of all the words over $\{a, b\}$ starting with a and ending in b.
(d) A language consists of all the words over $\{a, b\}$ having bb as a substring.
(e) A language consists of all the words over $\{a, b\}$ ending in aab.

Solution : Let $\Sigma = \{a, b\}$, and

All the words over $\Sigma = \{\in, a, b, aa, bb, ab, ba, aaa, \ldots\} = \Sigma^*$ *or* $(a + b)^*$ *or* $(a \cup b)^*$

(a) Regular expression for the given language is $(a + b)^* b$

(b) Regular expression for the given language is $(a + b)^* bb$

(c) Regular expression for the given language is $a\,(a + b)^* b$

(d) Regular expression for the given language is $(a + b)^* aa$ or $aa\,(a + b)^*$ or $(a + b)^* bb\,(a + b)^*$

(e) Regular expression for the given language is $(a + b)^* aab$

The table below shows some examples of regular expressions and the language corresponding to these regular expressions.

Regular expression	Meaning
(a + b)*	Set of strings of a's and b's of any length including the NULL string.
(a + b)*abb	Set of strings of a's and b's ending with the string abb.
ab (a + b)*	Set of strings of a's and b's starting with the string ab.
(a + b)*aa (a + b)*	Set of strings of a's and b's having a sub string aa.
a*b*c*	Set of strings consisting of any number of a's (may be empty string also) followed by any number of b's (may include empty string) followed by any number of c's (may include empty string).
$a^+b^+c^+$	Set of strings consisting of at least one 'a' followed by string consisting of at least one 'b' followed by string consisting of at least one 'c'.
aa*bb*cc*	Set of strings consisting of at least one 'a' followed by string consisting of at least one 'b' followed by string consisting of at least one 'c'.
(a + b)* (a + bb)	Set of strings of a's and b's ending with either a or bb.
(aa) * (bb)*b	Set of strings consisting of even number of a's followed by odd number of b's.
(0 + 1) * 000	Set of strings of 0's and 1's ending with three consecutive zeros(or ending with 000)
(11)*	Set consisting of even number of 1's

TABLE: Meaning of regular expressions

Example 1 : Obtain a regular expression to accept a language consisting of strings of a's and b's of even length.

Solution :

String of a's and b's of even length can be obtained by the combination of the strings aa, ab, ba, and bb. The language may even consist of an empty string denoted by $\in$. So, the regular expression can be of the form

$$(aa + ab + ba + bb)^*$$

The * closure includes the empty string.

Note : This regular expression can also be represented using set notation as

$$L(r)=\{(aa + ab + ba + bb)^n | n \geq 0\}$$

Example 2 : Obtain a regular expression to accept a language consisting of strings of a's and b's of odd length.

Solution :

String of a's and b's of odd length can be obtained by the combination of the strings aa, ab, ba and bb followed by either a or b. So, the regular expression can be of the form

$$(aa+ab+ba+bb)^* (a+b)$$

String of a's and b's of odd length can also be obtained by the combination of the strings aa, ab, ba and bb preceded by either a or b. So, the regular expression can also be represented as

$$(a+b)(aa+ab+ba+bb)^*$$

Note : Even though these two expressions are seems to be different, the language corresponding to those two expressions is same. So, a variety of regular expressions can be obtained for a language and all are equivalent.

Example 3 : Obtain a regular expression such that $L(r) = \{W \mid W \in \{0,1\}^*$ with at least three consecutive 0's $\}$.

Solution :

An arbitrary string consisting of 0's and 1's can be represented by the regular expression.

$$(0 + 1)^*$$

This arbitrary string can precede three consecutive zeros and can follow three consecutive zeros. So, the regular expression can be written as

$$(0+1)^*000(0+1)^*$$

Note : Using the set notation the regular expression can be written as

$$L(r) = \{(0 + 1)^m 000 (0 + 1)^n | m \geq 0 \text{ and } n \geq 0\}$$

Example 4 : Obtain a regular expression to accept strings of a's and b's ending with 'b' and has no substring aa.

Solution :

Note : The statement "strings of a's and b's ending with 'b' and has no substring aa" can be restated as "string made up of either b or ab". Note that if we state something like this, the substring aa will never occur in the string and the string ends with 'b'. So, the regular expression can be of the form

$$(b + ab)^*$$

But, because of * closure, even null string is also included. But, the string should end with 'b'. So, instead of * closure, we can use positive closure '+'. So, the regular expression to accept strings of a's and b's ending with 'b' and has no substring aa can be written as

$$(b+ab)^+$$

The above regular expression can also be written as

$$(b+ab)\,(b+ab)^*$$

Note : Using the set notation this regular expression can be written as

$$L(r)=\{(b + ab)^n | n\geq 1\}$$

Example 5 : Obtain a regular expression to accept strings of 0's and 1's having no two consecutive zeros.

Solution :

The first observation from the statement is that whenever a 0 occurs it should be followed by 1. But, there is no restriction on the number of 1's. So , it is a string consisting of any combination of 1's and 01's. So, the partial regular expression for this can be of the form

$$(1+01)^*$$

No doubt that the above expression is correct. But, suppose the string ends with a 0. What to do? For this, the string obtained from above regular expression may end with 0 or may end with ϵ (i. e., may not end with 0). So, the above regular expression can be written as

$$(1+01)^+(0+\epsilon)$$

Example 6 : Obtain a regular expression to accept strings of a's and b's of length ≤ 10.

Solution :

The regular expression for this can be written as

$$\in + a + b + aa + ab + ba + bb + \ldots\ldots + bbbbbbbbba + bbbbbbbbbb$$

But, using ..... in a regular expression is not recommended and so we can write the above expression as

$$(\in + a + b)^{10}$$

Example 7 : Obtain a regular expression to accept strings of a's and b's starting with 'a' and ending with 'b'.

Solution :

Strings of a's and b's of arbitrary length can be written as $(a+b)^*$

But, this should start with 'a' and end with 'b'. So, the regular expression can be written as

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

Hierarchy of Evaluation of Regular Expressions

We follow the following order when we evaluate a regular expression.

1. Parenthesis
2. Kleene closure
3. Concatenation
4. Union

Example 1: Consider the regular expression $(a+b)^* aab$ and describe the all words represented by this.

Solution :

$(a+b)^* aab$ = {All words over $\{a,b\}$} aab (Evaluating $(a+b)^*$ first)

= $\{\in, a, b, aa, bb, ab, ba, aaa, \ldots\}\, aab$

= {All words over $\{a, b\}$ ending in aab}

Example 2: Consider the regular expression $(a^* + b^*)^*$ and explain it.

Solution : We evaluate a^* and b^* first then $(a^* + b^*)^*$.

$(a^* + b^*)^*$ = (All the words over $\{a\}$ + all the words over $\{b\}$)*

= ($\{\in, a, aa, \ldots\}$ or $\{\in, b, bb, \ldots\}$)*

$= (\{\in, a, b, aa, bb, \ldots\})^*$

$= \{\in, a, b, aa, bb, ab, ba, aaa, bbb, abb, baa, aabb, \ldots\}$

= {All the words over $\{a, b\}$ }

$\equiv (a + b)^*$

So, $(a^* + b^*)^* = (a + b)^*$

3.3 IDENTITIES FOR REs

The two regular expressions P and Q are equivalent (denoted as P = Q) if and only if P represents the same set of strings as Q does. For showing this equivalence of regular expressions we need to show some identities of regular expressions.

Let P, Q and R are regular expressions then the identity rules are as given below

1. $\in R = R \in = R$
2. $\in^* = \in$ $\in$ is null string
3. $(\phi)^* = \in$ ϕ is empty string.
4. $\phi R = R\phi = \phi$
5. $\phi + = R = R$
6. $R + R = R$
7. $RR^* = R^* R = R^+$
8. $(R^*)^* = R^*$
9. $\in + RR^* = R^*$
10. $(P + Q)R = PR + QR$
11. $(P + Q)^* = (P^* Q^*) = (P^* + Q^*)^*$
12. $R^*(\in + R) = (\in + R)R^* = R^*$
13. $(R + \in)^* = R^*$
14. $\in + R^* = R^*$
15. $(PQ)^* P = P(QP)^*$
16. $R^* R + R = R^* R$

3.3.1 Equivalence of two REs

Let us see one important theorem named Arden's Theorem which helps in checking the equivalence of two regular expressions.

Arden's Theorem : Let P and Q be the two regular expressions over the input set Σ. The regular expression R is given as

$$R = Q + RP$$

Which has a unique solution as $R = QP^*$

Proof : Let, P and Q are two regular expressions over the input string Σ.
If P does not contain ϵ then there exists R such that

$$R = Q + RP \qquad \ldots (1)$$

We will replace R by QP* in equation 1.
Consider R. H. S. of equation 1.

$$= Q + QP^*P$$

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

$$= QP^* \qquad \because \epsilon + R^*R = R^*$$

Thus $R = QP^*$

is proved. To prove that $R = QP^*$ is a unique solution, we will now replace L.H.S. of equation 1 by Q + RP. Then it becomes

$$Q + RP$$

But again R can be replaced by Q + RP.

$$\therefore \quad Q + RP = Q + (Q + RP)P$$

$$= Q + QP + RP^2$$

Again replace R by Q + RP.

$$= Q + QP + (Q + RP)P^2$$

$$= Q + QP + QP^2 + RP^3$$

Thus if we go on replacing R by Q + RP then we get,

$$Q + RP = Q + QP + QP^2 + \ldots\ldots + QP^i + RP^{i+1}$$

$$= Q(\epsilon + P + P^2 + \ldots\ldots P^i) + RP^{i+1}$$

From equation 1,

$$R = Q(\epsilon + P + P^2 + \ldots\ldots + P^i) + RP^{i+1} \qquad \ldots (2)$$

Where $i \geq 0$

Consider equation 2,

$$R = Q\underbrace{(\epsilon + P + P^2 + \ldots\ldots + P^i)}_{P^*} + RP^{i+1}$$

$$\therefore \quad R = QP^* + RP^{i+1}$$

Let w be a string of length i.

In RP^{i+1} has no string of less than i + 1 length. Hence w is not in set RP^{i+1}. Hence R and QP^* represent the same set. Hence it is proved that

$R = Q + RP$ has a unique solution.

$R = QP^*$.

Example 1 : Prove $(1+00*1)+(1+00*1)(0+10*1)*(0+10*1) = 0*1(0+10*1)*$

Solution : Let us solve L.H.S. first,

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

We will take $(1+00*1)$ as a common factor

$$1+00*1\ \underbrace{(\in +(0+10*1)*(0+10*1))}$$

$(\in +R*R)$ where $R = (0+10*1)$

As we know, $(\in +R*R) = (\in +RR*) = R*$

$\therefore (1+00*1)((0+10*1)*)$ out of this consider

$$\underbrace{(1+00*1)}\ (0+10*1)*$$

Taking 1 as a common factor

$(\in +00*)1(0+10*1)*$

Applying $\in +00* = 0*$

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

= R. H. S.

Hence the two regular expressions are equivalent.

Example 2 : Show that $(0*1*)* = (0+1)*$

Solution : Consider L. H. S.

$= (0*1*)*$

$= \{\in, 0, 00, 1, 11, 111, 01, 10, \ldots\ldots\}$

= { any combination of 0's, any combination of 1's, any combination of 0 and 1, $\in$ }

Similarly,

R. H. S.

$= (0+1)*$

$= \{\epsilon, 0, 00, 1, 11, 111, 01, 10, \ldots\ldots\}$

$= \{\ \epsilon$, any combination of 0's, any combination of 1's, any combination of 0 and 1 $\}$

Hence, L. H. S. = R. H. S. is proved.

3.4 RELATIONSHIP BETWEEN FA AND RE

There is a close relationship between a finite automata and the regular expression we can show this relation in below figure.

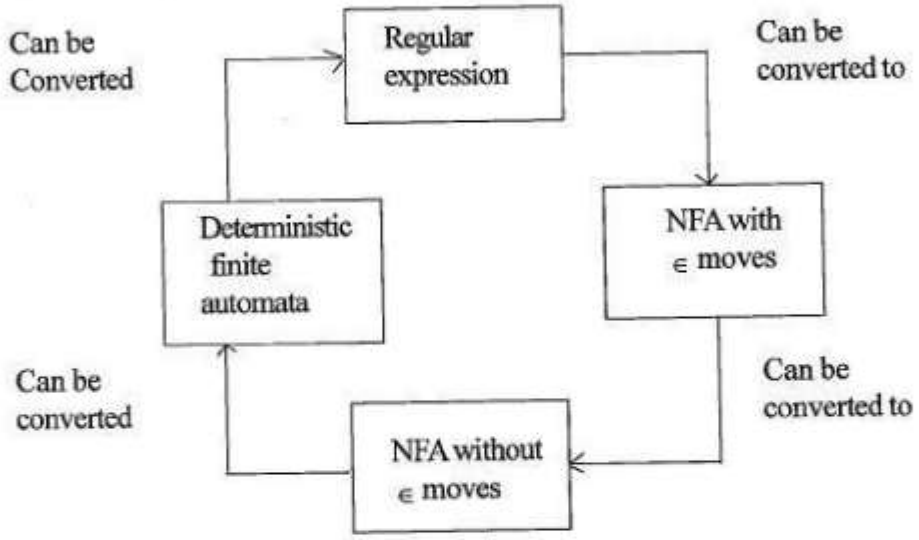


FIGURE : Relationship between FA and regular expression

The above figure shows that it is convenient to convert the regular expression to NFA with ϵ moves. Let us see the theorem based on this conversion.

3.5 CONSTRUCTING FA FOR A GIVEN REs

Theorem : If r be a regular expression then there exists a NFA with ϵ - moves, which accepts $L(r)$.

Proof : First we will discuss the construction of NFA M with ϵ - moves for regular expression r and then we prove that $L(M) = L(r)$.

Let r be the regular expression over the alphabet Σ.

Construction of NFA with ϵ - moves

Case 1 :

(i) $r = \phi$

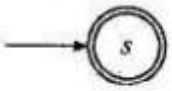
NFA $M = (\{s, f\}, \{\}, \delta, s, \{f\})$ as shown in Figure1 (a)

(No path from initial state s to reach the final state f.)

Figure 1 (a)

(ii) $r = \in$

NFA $M = (\{s\}, \{\}, \delta, s, \{s\})$ as shown in Figure 1 (b)

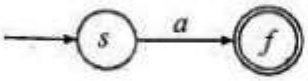


(The initial state s is the final state)

Figure 1 (b)

(iii) $r = a$, for all $a \in \Sigma$,

NFA $M = (\{s, f\}, \Sigma, \delta, s, \{f\})$



(One path is there from initial state s to reach the final state f with label a.)

Figure 1 (c)

Case 2 : $|r| \geq 1$

Let r_1 and r_2 be the two regular expressions over Σ_1, Σ_2 and N_1 and N_2 are two NFA for r_1 and r_2 respectively as shown in Figure 2 (a).

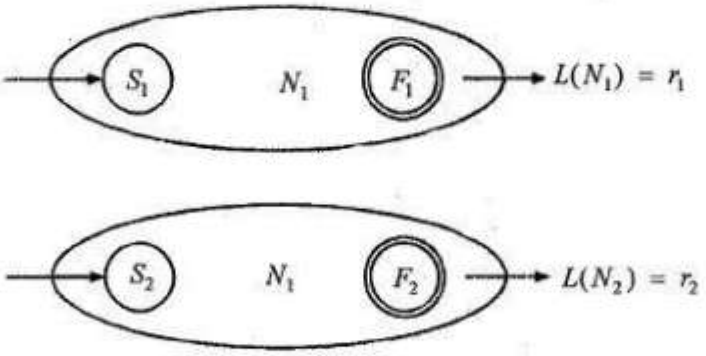


Figure 2 (a) NFA for regular expression r_1 and r_2

Rule 1 : For constructing NFA M for $\mathbf{r} = \mathbf{r}_1 + \mathbf{r}_2$ or $\mathbf{r}_1 \cup \mathbf{r}_2$

Let s and f are the starting state and final state respectively of M.

Transition diagram of M is shown in Figure 2 (b).

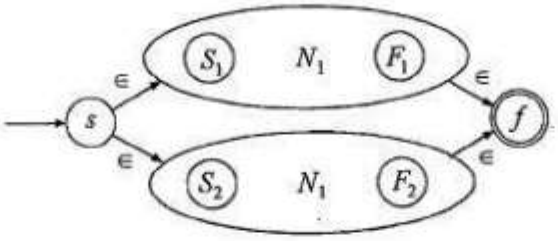


Figure 2 (b) NFA for regular expression $r_1 + r_2$

$$L(M) = \in L(N_1) \in \text{ or } \in L(N_2) \in$$
$$= L(N_1) \text{ or } L(N_2) = r_1 \text{ or } r_2$$

So, $r = r_1 + r_2$

$M = (Q, \Sigma_1 \cup \Sigma_2, \delta, s, \{f\})$, where Q contains all the states of N_1 and N_2.

Rule 2 : For regular expression $\mathbf{r} = \mathbf{r}_1\mathbf{r}_2$, NFA M is shown in Figure2 (c).

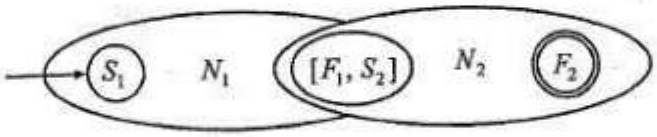


Figure 2 (c) NFA for regular expression r_1r_2

The final state (s) of N_1 is merged with initial state of N_2 into one state $[F_1S_2]$ as shown above in Figure2 (c).

$$L(M) = L(N_1) \text{ followed } L(N_2)$$
$$= L(N_1)\ L(N_2) = r_1r_2$$

So, $r = r_1r_2$

$M = (Q, \Sigma_1 \cup \Sigma_2, \delta, S_1, \{F_2\})$, where Q contains all the states of N_1 and N_2 such that final state(s) of N_1 is merged with initial state of N_2.

Rule 3 : For regular expression $\mathbf{r} = \mathbf{r_1^*}$, NFA M is shown in Figure2 (d)

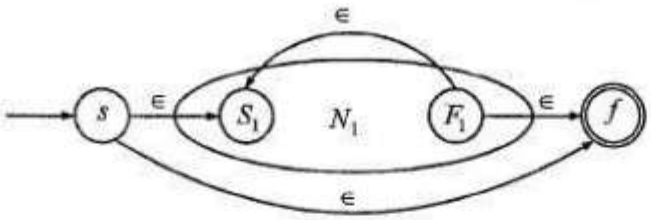


Figure 2 (d) NFA for regular expression for r_1^*

$$L(M) = \{\in, L(N_1), L(N_1)\ L(N_1), L(N_1)L(N_1)L(N_1), ...\}$$
$$= L(N_1)^*$$
$$= r_1^*$$

$M = (\{s, f\} \cup Q_1, \Sigma_1, \delta, s, \{f\})$, where Q_1 is the set of states of N_1.

Rule 4 : For construction of NFA M for $r = r_1^+$, M is shown in Figure 2 (e).

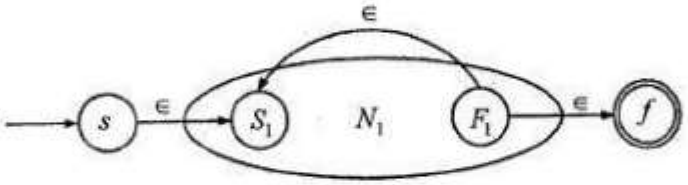


Figure 2(e) NFA for regular expression for r_1^+

$$L(M) = \{L(N_1), L(N_1)L(N_1), L(N_1)L(N_1)L(N_1), ...\}$$
$$= L(N_1)^+ = r_1^+$$

$M = (\{s, f\} \cup Q_1, \Sigma_1, \delta, s, \{f\})$, where Q_1 is the set of states of N_1.

Example 1 : Construct NFA for the regular expression a + ba *.

Solution : The regular expression

$r = a + ba^*$ can be broken into r_1 and r_2 as

$r_1 = a$

$r_2 = ba^*$

Let us draw the NFA for r_1, which is very simple.

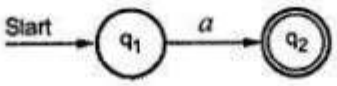


FIGURE 1: For r_1

Now, we will go for $r_2 = ba*$, this can be broken into r_3 and r_4 where $r_3 = b$ and $r_4 = a*$. Now the case for concatenation will be applied. The NFA will look like this r_2 will be shown in figure2.

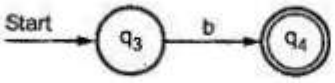


FIGURE 2: For r_3

and r_4 will be shown as

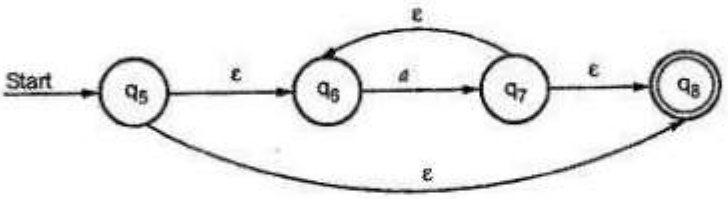


FIGURE 3 : For r_4

The r_2 will be $r_2 = r_3.r_4$

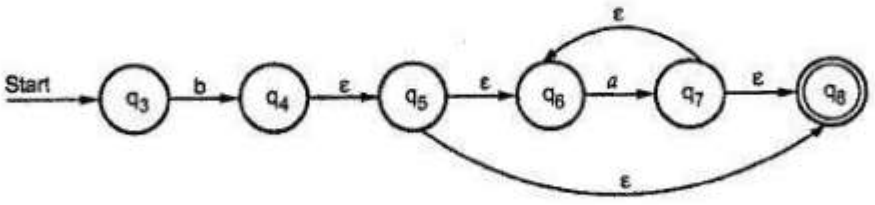


FIGURE 4 : For r_2

Now, we will draw NFA for $r = r_1 + r_2$ i. e. $a + ba^*$

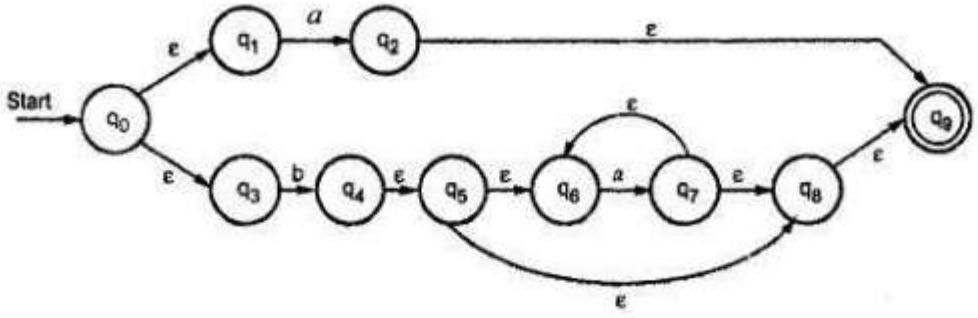


FIGURE 5 : NFA for $r = r_1 + r_2$ i. e. $a + ba^*$

Example 2 : Construct NFA with $\in$ moves for the regular expression $(0+1)^*$.

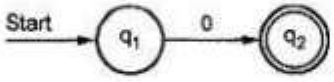
Solution : The NFA will be constructed step by step by breaking regular expression into small regular expressions.

$$r_3 = (r_1 + r_2)$$

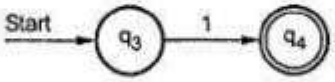
$$r = r_3^*$$

where $r_1 = 0,\ r_2 = 1$

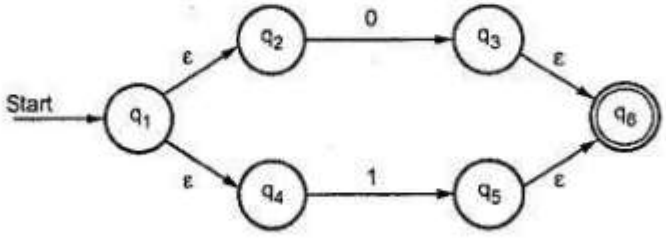
NFA for r_1 will be



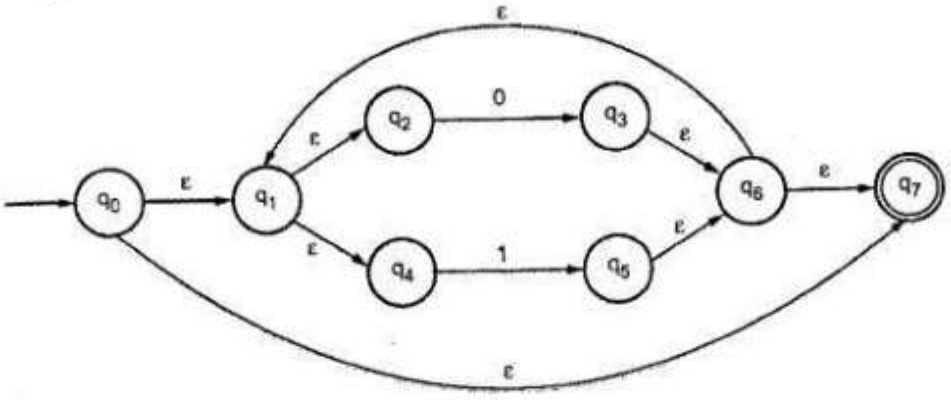
NFA for r_2 will be



NFA for r_3 will be



And finally



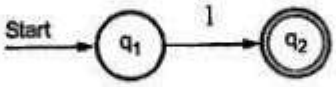
Example 3 : Construct NFA for the language having odd number of one's over the set $\Sigma = \{1\}$.

Solution : In this problem language L is given, we have to first convert it to regular expression. The r. e. for this L is written as r.e. = 1 (11)*.

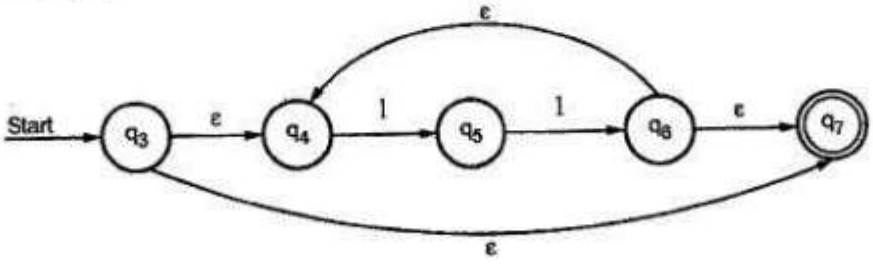
The r is now written as

$$r = r_1 \; r_2$$

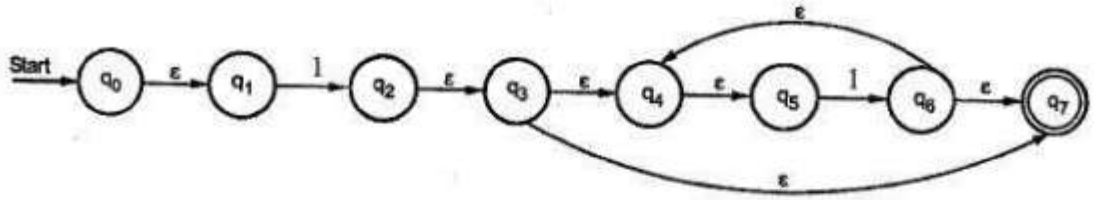
NFA for $r_1 = 1$ is



NFA for $r_2 = (11)^*$



The final NFA is



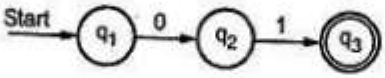
Example 4 : Construct NFA for the r. e. $(01+2^*)0$.

Solution : Let us design NFA for the regular expression by dividing the expression into smaller units

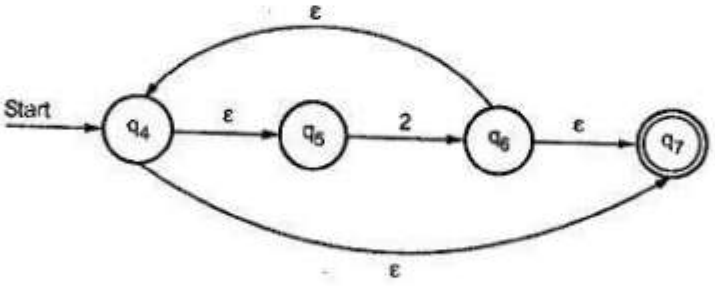
$$r = (r_1 + r_2)r_3$$

where $r_1 = 01$, $r_2 = 2^*$ and $r_3 = 0$

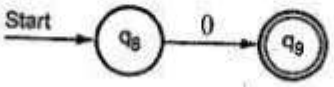
The NFA for r_1 will be



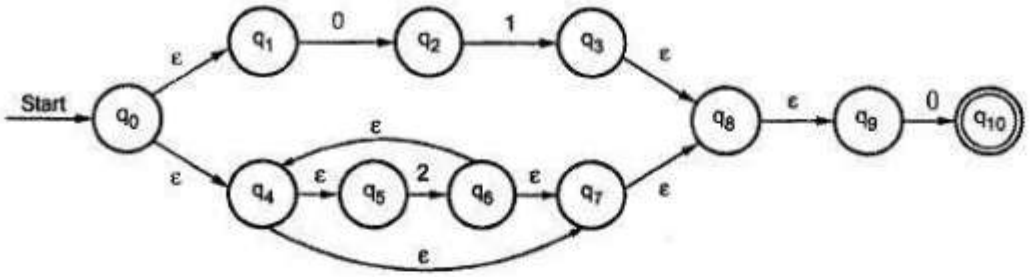
The NFA for r_2 will be



The NFA for r_3 will be



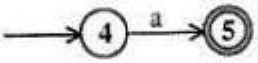
The final NFA will be



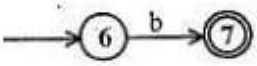
Example 5 : Obtain an NFA which accepts strings of a's and b's starting with the string ab.

Solution : The regular expression corresponding to this language is ab (a + b) *.

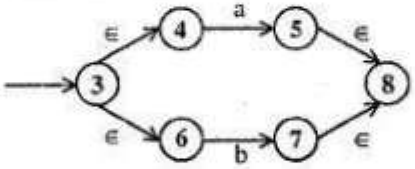
Step 1 : The machine to accept 'a' is shown below.



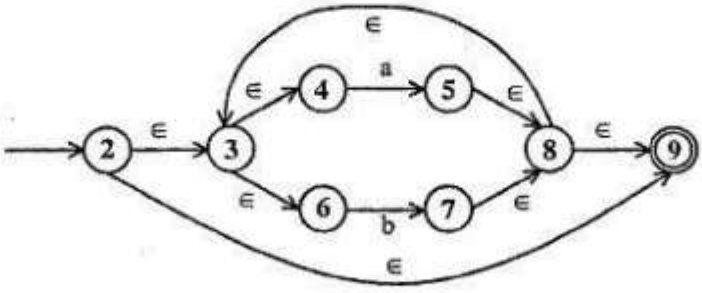
Step 2 : The machine to accept 'b' is shown below.



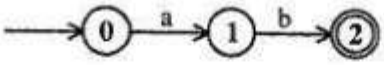
Step 3 : The machine to accept (a + b) is shown below.



Step 4 : The machine to accept (a + b)* is shown below .



Step 5 : The machine to accept ab is shown below.



Step 6 : The machine to accept ab (a + b)* is shown below .

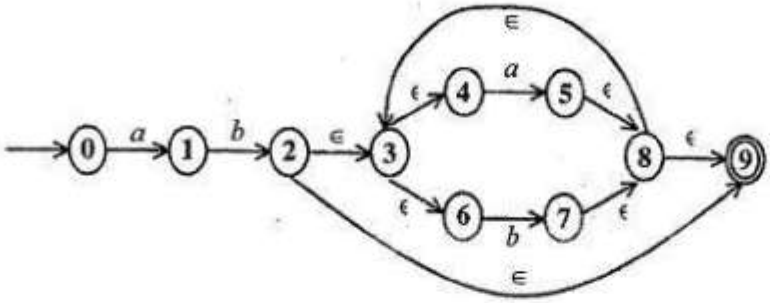
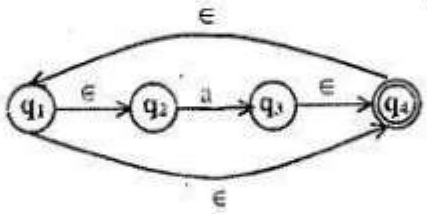


FIGURE : To accept the language (ab (a+b)*)

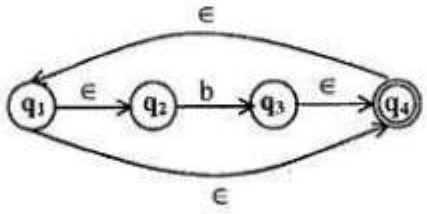
Example 6: Obtain an NFA for the regular expression $a^* + b^* + c^*$

Solution :

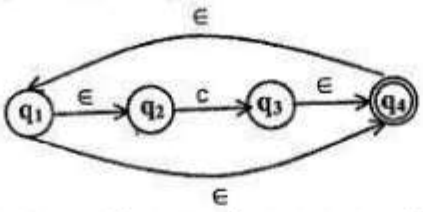
The machine corresponding the regular expression a* can be written as



The machine corresponding the regular expression b* can be written as



The machine corresponding the regular expression c* can be written as



The machine corresponding the regular expression $a^* + b^* + c^*$ is shown in below figure.

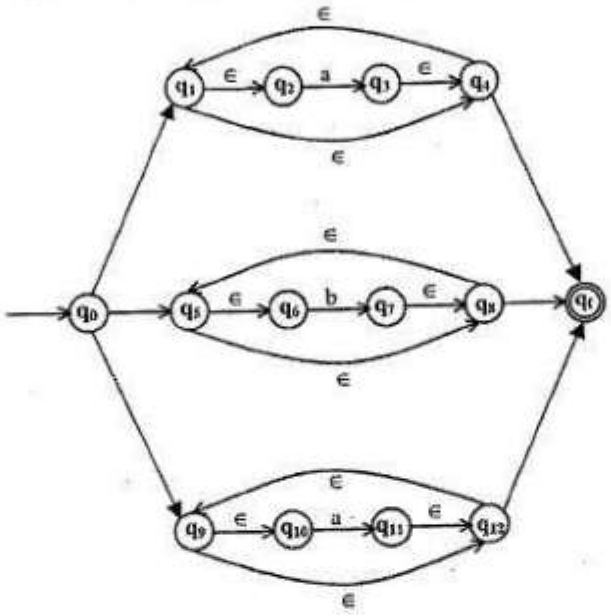
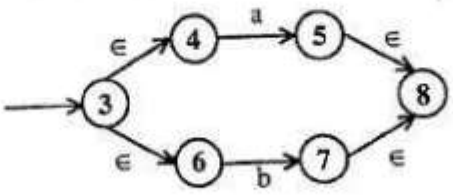


FIGURE: To accept the language $(a^* + b^* + c^*)$

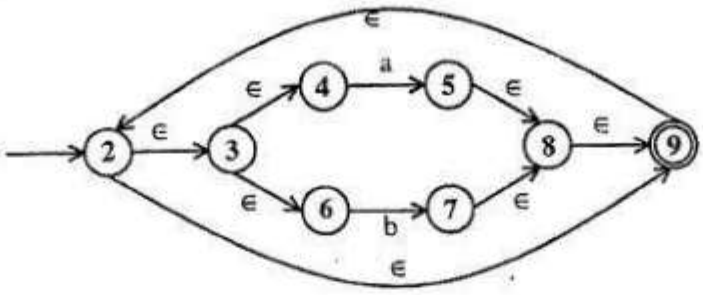
Example 7 : Obtain an NFA for the regular expression $(a+b)^* aa(a+b)^*$

Solution :

Step 1 : The machine to accept (a + b) is shown below.

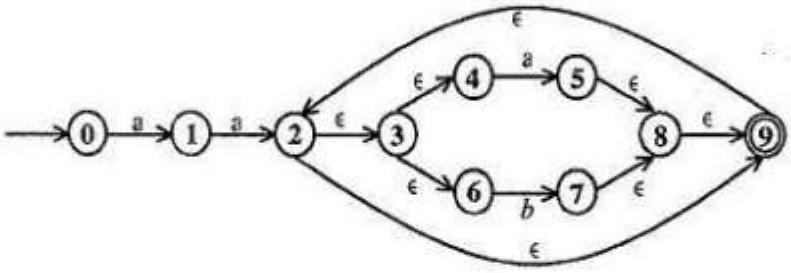


Step 2 : The machine to accept (a + b) * is shown below.



Step 3 : The machine to accept aa is shown below.

Step 4 : The machine to accept aa (a + b)* is shown below .



Step 5 : The machine to accept (a + b) * aa (a + b)* is shown in below figure.

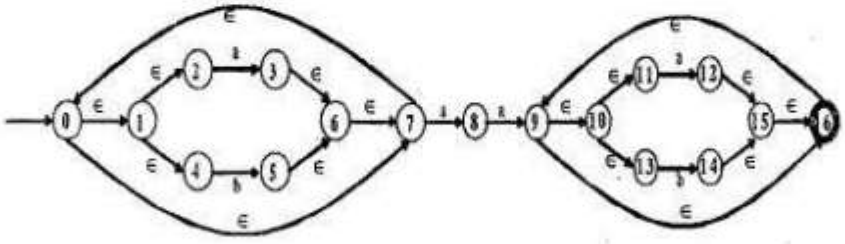


FIGURE : NFA to accept (a + b)* aa (a + b)*

Example 8 : Construction of DFA equivalent to a regular expression $(0+1)^*(00+11)(0+1)^*$ and also find the reduced DFA.

Solution : Given regular expression is $(0+1)^*(00+11)(0+1)^*$

Step 1 : (Construction of transition graph for NFA without $\in$ – moves).
First of all construct the transition graph with $\in$ using the construction rules

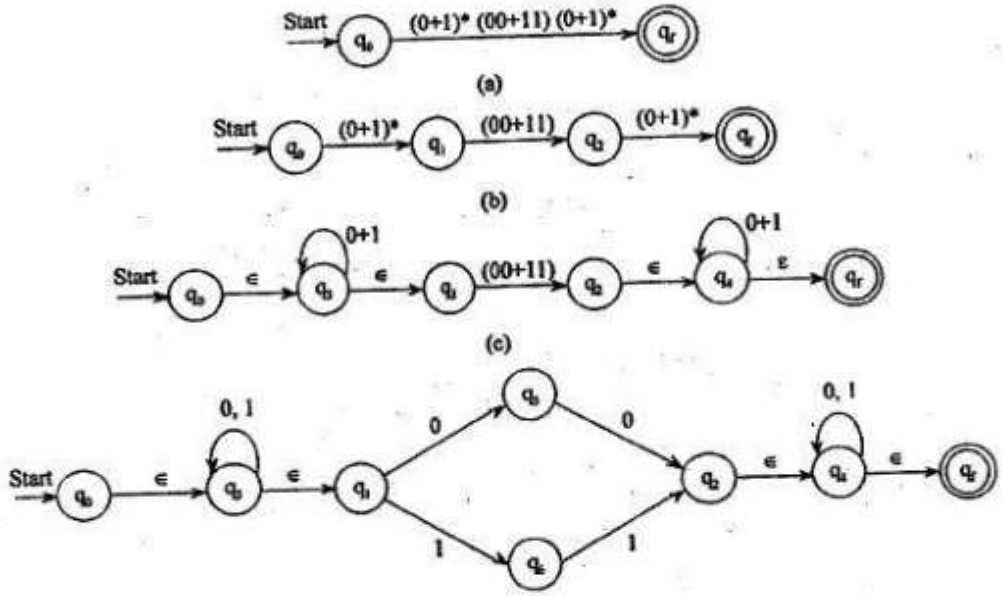


FIGURE: NFA for the given Regular Expression

Transition graph for NFA without $\in$ – moves is :

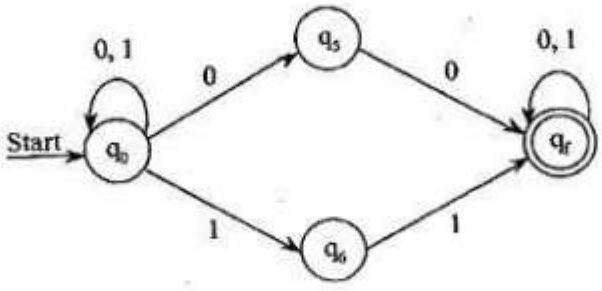


FIGURE : NFA without $\in$ - moves

Step 2 : We construct the transition table for NFA as given in below table :

	0	1
$\rightarrow q_0$	$\{q_0, q_5\}$	$\{q_0, q_6\}$
q_5	$\{q_f\}$	-
q_6	-	$\{q_f\}$
(q_f)	$\{q_f\}$	$\{q_f\}$

FIGURE: NFA Transition Table

Step 3 : Construct DFA table for NFA.

	Input	
States	0	1
$\rightarrow \{q_0\}$	$\{q_0, q_5\}$	$\{q_0, q_6\}$
$\{q_0, q_5\}$	$\{q_0, q_5, q_f\}$	$\{q_0, q_6\}$
$\{q_0, q_6\}$	$\{q_0, q_5\}$	$\{q_0, q_6, q_f\}$
$(\{q_0, q_5, q_f\})$	$\{q_0, q_5, q_f\}$	$\{q_0, q_6, q_f\}$
$(\{q_0, q_6, q_f\})$	$\{q_0, q_5, q_f\}$	$\{q_0, q_6, q_f\}$

FIGURE: DFA Transition Table

The state diagram for the successor table is the required DFA as shown in below figure.

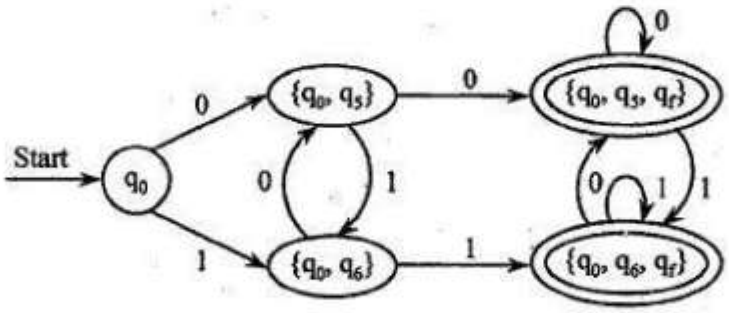


FIGURE: Required DFA for Regular expression $(0+1)^*(00+11)(0+1)^*$

As q_f is the only final state of NFA, $\{q_0, q_5, q_f\}$ and $\{q_0, q_6, q_f\}$ are the final states of DFA.

Reduce the Number of States of above DFA

As the rows corresponding to $\{q_0, q_5, q_f\}$ and $\{q_0, q_6, q_f\}$ are identical and delete the last row $\{q_0, q_6, q_f\}$.

States	Input 0	1
→ $\{q_0\}$	$\{q_0, q_5\}$	$\{q_0, q_6\}$
$\{q_0, q_5\}$	$\{q_0, q_5, q_f\}$	$\{q_0, q_6\}$
$\{q_0, q_6\}$	$\{q_0, q_5\}$	$\{q_0, q_6, q_f\}$
$\{q_0, q_5, q_f\}$	$\{q_0, q_5, q_f\}$	$\{q_0, q_6, q_f\}$

FIGURE : Reduced Transition Table of DFA

The reduced DFA transition diagram is,

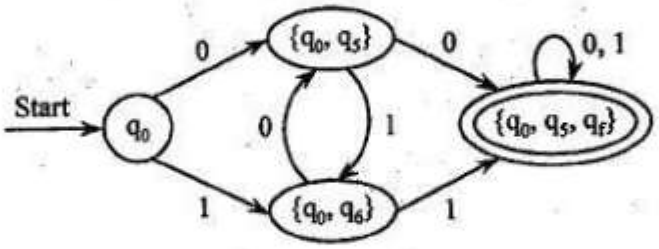


FIGURE : Reduced DFA for Regular Expression $(0+1)^*(00+11)(0+1)^*$

3.6 CONVERSION OF FA TO RE

Theorem : If L is accepted by a DFA, then L is denoted by a regular expression.

Proof : Let L be the set accepted by the DFA,

$$M = (\{q_1, q_2, \ldots\ldots q_n\}, \Sigma, \delta, q_1, F)$$

Let R_{ij}^{k} denote the set of all strings x such that $\delta(q_i, x) = q_j$ and if $\delta(q_i, y) = q_l$ for any y that is a prefix (initial segment) of x, other than x or $\in$, then $1 \le k$, i.e., R_{ij}^{k} is the set of all strings that take the finite automaton from state q_i to state q_j without going through any state numbered higher than k.

R_{ij}^{k} can be defined recursively as,

$$R_{ij}^{k} = R_{ij}^{k-1}\left(R_{kk}^{k-1}\right)^{*} R_{kj}^{k-1} \cup R_{ij}^{k-1} \qquad \ldots\ldots (1)$$

$$R_{ij}^{0} = \begin{cases} \{a / \delta(q_i, a) = q_j & \text{if } i \neq j \\ \{a / \delta(q_i, a) = q_j\} \cup \{\in\} & \text{if } i = j \end{cases}$$

To show that for each i, j and k, there exists a regular expression R_{ij}^{k} denoting the language R_{ij}^{k} i.e., by applying induction on k.

Basis Step :

If $(k = 0)$, R_{ij}^{0} is a finite set of strings each of which is either $\in$ or a single symbol.

r_{ij}^{0} can be expressed as,

$$r_{ij}^{0} = a_1 + a_2 + \ldots\ldots + a_p \; (or \; r_{ij} = a_1 + a_2 + \ldots\ldots + a_p + \in \; if \; i = j)$$

Where, $\{a_1, a_2, \ldots, a_p\}$ is the set of all symbols 'a' such that $\delta(q_i, a) = q_j$.

If there are no such a's, then ϕ (or $\in$ in the case i = j) serves as r_{ij}^{0}.

Induction :

The recursive formula for R_{ij}^{k} given in (1) clearly involves only the regular expression operators.

By induction hypothesis, for each 1 and m, a regular expression r_{lm}^{k-1} such that,

$$L\left(r_{lm}^{k-1}\right) = R_{lm}^{k-1}$$

$$r_{ij}^{k} = \left(r_{ik}^{k-1}\right)\left(r_{kk}^{k-1}\right)^{*}\left(r_{kj}^{k-1}\right) + r_{ij}^{k-1}$$

Which completes the induction.

To complete proof observe that $L(M) = \bigcup_{q_j \in F} R_{1j}^{n}$

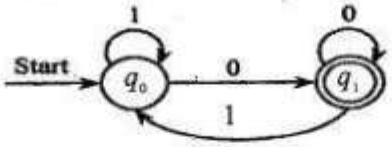
Since R_{1j}^{n} denotes the labels of all paths from q_1 to q_j.

$\therefore$ $L(M)$ is denoted by regular expression,

$$L(M) = r_{1j_1}^{n} + r_{1j_2}^{n} + r_{1j_p}^{n}$$

Where, $F = \{q_{j_1}, q_{j_2}, \ldots . q_{j_p}\}$

Example 1: Write equivalent regular expression for the following deterministic finite automaton.



Solution : A table is constructed as shown in below Table (K starts from 0 to number of states in the design) and the entries are calculated according to theorem.

	$k=0$	$k=1$
r_{11}^k	$1+\in$	$(1+\in)1^*$
r_{12}^k	0	01^*
r_{21}^k	1	11^*
r_{22}^k	$0+\in$	$11^*0+0+\in$

r_{ij}^0 values are calculated as, $r_{ij}^0 = \begin{cases} \{a/\delta(q_i,a)=q_j\} & \text{if } i \neq j \\ \{a/\delta(q_i,a)=q_j\} & \text{if } i=j \end{cases}$

r_{11}^0: $\delta(q_0,0)=q_1$ not satisfying above condition

$\delta(q_0,1)=q_0$ satisfying above condition and $\in$ is default added because i = j condition.

$r_{11}^0 = 1+\in$

r_{12}^0: $\delta(q_0,0)=q_1$ satisfying condition $(\because i \neq j)$

$\delta(q_0,1)=q_0$ not satisfying condition

$r_{12}^0 = 0$

r_{21}^0: $\delta(q_1,0)=q_1$ not satisfying

$\delta(q_1,1)=q_0$ satisfying condition

$\therefore r_{21}^0 = 1 (i \neq j)$

r_{22}^0: $\delta(q_1,0)=q_1$ satisfying condition

$\delta(q_1,1)=q_0$ not satisfying condition

$\therefore r_{22}^0 = 0+\in (i=j)$

r_{11}^1 : Where k = 1 we have to apply,

$r_{ij}^k = r_{ik}^{k-1}\left(r_{kk}^{k-1}\right)^*\left(r_{kj}^{k-1}\right) \cup r_{ij}^{k-1}$

$r_{11}^1 = r_{11}^0\left(r_{11}^0\right)^*\left(r_{11}^0\right) \cup r_{11}^0$

Considering values from table (k = 0),

$r_{11}^{1} = (1+\epsilon)(1+\epsilon)^{*}(1+\epsilon) \cup (1+\epsilon)$

Applying $(\epsilon + rr^{*}) = r^{*}$

$= 1+\epsilon((1+\epsilon)^{*}(1+\epsilon)+\epsilon)$

$= (1+\epsilon)(1+\epsilon)^{*}$ $(\because (1+\epsilon^{*}) = 1^{*})$

$= (1+\epsilon)\,1^{*}$

$r_{12}^{1} = (r_{11}^{0})(r_{11}^{0})^{*}(r_{12}^{0}) \cup (r_{12}^{0})$

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

$= 0((1+\epsilon)(1+\epsilon)^{*}+\epsilon)$ $(\because \epsilon + r\,r^{*} = r^{*})$

$= 0\,(1+\epsilon)^{*}$

$= 0\,1^{*}$

$r_{21}^{1} = (r_{21}^{0})(r_{11}^{0})^{*}(r_{11}^{0}) \cup (r_{21}^{0})$

$= 1\,(1+\epsilon)^{*}(1+\epsilon)+1$

$= 1((1+\epsilon)^{*}(1+\epsilon)+\epsilon)$

$= 1\,(1+\epsilon)^{*}$

$= 1\,1^{*}$

$r_{22}^{1} = (r_{21}^{0})(r_{11}^{0})^{*}(r_{12}^{0}) \cup (r_{22}^{0})$

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

$= 1\,1^{*}0+0+\epsilon$

Now the complete construction of regular expression is, in the given FA the starting state is q_0 and final state q_1. Write expressing from starting to all final states by taking k as total number of states.

r_{12}^{2} is final term to construct regular expression.

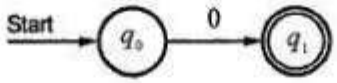
$r_{12}^{2} = (r_{12}^{1})(r_{22}^{1})^{*}(r_{22}^{1}) \cup (r_{12}^{1})$

$= 0\;1^{*}(1\,1^{*}0+0+\epsilon)^{*}(1\,1^{*}0+0+\epsilon)+0\,1^{*}$

$= 0\;1^{*}((1\,1^{*}0+0+\epsilon)^{*}(1\,1^{*}0+0+\epsilon)+\epsilon)$

$= 0\;1^{*}(1\,1^{*}0+0+\epsilon)^{*}$ $(\because \epsilon + r\,r^{*} = r^{*})$

Example 2: Construct the regular expression for the finite automata given in below figure.



Solution :

	k = 0
r_{11}	$\in$
r_{12}	0
r_{21}	ϕ
r_{22}	$\in$

In above table, we have calculated the values as r_{ij} will indicate the set of all the input string from q_i to q_j. If i = j then we add $\in$ with the input string. If $i \neq j$ and there is no path from q_i to q_j then we add ϕ.

Let us compute r_{11}^0

r_{11}^0 where $i = 1, j = 1, k = 0$. There is no path from q_i to q_j but i = j. So we add $\in$ in the k = 0 column at r_{11}^0 row.

Similarly

r_{12}^0 = The input from q_0 to q_1

$r_{12}^0 = 0$

r_{21}^0 = No input from q_1 to q_0 and $i \neq j$

So we add ϕ over there.

r_{22}^0 = No input from q_1 to q_1, since i = j.

We will add $\in$.

Let us build the table when k = 1

	k = 1	
	Computation	**Regular Expression**
r_{11}^1	$r_{ij}^k = r_{ik}^{k-1}\left(r_{kk}^{k-1}\right)^* r_{kj}^{k-1} + r_{ij}^{k-1}$ $i=1, j=1, k=1$ $r_{11}^1 = r_{11}^0\left(r_{11}^0\right)^*\left(r_{11}^0\right) + r_{11}^0$ $= \in (\in)^* (\in) + \in$ $r_{11}^1 = \in$	$\in$
r_{12}^1	$i=1, j=2, k=1$ $r_{12}^1 = r_{11}^0\left(r_{11}^0\right)^*\left(r_{12}^0\right) + r_{12}^0$ $r_{12}^1 = \in (\in)^* (0) + 0$ $= \in .0 + 0$ $= 0 + 0$ $= 0$	0
r_{21}^1	$i=2, j=1, k=1$ $r_{21}^1 = r_{21}^0\left(r_{11}^0\right)^*\left(r_{11}^0\right) + \left(r_{21}^0\right)$ $= \phi(\in)^* \in + \phi$ $= \phi + \phi \quad \therefore \phi \in = \phi$ $= \phi$	ϕ
r_{22}^1	$i=2, j=2, k=1$ $r_{22}^1 = r_{21}^0\left(r_{11}^0\right)^*\left(r_{12}^0\right) + r_{22}^0$ $= \phi.(\in)^* (0) + \in$ $= \phi + \in$ $= \in$	$\in$

Now let us compute for final state, which denotes the regular expression.

r_{12}^1 will be computed, because there are total 2 states and final state is q_1 whose start state is q_0.

$$r_{12}^2 = (r_{12}^1)(r_{22}^1)^*(r_{22}^1) + (r_{12}^1)$$
$$= 0(\in)^*(\in) + 0$$
$$= 0 + 0$$

$r_{12}^2 = 0$ which is a final regular expression.

3.6.1 Arden's Method for Converting DFA to RE

As we have seen the Arden's theorem is useful for checking the equivalence of two regular expressions, we will also see its use in conversion of DFA to RE.

Following algorithm is used to build the r. e. from given DFA.

1. Let q_0 be the initial state.
2. There are $q_1, q_2, q_3, q_4, \ldots q_n$ number of states. The final state may be some q_j where $j \leq n$.
3. Let α_{ij} represents the transition from q_i to q_j.
4. Calculate q_i such that

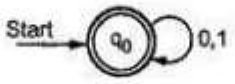
$$q_i = \alpha_{ij} \cdot q_j$$

If q_i is a start state

$$q_i = \alpha_{ji} \cdot q_j + \in$$

5. Similarly compute the final state which ultimately gives the regular expression r.

Example 1 : Construct RE for the given DFA.



Solution :

Since there is only one state in the finite automata let us solve for q_0 only.

$$q_0 = q_0 0 + q_0 1 + \in$$
$$q_0 = q_0 (0 + 1) + \in$$

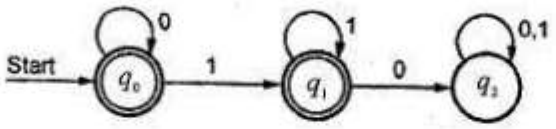
$$= \in .(0 + 1)^* \quad \because R = Q + RP$$

$$q_0 = (0+1)^*$$

Since q_0 is a final state, q_0 represents the final r. e. as

$$r = (0+1)^*.$$

Example 2 : Construct RE for the given DFA.



Solution : Let us build the regular expression for each state.

$$q_0 = q_0 0 + \in$$

$$q_1 = q_0 1 + q_1 1$$

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

Since final states are q_0 and q_1, we are interested in solving q_0 and q_1 only.

Let us see q_0 first

$$q_0 = \in + q_0 0$$

Which is R = Q + R P equivalent so we can write

$$q_0 = \in .(0)^*$$

$$q_0 = 0^* \quad \because \in .R = R$$

Substituting this value into q_1, we will get

$$q_1 = 0^* 1 + q_1 1$$

$$q_1 = 0^* 1(1)^* \quad \because R = Q + RP \Rightarrow QP^*$$

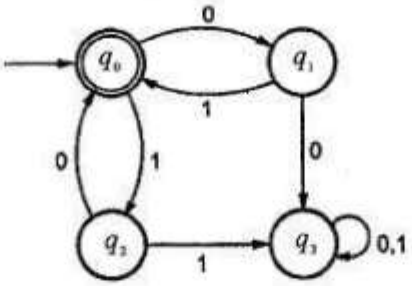
The regular expression is given by

$$r = q_0 + q_1$$

$$= 0^* + 0^* 1 . 1^*$$

$$r = 0^* + 0^* 1^+ \qquad \because 1 . 1^* = 1^+$$

Example 3 : Construct RE for the DFA given in below figure.



Solution : Let us see the equations

$$q_0 = q_1 1 + q_2 0 + \epsilon$$
$$q_1 = q_0 0$$
$$q_2 = q_0 1$$
$$q_3 = q_1 0 + q_2 1 + q_3(0+1)$$

Let us solve q_0 first,

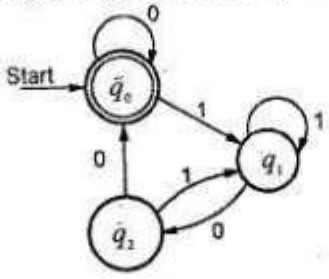
$$q_0 = q_1 1 + q_2 0 + \epsilon$$
$$q_0 = q_0 01 + q_0 10 + \epsilon$$
$$q_0 = q_0(01+10) + \epsilon \qquad \because R = Q + RP$$
$$q_0 = \epsilon(01+10)^* \qquad \Rightarrow QP^* \text{ where}$$
$$q_0 = (01+10)^* \qquad R = q_0, Q = \epsilon, P = (01+10)$$

Thus the regular expression will be

$$r = (01+10)^*$$

Since q_0 is a final state, we are interested in q_0 only.

Example 4 : Find out the regular expression from given DFA.



Solution : Let us solve the DFA by writing the regular expression, for each state .

$$q_0 = q_0 0 + q_2 0 + \in \quad \because \text{Initial state}$$

$$q_1 = q_1 1 + q_2 1 + q_0 1$$

$$q_2 = q_1 0$$

For getting the r. e. we have to solve q_0 the final state.

$$q_1 = q_1 1 + q_1 01 + q_0 1$$

$$q_1 = q_1 (1 + 01) + q_0 1$$

We will compare R = Q + R P with above equation, so $R = q_1, Q = q_0 1,\ P = (1 + 01)$ which ultimately gets reduced to QP*.

$$q_1 = q_0 1(1 + 01)^*$$

Substituting this value to q_0

$$q_0 = q_0 0 + q_2 0 + \in$$

$$= q_0 0 + q_1 00 + \in$$

$$= q_0 0 + q_0 (1(1 + 01)^*)00 + \in$$

$$q_0 = q_0 (0 + 1(1 + 01)^* 00) + \in$$

Again $R = Q + RP$

Where $R = q_0$

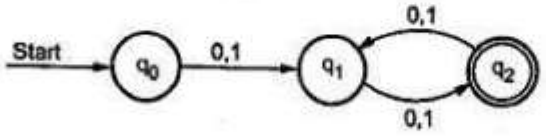
$Q = \in$

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

Hence $q_0 = \in .[0 + p(1 + 01)^* .00]^*$

$q_0 = [0 + 1(1 + 01)^* .00]^* \because \in .R = R$

Example 5 : Construct the regular expression for following DFA.



Solution : We can get the regular expression from state q_2. Let us see the equation of each state.

$$q_0 = \in$$
$$q_1 = q_0 1 + q_0 0 + q_2 1 + q_2 0$$
$$q_2 = q_1 1 + q_1 0$$

Putting value of q_0 in q_1

$$q_1 = \in .1 + \in .0 + q_2(0+1)$$
$$q_1 = (1+0) + q_2(0+1)$$

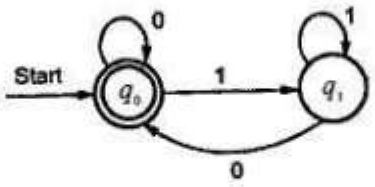
Now solve q_2

$$q_2 = (1+0)\, q_1$$
$$= ((1+0))\,[(1+0) + q_2(1+0)]$$
$$q_2 = (1+0).\,(1+0) + q_2(1+0)\,(1+0)$$

Here $R = q_2$, $Q = (1+0)\ (1+0)$, $P = (1+0)\ (1+0)$

$q_2 = (1+0)\ (1+0)\,[(1+0)\ (1+0)]^*$ is a regular expression.

Example 6 : Give the regular expression of following DFA.



For given DFA we can write the equation

$$q_0 = q_0 0 + q_1 0 + \in \quad \text{.... (1)}$$
$$q_1 = q_0 1 + q_1 1 \quad \text{... .(2)}$$

By theorem R = Q + RP we get R = QP*

$$R = q_1$$
$$Q = q_0 1$$
$$P = 1$$
$$\therefore \quad q_1 = q_0 11^*$$

As we know $R^+ = RR^*$ we can also write

$q_1 = q_0 1^+$

Let us put value of q_1 in equation (1)

$$q_0 = q_0 0 + q_0 1^+ 0 + \epsilon$$

$$q_0 = q_0 (0 + 1^+ 0) + \epsilon$$

Again we will apply R = Q + RP gives QP*

$R = q_0$

$Q = \epsilon$

$P = 0 + 1^+ 0$

$q_0 = \epsilon (0 + 1^+ 0)^*$

$q_0 = (0 + 1^+ 0)^*$ $\quad \because R\epsilon = \epsilon R = R$

In the given DFA, q_0 is a final state the equation computed for state q_0 will be regular expression. Hence r. e. for above DFA is

$r.e. = (0 + 1^+ 0)^*$

3.7 REGULAR AND NON - REGULAR LANGUAGES

The languages accepted by finite automata are described by regular expressions. So to prove a language is accepted by finite automata it is sufficient to prove the regular expression of that language is accepted by finite automata.

The languages which are accepted by some finite automata are called regular languages. Here it means that the FA accepts only the words of this language and does not accept any word outside it.

1. Some of the words of the language are not accepted by FA.

   (or)

2. All the words of the language are accepted in addition to that some extra strings are also accepted.

All languages are either regular or non regular, none of the languages are both.

By looking at some of the languages we can say whether they are regular or not.

i) The languages whose words need some sort of comparison can never be regular.

Example : $L = \{a^n b^n, n \geq 0\}$

Here the number of a's must be equal to number of b's for each 'a' we check the existence of b which cannot be done using FA.

ii) The languages whose words are in arithmetic progression and need no comparisons will be regular.

Example : 1. $L = \{a^{2n}, n \geq 1\}$

The words of this language are , aa, $aaaa$, $aaaaaa$, ..., a^{2n} which are in A.P with period 2. Hence it is a regular language.

2. $L = \{ a^p, p \text{ is prime} \}$

The words of this language are $\{ a, aa, aaa, aaaaaaa, ... a^p \}$. We can see these words are not in A.P. Hence it is not regular.

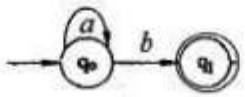
In this section, we will discuss how to prove that certain language is not regular (non - regular) language. Pumping Lemma is a useful tool to prove that a certain language is not regular language.

Since, the number of states in a FA is finite, say it is n (for some fixed value of n), and then it can recognize all the words of length less than n without any loop. Suppose, a regular language L has infinite number of words and the length of these words may or may not be equal to n. So, how can a FA recognize the L ? A FA can recognize L having some loop(s) and whenever the length of a given word is greater than or equal to n. So, we conclude that the loop in FA makes it able to accept those strings, which have length greater than or equal to its total number of states.

When a string z has bigger length (greater than number of states in FA) then we break this string into three parts, say u, v (v should not be null string), and w. Let FA has loop for v, and $z = uvw \in L$ is accepted by FA.

So, $z = uv^i w$ for i = 0,1, ... is also accepted by FA having some loop for v. This is the main concept, used in Pumping Lemma.

Now, consider a regular language L = a*b and corresponding FA shown in below figure.



We see the list of accepted strings given below :

b , ab, aab, aaab, ...

Let $u = \in, v = a$ (v should not be $\in$), and w = b, then a^i i.e. $z = uv^i w$ for some i = 0, 1, ... is accepted by FA. Now, we have good base to discuss the Pumping Lemma.

3.8 Pumping Lemma for Regular Sets

Pumping Lemma is useful because

1. It gives a method for pumping (generating) many substrings from a given string. In other words, we say, it provides means to break a given long input string into several substrings.

2. It gives necessary condition(s) to prove a set of strings is not regular.

Theorem :

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA having n states. M recognizes the language L. A long string $z \in L$ such that $|z| \geq n$ and $z = uvw$, where $v \neq \in$, then $uv^i w \in L$ for $i \geq 0$.

Proof :

M recognizes L and L is a regular set. If $z \in L$ such that $w = uvw$. Here v is optional in z and $|z| \geq n$, where n is the number of states in DFA.

Consider following DFA shown in below figure.

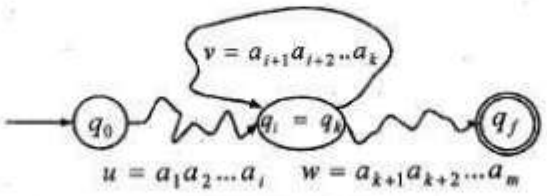


FIGURE : DFA for $uv^i w$

Let $z = a_1a_2a_3...a_ia_{i+1}...a_ka_{k+1}..a_m$

Where $u = a_1a_2a_3..a_i, v = a_{i+1}...a_k$ and $w = a_{k+1}...a_m$

The length of z is m and $m \geq n$. It means, when $m \geq n$, it indicates that there is some loop in transition diagram of M. Let v is the string obtained from the edges involved in looping as shown in above figure.

Case 1 : When $z = uv^0w = uw$ for $i = 0$, it means, uw is accepted and $uw \in L$.

Case 2 : $z = uv^iw$ for $i \geq 1$, it means that control of DFA M goes i - times into the loop with label v and uv^iw is accepted by M.

So, for all values of $i \geq 0$, $z = uv^iw$ is accepted by M.

Hence, the statement of the theorem is proved.

Application of Pumping Lemma

Pumping Lemma is used to prove certain sets are not regular sets. This is done as follows :

Step 1 : We assume that given set is regular and accepted by DFA M having n states.

Step 2 : Choose a string z such that $|z| \geq n$ and use Pumping Lemma to write $z = uv^iw$ for $i \geq 0, v \notin \in$, and $|uw| \leq n$.

Step 3 : Find a suitable integer i such that $uv^iw \notin L$ and this contradicts our assumption made in step 1 and hence L is not regular.

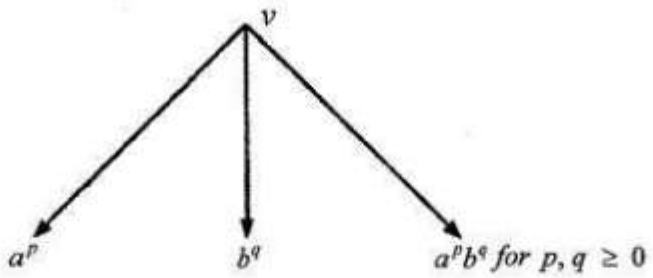
Example 1 : Prove that $L = \{a^nb^n : n \geq 1\}$ is not regular.

Solution : In given language the number of $a's$ is equal to the number of $b's$. This is the one clue to find the contradiction.

Step 1 : Let L is regular and accepted by DFA M with n states.

Step 2 : String $z \in L$ such that $|z| \geq n$ and $z = uv^iw \in L$ for $i \geq 0, v \notin \in$, and $|uw| \leq n$.

Step 3 : Selecting substring v



Let $z = uv^i w$ for $i = 0$

Case 1 : When $v = a^p$, then

$z = a^{n-p} b^n$

Number of $a's = n - p$, and number of $b's = n$

Number of $a's$ = Number of $b's$ if and only if $p = 0$ and number of $a's$ and $b's$ is not equal when $p > 0$.

So, for $p > 0$, $z = uv^i w \notin L$

Case 2 : When $v = a^q$, then

$z = a^n b^{n-q}$

Number of $a's = n$, and number of $b's = n - q$

Number of $a's$ = Number of $b's$ if and only if $q = 0$ and number of $a's$ and $b's$ is not equal when $q > 0$.

So, for $q > 0$, $z = uv^i w \notin L$.

Case 3 : When $v = a^p b^q$, then $z = a^{n-p} b^{n-q}$

Number of $a's = n - p$, and number of $b's = n - q$.

Number of $a's$ = Number of $b's$ if and only if $q = p$.

So, for $p \neq q$, $z = uv^i w \notin L$

Since, we get contradiction in all the cases, therefore L is not regular.

Example 2 : Show that $L = \{a^n b^n | n \geq 0\}$ is not regular.

Solution :

Step 1 : Let L is regular and n be the number of states in FA. Consider the string $z = a^n b^n$.

Step 2 : Note that $|z|=2n$ and is greater than n. So, we can split z into uvw such that $|uv| \leq n$ and $|v| \geq 1$ as shown below.

$$z = \overbrace{aaaaaa}^{u} \overbrace{a}^{v} \overbrace{bbbbbbb}^{w}$$

where $|u| = n-1$ and $|v| = 1$ so that $|uv| = |u| + |v| = n-1+1 = n$ and $|w| = n$. According to pumping lemma, $uv^i w \in L$ for i = 0, 1, 2, ...

Step 3 : If i is 0 i. e., v does not appear and so the number of a's will be less than the number of b's and so the string w does not contain some number of a's followed by same number of b's (equal to that of a's)

Similarly, if i = 2, 3,..., then number of a's will be more than the number of b's and so number of a's followed by equal number of b's does not exist. But, according to pumping lemma, n number of a's should be followed by n number of b's which is a contradiction to the assumption that the language is regular. So, the language L is not regular.

Example 3: Prove that $L = \{a^{i^2} : i \geq 1\}$ is not regular

Solution :

Method - I (Using Pumping Lemma for regular sets)

In L, all words have their lengths in perfect square and this is the clue for proving non - regular.

Step 1 : Let L be regular and accepted by DFA M with n states.

Step 2 : String $z \in L$ such that $|z| \geq n$ and $z = uv^i w \in L$ for $i \geq 0, y \notin \in$, let $|z| = n^2 \geq n$, and $|uw| \leq n$, (n is the number of states).

Step 3 : Since, length of v can not exceed n (the number of states), it means, $|v| \leq n$.

Let i = 2, so $z = uv^2 w \in L$, and

$|z| = |uvw| + |v| = n^2 + |v|$

So, $n^2 \leq |z| \leq n^2 + n$ (Since, $|v| \leq n$)

Or, $n^2 \leq |z| \leq n^2 + n + (n+1)$ (Adding n + 1 to make perfect square)

Or, $n^2 \le |z| \le (n+1)^2$

It means, the length of z is between n^2 and $(n+1)^2$, and is not a perfect square. Therefore, L is not regular.

Method - II

For example,

$z = a^{i^2}$

Let i = 2

z = aaaa

z = uvw

Assume uvw = aaaa

Take u = a

v = aa

w = a

By pumping lemma, even if we pump v i.e. increase v then language should show the length as perfect square .

uvw

= uv.vw

= aaaaaa

= length of a is not a perfect square

Thus the behaviour of the language is not regular, as after pumping something onto it does not show the same property (being square for this example.)

Example 4 : Show that $L = \{ww^R | w \in (0+1)^*\}$ is not regular.

Solution :

Step 1 : Let L is regular and n be the number of states in FA. Consider the string

$$z = \overbrace{\underbrace{1.....1}_{n}\underbrace{0.....0}_{n}}^{w}\overbrace{\underbrace{0.....0}_{n}\underbrace{1.....1}_{n}}^{w^R}$$

where n is the number of states of FA, w = 1 ... 10 .... 0 and reverse of w i. e., w^R = 0....01.....1.

Step 2 : Split the string z into uvw such that $|uv| \leq n$ and $|v| \geq 1$ as shown below.

$$z = \underbrace{1 \ldots 1}_{u}\underbrace{0 \ldots 0}_{v}\underbrace{0 \ldots 0}_{w}\underbrace{1 \ldots 1}$$

where $|u| = n-1$ and $|v| = 1$ so that $|uv| = |u| + |v| = n-1+1 = n$ which is true. According to pumping lemma, $uv^i w \in L$ for i = 0, 1, 2, ...

Step 3 : If i is 0 i. e., v does not appear and so the number of 1's on the left of z will be less than the number of 1's on the right of z and so the string is not in the form ww^R. So, $uv^i w \notin L$ when i = 0. This is a contradiction to the assumption that the language is regular. So, ww^R is not regular.

Example 5 : Show that $L = \{ 0^{2n} \mid n \geq 1 \}$ is regular.

Solution : This is a language length of string is always even.

i.e. n = 1 ; z = 00

n = 2 ; z = 00 00 and so on.

Let z = uvw

$z = 0^{2n}$

$|z| = 2^n = uv^i w$

If we add 2n to this string length.

$|z| = 4n = uv.vw$

= even length of string.

Thus even after pumping 2n to the string we get the even length. So the language L is regular language.

Example 6: Prove that L= { ww | w in (a + b)* } is not regular.

Solution : Prove the result by the method of contradiction.

Step 1 : Suppose L is regular, let 'n' be the number of states in the automaton M accepting 'L'.

Step 2 : Let us consider $ww = a^n b^n a^n b$ in L. $|ww| = 2(n+1) > n$ apply pumping lemma we write ww = xyz with $|y| \neq 0, |xy| \leq n$.

Step 3 : To find i such that $xy^i z \notin L$ for getting a contradiction. The string 'y' can be in only one of the following forms.

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

Case 2 : y has only one b.

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

But $|y| \leq |xy| \leq n$.

In case 1, we can take i = 0.

Then $xy^0z = xz$ is of the form $a^m ba^n b$. Where $m = n - k < n (or\ a^n b\ a^n b)$ $x\ z$ can not be written in the form uu with $u \in \{a,b\}^*$ and so $xz \notin L$.

In case 2 also. We can take i = 0.

Then $xy^0z = xz$ has only one b.

So $xz \notin L$ as any element in L should have even number of a's and even number of b's.

Thus in both cases we get contradiction.

$\therefore$ L is not regular.

Example 7 : Show that $L = \{a^p \mid p\ is\ a\ prime\ number\}$ is not regular.

Method - I :

Step 1 : Let L is regular and get a contradiction. Let n be the number of states in the FA accepting L.

Step 2 : Let p be a prime number greater than n. Let $z = a^p$. By pumping lemma, z can be written as $z = uvw$, with $|uv| \leq n$ and $|v| > 0$. u, v, w are simply strings of a's. So, $v = a^m$ for some $m \geq 1$ (and $\leq$ n).

Step 3 : Let i = p + 1. Then $|uv^i w| = |uvw| + |v^{i-1}| = p + (i-1)m = p + pm$. By pumping lemma, $uv^i w \in L$. But $|uv^i w| = p + pm = p(1+m)$ and $p(1+m)$ is not a prime. So $uv^i w \notin L$. This is a contradiction. Thus L is not regular.

Method - II : Let us assume L is a regular and P is a prime number.

$z = a^p$

$|z| = uvw$ i = 1

Now consider $z = uv^i w$ where i = 2

$= uv.vw$

Adding 1 to P we get,

$P < |uvvw|$

$P < P + 1$

But P + 1 is not a prime number. Hence what we have assumed becomes contradictory. Thus L behaves as it is not a regular language.

Example 8 : Show that the language $L = \{a^i\ b^{2i} | i > 0\}$ is not regular.

Solution : The set of strings accepted by language L is,

$L = \{abb, aabbbb, aaabbbbbb, aaaabbbbbbbb...\}$

Applying Pumping lemma for any of the strings above.

Take the string abb.

It is of the form uvw.

Where, $|uv| \leq i, |v| \geq 1$

To find i such that $uv^i w \notin L$

Take i = 2 here, then

$uv^2w = a(bb)b$

$= abbb$

Hence $uv^2w = abbb \notin L$

Since abbb is not present in the strings of L.

$\therefore$ L is not regular.

Example 9 : Show that $L = \{0^n | n$ is a perfect square $\}$ is not regular.

Solution :

Step 1 : Let L is regular by Pumping lemma. Let n be number of states of FA accepting L.

Step 2 : Let $z = 0^n$ then $|z| = n \geq 2$.

Therefore, we can write z = uvw ; Where $|uv| \leq n, |v| \geq 1$.

Take any string of the language L = { 00, 0000, 000000 .... }

Take 0000 as string, here u = 0, v = 0, w = 00 to find i such that $uv^i w \notin L$.

Take i = 2 here, then

$uv^i w = 0(0)^2 00$

$= 00000$

This string 00000 is not present in strings of language L. So $uv^i w \notin L$.

$\therefore$ It is a contradiction.

3.9 PROPERTIES OF REGULAR SETS

Regular sets are closed under following properties.

1. Union
2. Concatenation

3. Kleene Closure
4. Complementation
5. Transpose
6. Intersection

1. Union : If R_1 and R_2 are two regular sets, then union of these denoted by $R_1 + R_2$ or $R_1 \cup R_2$ is also a regular set.

Proof : Let R_1 and R_2 be recognized by NFA N_1 and N_2 respectively as shown in Figure1(a) and Figure1(b).

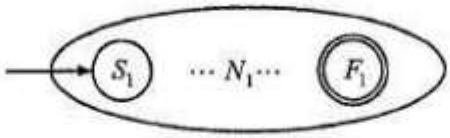


FIGURE 1(a) NFA for regular set R_1

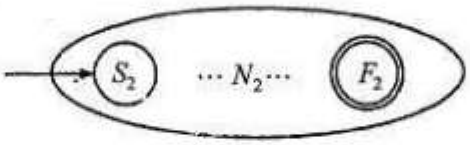


FIGURE 1(b) NFA for regular set R_2

We construct a new NFA N based on union of N_1 and N_2 as shown in Figure 1 (c)

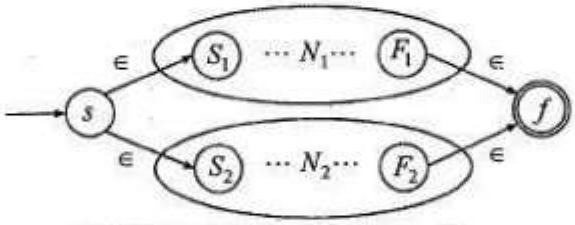


FIGURE 1(c) NFA for $N_1 + N_2$

Now,

$$L(N) = \in L(N_1) \in + \in L(N_2) \in$$
$$= \in R_1 \in + \in R_2 \in$$
$$= R_1 + R_2$$

Since, N is FA, hence $L(N)$ is a regular set (language). Therefore, $R_1 + R_2$ is a regular set.

2. **Concatenation :** If R_1 and R_2 are two regular sets, then concatenation of these denoted by R_1R_2 is also a regular set.

Proof : Let R_1 and R_2 be recognized by NFA N_1 and N_2 respectively as shown in Figure 2(a) and Figure 2(b).

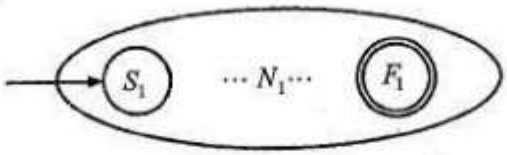


FIGURE 2(a) NFA for regular set R_1

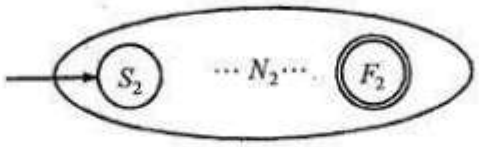


FIGURE 2(b) NFA for regular set R_2

We construct a new NFA N based on concatenation of N_1 and N_2 as shown in Figure2(c).

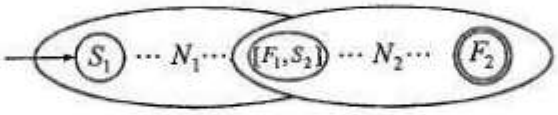


FIGURE 2(c) NFA for regular set R_1R_2

Now,

$L(N)$ = Regular set accepted by N_1 followed by regular set accepted by $N_2 = R_1R_2$

Since, $L(N)$ is a regular set, hence R_1R_2 is also a regular set.

3. **Kleene Closure :** If R is a regular set, then Kleene closure of this denoted by R^* is also a regular set.

Proof : Let R is accepted by NFA N shown in Figure 3(a).

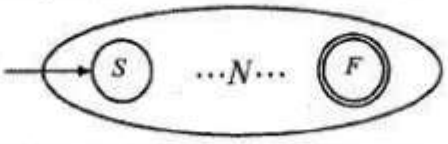


FIGURE 3(a) NFA for regular set R

We construct a new NFA based on NFA N as shown in Figure 3(b).

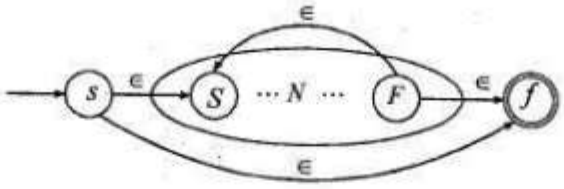


FIGURE 3(b) NFA for regular expression for R^*

Now,

$$L(N) = \{\in, R, R\,R, R\,R\,R, ...\}$$

$$= L^*$$

Since, $L(N)$ is a regular set, therefore R^* is a regular set.

4. **Complement :** If R is a regular set on some alphabet Σ, then complement of R is denoted by $\Sigma^* - R$ or $\overline{R}$ is also a regular set.

Proof : Let R be accepted by NFA $N = (Q, \Sigma, \delta, s, F)$. It means, $L(N) = R$. N is shown in Figure 4(a).

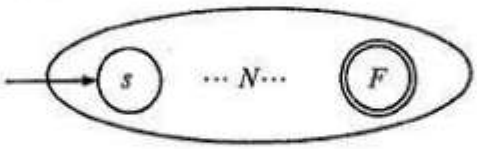


FIGURE 4(a) NFA for regular set R

We construct a new NFA N' based on N as follows :

(a) Change all final states to non-final states.
(b) Change all non-final states to final states.
N' is shown in Figure 4(b)

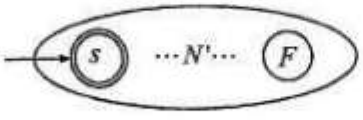


FIGURE 4 (b) NFA

Now,

$L(N')$ = {All the words which are not accepted by NFA N}

= { All the rejected words by NFA N}

$= \Sigma^* - R$

Since, $L(N')$ is a regular set, therefore $(\Sigma^* - R)$ is a regular set.

5. Transpose : If R is a regular set, then the transpose denoted by R^T, is also a regular set.

Proof : Let R be accepted by NFA $N = (Q, \Sigma, \delta, s, F)$ as shown in Figure 5(a).

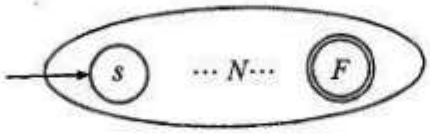


FIGURE 5 (a) NFA N for regular set R

If w is a word in R, then transpose (reverse) is denoted by w^T.

Let $w = a_1 a_2 ... a_n$

Then $w^T = a_n a_{n-1} .... a_1$

We construct a new N' based on N using following rules :

(a) Change the all final states into non-final states and merge all these into one state and make it initial state.

(b) Change initial state to final state.

(c) Reverse the direction of all edges.

N' is shown in Figure5 (b)

FIGURE 5(b) NFA N' for regular set R^T

Let $w = a_1 a_2 \ldots a_n$ be a word in R, then it is recognized by N and $w^T = a_n a_{n-1} \ldots a_1$ is recognized by N' as shown in Figure5 (b)

In general, we say that if a word w in R is accepted by N, and then N' accepts w^T.

Since, $L(N')$ is a regular set containing all w^T; it means, $L(N') = R^T$.

Thus, R^T is a regular set.

6. **Intersection :** if R_1 and R_2 are two regular sets over Σ, then intersection of these denoted by $R_1 \cap R_2$ is also a regular set.

Proof : By De Morgan's law for two sets A and B over R,

$A \cap B = R^* - ((R^* - A) \cup (R^* - B))$

So, $R_1 \cap R_2 = \Sigma^* - ((\Sigma^* - R_1) \cup (\Sigma^* - R_2))$

Let $R_3 = (\Sigma^* - R_1)$ and $R_4 = (\Sigma^* - R_2)$

So, R_3 and R_4 are regular sets as these are complement of R_1 and R_2.

Let $R_5 = R_3 \cup R_4$

So, R_5 is a regular set because it is the union of two regular sets R_3 and R_4.

Let $R_6 = \Sigma^* - R_5$

So, R_6 is a regular set because it is the complement of regular set R_5.

Therefore, intersection of two regular sets is also regular set.

REVIEW QUESTIONS

Q1. What is regular set ? Explain with an example.

Answer :

For Answer refer to Topic : 3.1, Page No : 3.1.

Q2. What is regular expression ? Explain with an example.

Answer :

For Answer refer to Topic : 3.2 , Page No : 3.2.

Q3. Obtain a regular expression to accept a language consisting of strings of a's and b's of even length.

Answer :

For Answer refer to example - 1 , Page No : 3.4.

Q4. Obtain a regular expression to accept a language consisting of strings of a's and b's of odd length.

Answer :

For Answer refer to example - 2 , Page No : 3.4.

Q5. Obtain a regular expression such that $L(r) = \{W \mid W \in \{0,1\}^*$ with at least three consecutive 0's $\}$.

Answer :

For Answer refer to example - 3 , Page No : 3.4.

Q6. Obtain a regular expression to accept strings of a's and b's ending with 'b' and has no substring aa.

Answer :

For Answer refer to example - 4 , Page No : 3.5.

Q7. Obtain a regular expression to accept strings of 0's and 1's having no two consecutive zeros.

Answer :

For Answer refer to example - 5 , Page No : 3.5.

Q8. Obtain a regular expression to accept strings of a's and b's of length ≤ 10.

Answer :

For Answer refer to example - 6 , Page No : 3.5.

Q9. Obtain a regular expression to accept strings of a's and b's starting with 'a' and ending with 'b'.

Answer :

For Answer refer to example - 7 , Page No : 3.6.

Q10. Explain equivalence of two REs using Arden's theorem.

Answer :

For Answer refer to Topic : 3.3.1, Page No : 3.7.

Q11. Prove $(1+00*1)+(1+00*1)(0+10*1)*(0+10*1)=0*1(0+10*1)*$

Answer :

For Answer refer to example - 1 , Page No : 3.9.

Q12. Show that $(0*1*)*=(0+1)*$

Answer :

For Answer refer to example - 2 , Page No : 3.9.

Q13. If r be a regular expression then there exists a NFA with $\in$- moves, which accepts L(R).

Answer :

For Answer refer to Topic : 3.5 , Page No : 3.10.

Q14. Construct NFA for the regular expression a + ba *.

Answer :

For Answer refer to example - 1 , Page No : 3.13.

Q15. Construct NFA with $\in$ moves for the regular expression $(0+1)*$.

Answer :

For Answer refer to example - 2 , Page No : 3.15.

Q16. Construct NFA for the language having odd number of one's over the set $\Sigma = \{1\}$.

Answer :

For Answer refer to example - 3 , Page No : 3.16.

Q17. Construct NFA for the r. e. $(01+2^*)0$.

Answer :

For Answer refer to example - 4 , Page No : 3.17.

Q18. Obtain an NFA which accepts strings of a's and b's starting with the string ab.

Answer :

For Answer refer to example - 5 , Page No : 3.18.

Q19. Obtain an NFA for the regular expression $a^* + b^* + c^*$

Answer :

For Answer refer to example - 6 , Page No : 3.19.

Q20. Obtain an NFA for the regular expression $(a+b)^* aa(a+b)^*$

Answer :

For Answer refer to example - 7 , Page No : 3.20.

Q21. Construction of DFA equivalent to a regular expression $(0+1)^*(00+11)(0+1)^*$ and also find the reduced DFA.

Answer :

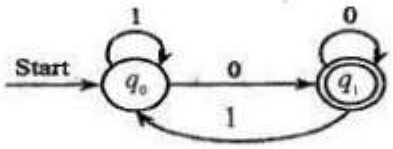
For Answer refer to example - 8 , Page No : 3.22.

Q22. If L is accepted by a DFA, then L is denoted by a regular expression.

Answer :

For Answer refer to Theorem , Page No : 3.24.

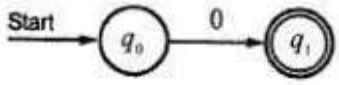
Q23. Write equivalent regular expression for the following deterministic finite automaton.



Answer :

For Answer refer to example - 1 , Page No : 3.26.

Q24. Construct the regular expression for the finite automata given in below figure.



Answer :

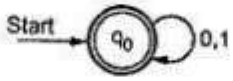
For Answer refer to example - 2 , Page No : 3.28.

Q25. Explain Arden's method for converting DFA to RE.

Answer :

For Answer refer to Topic : 3.6.1 , Page No : 3.30.

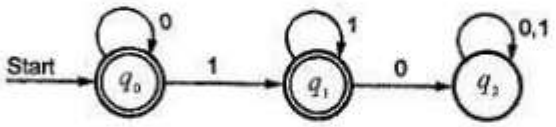
Q26. Construct RE for the given DFA.



Answer :

For Answer refer to example - 1 , Page No : 3.30.

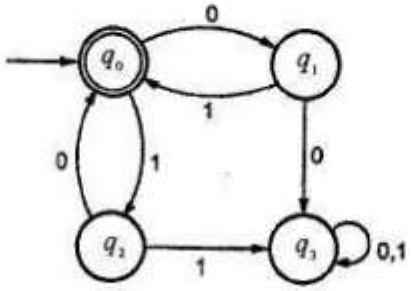
Q27. Construct RE for the given DFA.



Answer :

For Answer refer to example - 2 , Page No : 3.31.

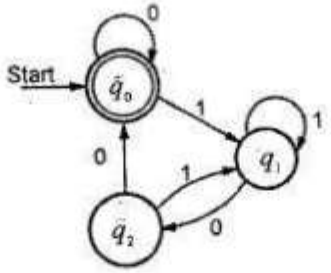
Q28. Construct RE for the DFA given in below figure.



Answer :

For Answer refer to example - 3 , Page No : 3.32.

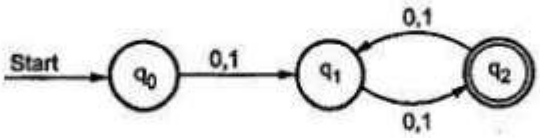
Q29. Find out the regular expression from given DFA.



Answer :

For Answer refer to example - 4 , Page No : 3.32.

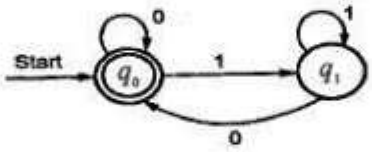
Q30. Construct the regular expression for following DFA.



Answer :

For Answer refer to example - 5 , Page No : 3.33.

Q31. Give the regular expression of following DFA.



Answer :

For Answer refer to example - 6 , Page No : 3.34.

Q32. State and prove Pumping Lemma for regular sets.

Answer :

For Answer refer to Theorem , Page No : 3.37.

Q33. Prove that $L = \{a^n b^n : n \geq 1\}$ is not regular.

Answer :

For Answer refer to example - 1 , Page No : 3.38.

Q34. Show that $L=\{a^n b^n | n \geq 0\}$ is not regular.

Answer :

For Answer refer to example - 2 , Page No : 3.40.

Q35. Prove that $L=\{a^{i^3} : i \geq 1\}$ is not regular .

Answer :

For Answer refer to example - 3 , Page No : 3.40.

Q36. Show that $L=\{ww^R | w \in (0+1)^*\}$ is not regular.

Answer :

For Answer refer to example - 4 , Page No : 3.41.

Q37. Show that $L=\left\{ 0^{2^n} | n \geq 1 \right\}$ is regular.

Answer :

For Answer refer to example - 5 , Page No : 3.42.

Q38. Prove that L= { ww | w in (a + b)* } is not regular.

Answer :

For Answer refer to example - 6 , Page No : 3.42.

Q39. Show that $L=\{a^p \mid p \text{ is a prime number}\}$ is not regular.

Answer :

For Answer refer to example - 7 , Page No : 3.43.

Q40. Show that the language $L=\{a^i b^{2^i} \mid i > 0\}$ is not regular.

Answer :

For Answer refer to example - 8 , Page No : 3.44.

Q41. Show that $L=\{0^n | n$ is a perfect square $\}$ is not regular.

Answer :

For Answer refer to example - 9 , Page No : 3.44.

Q42. List and prove various closure properties of regular sets.

Answer :

For Answer refer to Topic : 3.9 , Page No : 3.44.

OBJECTIVE TYPE QUESTIONS

1. Find the regular expression for the set of all strings over $\{a,b\}$ in which there are atleast two occurrences of b between any two occurrences of a.

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

   (c) $b^*+(b+abb)^*ab^*$ (d) None of the above.

2. $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)=?$

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

   (c) $0^*1(0+10^*1)^*$ d) None of the above.

3. The empty string is the string with:

   (a) zero occurrence of symbol (b) non zero occurrence of symbol

   (c) no occurrence of symbols (d) None of the above

4. Which of the following regular expressions over {0,1} denotes the set of all string not containing 100 as a substring?

   (a) 0^*1010^* (b) $0^*(1^*0)^*$

   (c) $0^*1^*01^*$ (d) $0^*(10+1)^*$

5. Find the regular expression for the set of all strings having atmost one pair of 0's or atmost one pair of 1's

   (a) $(1+00)^*+(1+01)^*(1+10)^*+(1+11)^*+(0+10)^*11(0+10)^*$

   (b) $(1+01)^*+(1+00)^*(1+10)^*+(1+10)^*+(1+10)^*11(0+10)^*$

   (c) $(1+01)^*+(1+01)^*00(1+01)^*+(0+10)^*+(0+10)^*11(0+10)^*$

   (d) None of the above.

6. $\in+1^*(011)^*(1^*(011)^*)^*=?$

   (a) $1^*(011)^*$ (b) $(1+011)^*$

   (c) $1^*01^*(1+011)^*$ (d) None of the above.

7. Find the regular expression for the set of all strings of the form vw where a's occur in pairs in v and b's occur in pairs in w.

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

   (c) $(aa+b)^*(bb+a)^*$ (d) None of the above.

8. The intersection of $(a+b)^*a$ and $b(a+b)^*$ is given by

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

(c) $(a+b)^*ab(a+b)^*$ d) $b(a+b)^*a$

9. Which one is false

(a) $(r_1+r_2)^* = (r_1^* r_2^*)^*$ (b) $(r^*)^* = r^*$

(c) $r_1^*(r_1+r_2)^* = (r_1+r_2)^*$ (d) none.

10. The set of regular languages over a given alphabet set is not closed under

(a) Intersection (b) union

(c) Complement (d) none

11. Which of the following pairs are equivalent

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

(c) $(a+b)^*$ and (a^*+b^*) (d) None

12. The language of all words with at least 2 a's can be described as

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

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

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

(d) all

13. Which of the following pairs are not equivalent

(a) x^+ and x^*x^+ (b) $(ab)^*$ and a^*b^*

(c) $x(xx)^*$ and $(xx)^*x$ (d) $1(01)^*$ and $(10)^*1$

14. Let L may be language. Define even(w) as the string obtained by extracting from w the letters in even numbered positions i.e., if $w = a_1a_2a_3a_4.....$, then even(w) $= a_2a_4.....$ Corresponding to this, we can define a language : even(L) = { even(w) : $w \in L$} then given L is regular, even(L) is

(a) is not context free

(b) context free

(c) must be regular

(d) may not be regular

15. Which is the correct regular expression for the language : "Set of strings over alphabet $\{a,b,c\}$ containing at least 1 'a' and at least 1 'b'

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

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

(c) $(a+b+c)^*[a(a+b+c)^*b + b(a+b+c)^*a](a+b+c)^*$

(d) none of these.

16. Which is the correct order of precedence of regular expression operators in increasing order?

(a) *,( ),+,... (b) ( ),.,*,+ (c) *,( ),....,+ (d) ( ),*,....,+

17. Which of the following is accepted by $L(aa^* + aba^*b^*)$

(a) abab (b) aaab (c) abba (d) None.

18. Let r_1 *and* r_2 are regular expression and let $\cup$ stands for equivalence in the sense of the language generated, then

(a) $r_1(r_1 + r_2)^* \equiv (r_1 + r_2)^*$ (b) $(r_1^2 + r_2)^* \equiv (r_1^* r_2^*)^*$

(c) $(r_1^*)^* \equiv r_1$ (d) None of these

19. Regular expression for the language, $L = \{w \in \{0,1\}^* : w$ has no pair of consecutive zeros$\}$ is

(a) $r = (1+01)^*(0+1^*)$ (b) $r = (1^*011^*)^*(0+A)+1^*(0+A)$

(c) $r = (1+01)^*(0+A)$ (d) all of these

20. For $L(r) = \{a, bb, aa, abb, ba, bbb, ...\}, r$ is given by

(a) $r = (a+b)^*(a+bb)$ (b) $r = (aa+b)(a+b)^*$

(c) $r = (a+bb)^*$ (d) $r = a(a+bb)^*$

21. A language $L = \{awa : w \in \{a,b\}^*\}$ is

(a) context sensitive (b) regular

(c) context free (d) none of these

22. The value of the relation $A + RR^*$ is

(a) R^* (b) ϕ (c) $\in$ (d) R

23. The value of the relation $(R^*)^*$ is

(a) $\in$ (b) R (c) R^* (d) None of the above

24. The value of the relation ϕ^* is

(a) ϕ (b) Σ (c) $\in$ (d) None of the above

25. The value of the relation A^* is

(a) ϕ (b) Σ (c) $\in$ (d) None of the above

26. The value of the relation $R \in$ is

(a) ϕ (b) R (c) $\in$ (d) None of the above

27. The value of the relation $\in R$ is

(a) ϕ (b) R (c) $\in$ (d) None of the above

28. The value of the relation $R\phi$ is

(a) ϕ (b) R (c) $\in$ (d) None of the above

29. The value of the relation ϕR is

(a) ϕ (b) R (c) $\in$ (d) None of the above

30. The value of the relation $\phi + R$ is

(a) ϕ (b) R (c) $\in$ (d) None of the above

31. Which of the following identities for regular expression does not hold good?

(a) $(R+S)^* = R^* + S^*$ (b) $(R^*S^*)^* = (R+S)^*$

(c) $(\in + R^*) = R^*$ (d) $(R^*)^* = R^*$

32. ϕ^* (Kleene's closure of ϕ)(ϕ is the empty language over Σ) is equivalent to

(a) Σ (b) ϕ (c) $\in$ (d) none of these.

33. Give English description of the language of the regular expression : $(1+\in)(00^*1)^*0^*$

(a) alternating 1's and 0's (b) 0's only in pairs

(c) no pair of consecutive 1's (d) set of all strings of 0's and 1's containing

34. Let L be the language $\{\in,0,10\}$ over $\{0,1\}$. Determine the set $L \cup \overline{L}$.

(a) $\{0,1\}^*$ (b) ϕ

(c) same as the given set (d) None of the above

35. The regular expression representing the set of all strings over $\{x, y\}$ ending with xx beginning with y

(a) $x(x+y)^* yy$ (b) $y(x+y)^* xx$ (c) $yy(x+y)^* x$ (d) $xx(x+y)^* y$

36. How many strings of length t are in Σ^* if Σ is an alphabet of cardinality r.

(a) r + t (b) tr (c) rt (d) None of the above

37. Which of the following doesn't hold?

(a) $\in +1^*(011)^*(1^*(011)^*)^* = \in +1^*(0111^*)$

(b) $(111^*)^* = (11+111)^*$

(c) $(1+0)^* = 1^*(01^*)^*$

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

38. A solution to the equation $R = Q + RP$ is

(a) $R = PQ^*$ (b) $P = RQ^*$ (c) $Q = RP^*$ (d) $R = QP^*$

39. The value of the relation $(P^* + Q^*)^*$ is

(a) $(P^*Q^*)^*$ (b) Σ^* (c) P^*Q^* (d) None of the above

40. The value of the relation $(P+Q)^*$ is

(a) $P^* + Q^*$ (b) $(P^*Q^*)^*$ (c) P^*Q^* (d) Σ^*

41. The value of the relation $R + R$ is

(a) R^* (b) ϕ (c) $\in$ (d) R

42. The value of the relation $RR^* + \in$ is

(a) R^* (b) ϕ (c) $\in$ (d) R

43. In English language the set represented by a^*b+b^*a is:

(a) Strings of a's followed by one b and strings of b's followed by one a.

(b) String containing single a and single b

(c) Strings of a's followed by one b or strings of b's followed by one a.

(d) String containing single a or single b

44. The regular expression representing the set of all strings over $\{a,b\}$ with three consecutive b's

(a) $(a+b)^*bbb(a+b)$ (b) $(a+b)^*bb(a+b)^*$

(c) $(a+b)bbb(a+b)^*$ (d) $(a+b)^*bbb(a+b)^*$

45. For the following conditions find all the strings over the alphabet {a, } that satisfy the condition, (i) no symbol is repeated in the string and (ii) the length of string is 3.

(a) No such string is possible (b) All possible strings of length 3

(c) Only single string ab (d) None of the above.

46. Find the true statement

(a) If R is regular expression then so is R^*

(b) If R and S are regular expression then so is $R \cup S$

(c) If R and S are regular expression then so is R.S

(d) All of the above.

47. Find the true statement

(a) ϕ represents empty word, $\in$ represents empty language.

(b) $\in$ represents empty word, ϕ represents empty language

(c) $\in, \phi$ represents empty word

(d) $\in, \phi$ represents empty language

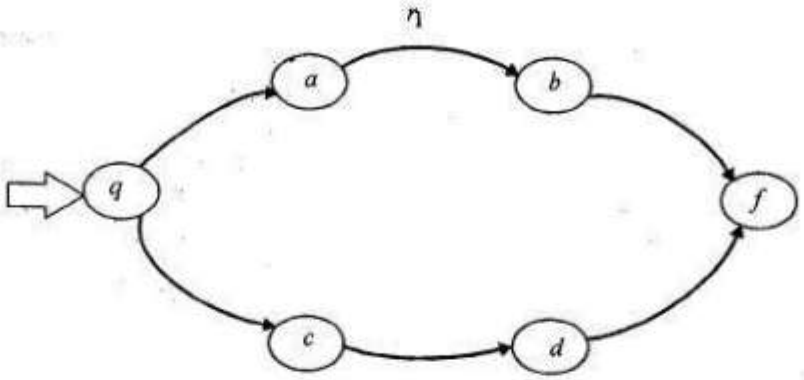
48. Regular expression for the set $\{a^2, a^5, a^8, ...\}$ is :

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

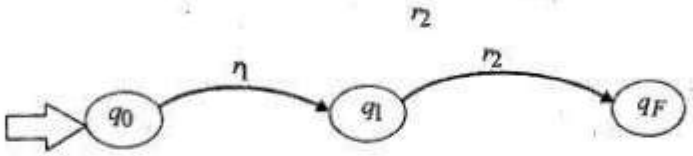
(c) $aa(aaaa)^*$ (d) $aa(aaa)^*$

49. Let r_1 *and* r_2 are regular expression which of the following represent $r_1 + r_2$

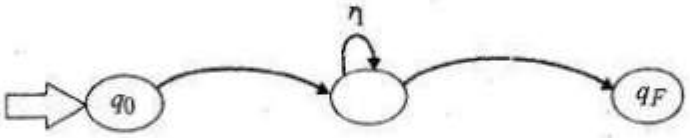
(a)



(b)



(c)



(d) none.

50. $01^* + 0$ is represented by

(a) $(1^*(0+0))$

(b) $(0(1^* + 0))$

(c) $((01)^* + 0)$

(d) $((0(1^*)) + 0)$

51. Which of the following identities doesn't hold?

(a) $R^*.R^* = R^*$

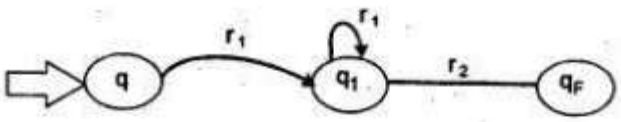
b) $(R \cup S)^* = (R^*.S^*)^*$

(c) $(R^* \cup S^*)^* = (R.S)^*$

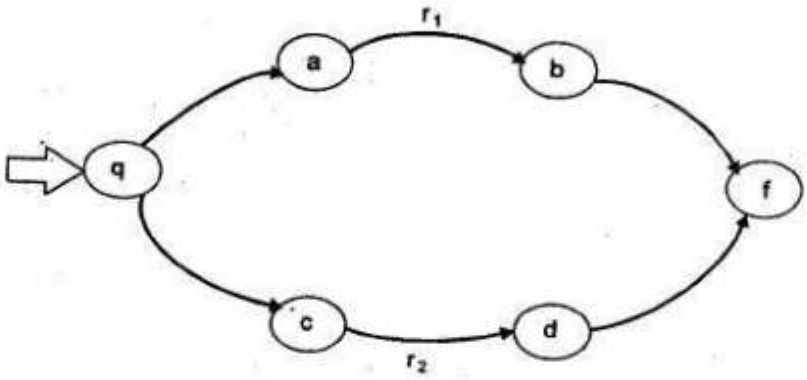
d) $(R \cup S)^* = (R^* \cup S^*)^*$

52. Let r_1 *and* r_2 are regular expression which of the following represent r_1

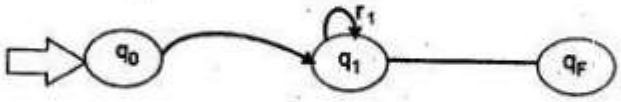
(a)



(b)



(c)



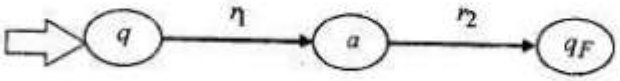
(d) none

53. Which of the following is false

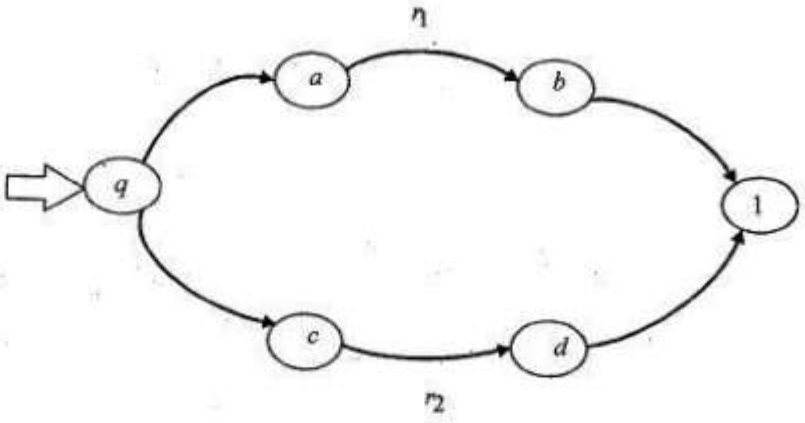
(a) $RR^* = R^*R$ (b) $R + R = R$ (c) $(R^*)^* = R^*$ (d) none.

54. Let r_1 *and* r_2 are regular expression which of the following represent $r_1.r_2$.

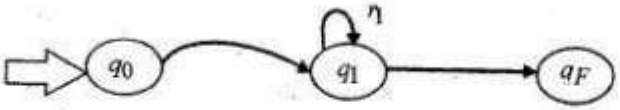
(a)



(b)



(c)



(d) none

55. Which of the following are correct

(a) If L^* is regular then L is regular

(b) If $L_1 \cup L_2$ is regular and L_1 is regular, then L_2 is regular,

(c) If L_1L_2 is regular and L_1 is regular, then L_2 is regular.

(d) all

56. Which of the following is set of strings of the form vw where a's occur in pairs in v and b's occur in pairs in w.

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

(c) $a^*b + b^*a$ (d) $a(a + b)^*ab$

57. The set of all strings which are either strings of a's followed by one b or strings of b's followed by one a.

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

(c) $a^*b + b^*a$ (d) $a(a + b)^*ab$

58. Select which of following represent a set of all strings with a and ending with ab.

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

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

59. $(a^*ab+ba)^*a^*$ is equivalent to

(a) $(a+b+ab)^*$ (b) $(aba+bab)^*$

(c) $(a+ab+ba)^*$ (d) $(ab+ba+aba)$

60. The two regular expressions are equivalent i.e., $\in +(a+b)^*b(a+b)^* = [a^*b(a^*ba^*b)^*a^*]^*$

(a) True (b) False.

61. The set all strings of 0's and 1's such that every pair of adjacent 0's appears before any pair of adjacent 1's

(a) $(10+0)^*(epsilon+1)(01+01)^*(epsilon+0)$

(b) $(10+0)^*(epsilon+1)(01+1)^*(epsilon+0)$

(c) $(10+0)^*(epsilon+1)^*(epsilon+0)$

(d) $(100)^*(epsilon+1)(01+1)^*(epsilon+0)$

62. Write the regular expression for the following:

"The set of the strings over alphabet $\{a,b,c\}$ containing at least one a and at least one b"

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

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

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

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

63. The reversal of the language $L=\{001,10,111\}$ *is* :

(a) {111,01,110} (b) {100,01,111}

(c) {111,10,001} (d) none

64. $(L^*)^*$ equal to :

(c) (L^{**}) (a) L^* (b) L^{**} (d) none

65. The language generated by the regular expression $(aa)^*(bb)^*b$ is

(b) $a^{2n}b^{2n+1}$ (a) $(ab)^{2n}b$ (c) none of these.

66. $(1^*011^*)^*(0+\epsilon)$ is equivalent to

(b) $(0+1)(1+10)^*$ (a) $(1+01)^*(0+\epsilon)$ (c) none of these

67. Which of the following identity doesn't hold?

(a) $\phi + R = R + \phi = R$ (b) $\phi R = R\phi = \phi$

(c) $\epsilon + R = R + \epsilon = R$ (d) $\epsilon R = R\epsilon = R$

68. The language of all words that have at least one a and at least one b is

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

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

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

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

69. Let a and b be two regular expressions then $(a^* \cup b^*)^*$ is equivalent to

(a) $a \cup b$ (b) $(b \cup a)^*$ (c) $(b^* \cup a^*)^*$ (d) $(a \cup b)^*$

70. If e_1 *and* e_2 are regular expressions denoting the languages L_1 and L_2 respectively, then which is false?

(a) $(e_1)^*$ is a regular expression denoting L_1^*

(b) ϕ is not a regular expression

(c) $(e_1)(e_2)$ is a regular expression denoting L_1L_2

(d) $(e_1)(e_2)$ is a regular expression denoting $L_1 \cup L_2$

71. What is the regular expression defining the language of all words with an odd number of b's is

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

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

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

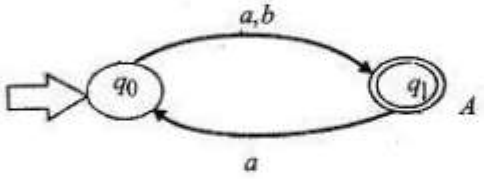
(d) None of these.

72. $(1+0)^*$ represents
(a) Set of strings over 1 and 0.
(b) Set of strings starting with 1 and ending with 0
(c) Set of strings with equal number of 1's and 0's
(d) Set of strings with even number of 1's and 0's

73. Which one is TRUE
(a) $\{1010\}$ *belongs to* $(10)^*$
(b) $(10)^* = 1^* + 0^*$
(c) $(10)^* = (1^*0^*)^*$
(d) $(10)^* = 1^*0^*$

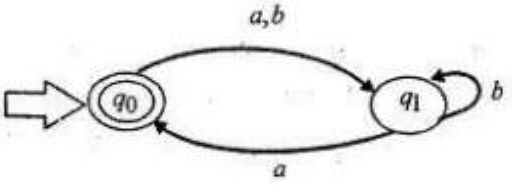
74. The $R.E. = (10+01+11+00)^*$ represents
(a) set of strings with at least one 0 and at least one 1
(b) set of strings with even length
(c) set of strings with equal 0 and 1's
(d) All strings over 0 and 1

75. Regular expression generated by the following automaton is given as



(a) $(a+b)(ab+aa)^*$
(b) $(a+b)(ab+aa)^*a$
(c) $\in +(a+b)(ab+aa)^*a$
(d) $\in +(a+b)(ab+aa)^*$

76. Regular expression generated by the following automaton is given as:



(a) $(a+b)(b+ab+aa)^*a$
(b) $\in +(a+b)(b+ab+aa)a$
(c) $(a+b)(b+ab+aa)^*$
(d) $\in +(a+b)(b+ab+aa)^*a$

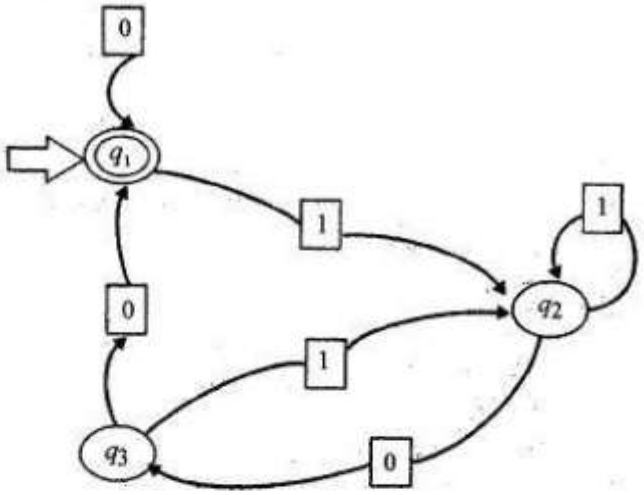
77. Regular expression corresponding to the finite automaton drawn below is given by

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

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

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

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



78. The regular expression $(1+00^*1)+(1+00^*1)(0+0+10^*1)^*(0+10^*1)$ is equivalent to

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

(c) $(1+00^*1)(0+10^*1)^*$ (d) All of the above.

79. Which of the following is regular?

(a) String of odd number of zeroes.

(b) Strings of 0's, whose length is a prime number

(c) String of all palindromes made up of 0's and 1's

(d) String of 0's whose length is a perfect square

80. The recognizing capability of NDFA and DFA

(a) must be same (b) may be different

(c) must be different (d) none of the above.

81. The intersection of the two regular languages below:

$L_1 = (a+b)^*a$ and $L_2 = b(a+b)^*$ is given by

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

(c) $(a+b)^*ab(a+b)^*$ (d) $b(a+b)^*a$

82. Which of the following regular expression over {0,1} denotes the set of all string not containing 100 as a substring?

(a) $0^*(10+1)^*$ (b) 0^*101^* (c) 0^*1010^* (d) $0^*(1^*0)^*$

83. The set all strings over $\{a,b\}$ in which there are atleast two occurrences of b between the two occurrences of a.

(a) $a(bbabb)^* a$ (b) $b^*(b+ab)^* b^*$

(c) $b^* + (b+abb)^* ab^*$ (d) None of the above.

84. Set of all strings over $\{0,1\}$ having atmost one pair of 0's or atmost one pair of 1's.

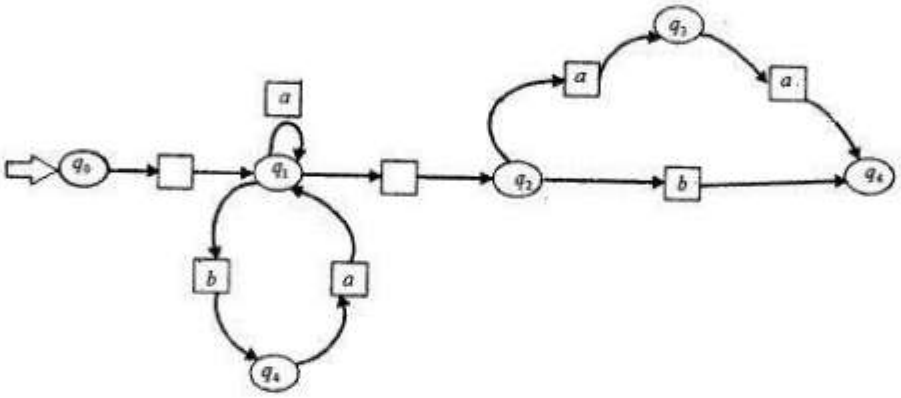
(a) $(1^* + (01)^*)^* + (1^*)(01)^* 00(1^*(01)^*)^* + (0^*)(10)^* + ((0^*) + (10)^*)^* 11(0+10)^*$

(b) $(1+01)^* + (1+01)^* 00(1+01)^* + (0+10)^* + (0+10)^* 11(0+10)^*$

(c) $(1+01)^* + (0+10)^* 00(0+10)^* + (0+10)^* + (1+01)^* 11(1+01)^*$

(d) None of the above.

85. Regular expression corresponding to the FA given below is



(a) $a^* + (ab+a)^*$ (b) $(ab+a)^*(aa+b)$

(c) $(a^* b + b^* a)^*$ (d) None of the above.

86. Which of the following closure properties hold for regular sets?

(i) If L is regular, then L^T is also regular.

(ii) If L is regular set over Σ, then $\Sigma^* - L$ is also regular over $\cap$.

(iii) If X and Y are regular sets over Σ, then X intersection Y is also regular over Σ

(a) Only (i), (ii) and (iii). (b) Only (i) and (iii)

(c) Only (ii) (d) Only (i)

87. If L is an infinite regular language, then there exist some positive integer m such that any string $w \in L$ whose length is m or greater can be decomposed into three part, xyz, where

(a) $w = xy^i z$ is also in L for all i = 0,1,2,3,. (b) $|xy|$ *is less than or equal to m.*

(c) $|y| > 0$.. (d) All of the above.

88. If L is the language $L(01^*2)$, what is h(L):

(a) aba^*ab (b) $aab(ba)^*$ (c) aab^*ba (d) $a(ab)^*ba$

89. The inverse homomorphism of a regular language is :

(a) not regular (b) regular (c) none

A homomorphism is a function from some alphabet Σ_1 to strings in another alphabet Σ_2. If $x = a_1a_2,...a_k \in \Sigma_1^*$, *then* $h(x) = h(a_1)h(a_2)....h(a_k)$, *and if* $L \subseteq \Sigma^*$, then $h(L) = \{h(x) / x \in L\}$.

90. Suppose h is the homomorphism from the alphabet {0,1,2} to the alphabet {a,b} defined by: $h(0) = a$; $h(1) = ab$, *and* $h(2) = ba$ .What is $h(0120)$?

(a) ababa (b) abbbb (c) aaabb (d) aabba

91. If L is regular, then {x: reverse (x) in L} is also regular

(a) May or may not be (b) Yes

(c) No (d) None of the above.

92. Finite state machines..... can recognize palindromes

(a) may not (b) may (c) can't (d) can

93. Pick the correct statement. The logic of Pumping lemma is a good example of

(a) Iteration (b) Recursion

(c) The divide and conquer technique (d) The Pigeon hole principle

94. Which of the following is not regular

(a) String of zero whose length is prime

(b) String of zero whose length is perfect square

(c) Set of palindromes over 0 and 1

(d) All

95. Pumping lemma can be used

(a) Whether two languages are equivalent

(b) To check whether a language is regular

(c) To check whether a language is irregular

(d) None.

96. Let L be a regular language defined are Σ^*. Then

(a) index (R_L) may be zero (b) index (R_L) is finite

(c) index (R_L) may be infinite (d) None.

97. Let $\Sigma = \{0,1\}$ and R be the relation defined on Σ^* by $(x, y) \in R$ iff $|x| - |y| = odd$ Then R is

(a) not a right congruence (b) an equivalence relation

(c) a right congruences (d) none.

98. Let $\Sigma = \{a\}$ and let I be the identify relation on Σ^*. Let $L = \{\in\} \cup \{a\} \cup \{aa\}$. Then index (I) is

(a) 3 (b) finite (c) infinite (d) None.

99. Is there a finite automation which accepts all palindromes over {a,b}?

(a) No,but it cannot be proved. (b) No, it can be proved.

(c) Yes, but it cannot be proved (d) Yes,it can be proved

100. Which of the following sets is regular?

(a) $\{a^{2n} \mid n \geq 1\}$ (b) $\{ww \mid w \in \{a,b\}^*\}$

(c) $\{a^p \mid p \text{ is a prime}\}$ (d) $\{a^{i^2} \mid i \geq 1\}$

101. Which of the following languages cannot be produced by a regular grammar?

(i) $\{a^n b^n : n \geq 0\}$

(ii) $\{a^m b^k : k > n \geq 0\}$

(iii) $\{ww^R : w \in \{a,b\}^*\}$

(a) (ii) and (iii) (b) (i)

(c) (i) and (ii) (d) All of the above.

ANSWER KEY

1(c)	2 (b)	3 (a)	4 (d)	5 (c)	6 (a)	7 (c)	8 (d)	9 (a)
10 (d)	11 (c)	12 (b)	13 (b)	14 (c)	15 (c)	16(b)	17(a)	18(b)
19.(d)	20.(a)	21.(a)	22.(a)	23.(c)				
24.(c)	25.(c)	26.(b)	27.(b)	28.(a)	29.(a)	30.(b)	31.(a)	
32(c)	33.(a)	34.(b)	35.(b)	36.(c)	37.(a)	38.(d)	39.(a)	
40.(b)	41.(a)	42.(a)	43.(c)	44.(d)	45.(a)	46.(d)	47.(b)	
48.(d)	49.(a)	50.(d)	51.(c)	52.(d)	53.(d)	54.(a)	55.(c)	
56.(b)	57.(c)	58.(d)	59.(c)	60.(a)	61.(b)	62.(d)	63.(b)	
64.(b)	65.(a)	66.(b)	67.(c)	68.(a&b)	69.(d)	70.(b)		
71.(a)	72.(a)	73.(a)	74.(b)	75.(a)	76.(d)	77.(d)	78.(b)	
79.(a)	80.(a)	81.(d)	82.(a)	83.(c)	84.(b)	85.(b)	86.(a)	
87.(d)	88.(d)	89.(b)	90.(d)	91.(b)	92.(c)	93.(d)	94.(d)	
95.(c)	96.(b)	97.(a)	98.(c)	99.(b)	100.(a)	101.(d)		

4

REGULAR GRAMMARS

After going through this chapter, you should be able to understand :

- Regular Grammar
- Equivalence between Regular Grammar and FA
- Interconversion

4.1 REGULAR GRAMMAR

Definition : The grammar G = (V, T, P, S) is said to be regular grammar iff the grammar is right linear or left linear.

A grammar G is said to be right linear if all the productions are of the form

$A \rightarrow wB$ and / or $A \rightarrow w$ where $A, B \in V$ and $w \in T^*$.

A grammar G is said to be left linear if all the productions are of the form

$A \rightarrow Bw$ and / or $A \rightarrow w$ where $A, B \in V$ and $w \in T^*$.

Example 1 : The grammar

$S \rightarrow aaB \mid bbA \mid \in$

$A \rightarrow aA \mid b$

$B \rightarrow bB \mid a \mid \in$

is a right linear grammar. Note that $\in$ and string of terminals can appear on RHS of any production and if non - terminal is present on R. H. S of any production, only one non - terminal should be present and it has to be the right most symbol on R. H. S.

Example 2 :

The grammar

$S \rightarrow Baa \mid Abb \mid \in$

$A \rightarrow Aa \mid b$

$B \rightarrow Bb \mid a \mid \in$

is a left linear grammar. Note that $\in$ and string of terminals can appear on RHS of any production and if non - terminal is present on L. H. S of any production, only one non - terminal should be present and it has to be the left most symbol on L. H. S.

Example 3 :

Consider the grammar

S → aA
A → aB | b
B → Ab | a

In this grammar, each production is either left linear or right linear. But, the grammar is not either left linear or right linear. Such type of grammar is called linear grammar. So, a grammar which has at most one non terminal on the right side of any production without restriction on the position of this non - terminal (note the non - terminal can be leftmost or right most) is called linear grammar.

Note that the language generated from the regular grammar is called regular language. So, there should be some relation between the regular grammar and the FA, since, the language accepted by FA is also regular language. So, we can construct a finite automaton given a regular grammar.

4.2 FA FROM REGULAR GRAMMAR

Theorem : Let G = (V, T, P, S) be a right linear grammar. Then there exists a language L(G) which is accepted by a FA. i. e., the language generated from the regular grammar is regular language.

Proof : Let $V = (q_0, q_1, ...)$ be the variables and the start state $S = q_0$ Let the productions in the grammar be

$$q_0 \rightarrow x_1 q_1$$
$$q_1 \rightarrow x_2 q_2$$
$$q_2 \rightarrow x_3 q_3$$
$$\vdots$$
$$q_n \rightarrow x_n q_n$$

Assume that the language L(G) generated from these productions is w. Corresponding to each production in the grammar we can have a equivalent transitions in the FA to accept the string w. After accepting the string w, the FA will be in the final state. The procedure to obtain FA from these productions is given below :

Step 1 : q_0 which is the start symbol in the grammar is the start state of FA.

Step 2 : For each production of the form

$$q_i \rightarrow wq_j$$

the corresponding transition defined will be

$$\delta^*(q_i, w) = q_j;$$

Step 3 : For each production of the form $q_i \rightarrow w$

the corresponding transition defined will be $\delta^*(q_i, w) = q_f$, where q_f is the final state,

As the string $w \in L(G)$ is also accepted by FA, by applying the transitions obtained from step1 through step3, the language is regular. So, the theorem is proved.

Example 1 : Construct a DFA to accept the language generated by the following grammar

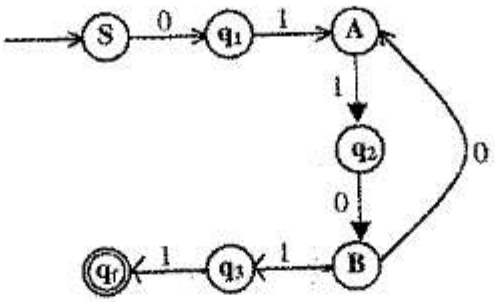
$$S \rightarrow 01A$$
$$A \rightarrow 10B$$
$$B \rightarrow 0A \mid 11$$

Solution :

Note that for each production of the form A → wB, the corresponding transition will be $\delta(A, w) = B$. Also, for each production $A \rightarrow w$, we can introduce the transition $\delta(A, w) = q_f$ where q_f is the final state. The transitions obtained from grammar G is shown using the following table :

Productions			Transitions
S	→	01A	$\delta(S, 01) = A$
A	→	10B	$\delta(A, 10) = B$
B	→	0A	$\delta(B, 0) = A$
B	→	11	$\delta(B, 11) = q_f$

The FA corresponding to the transitions obtained is shown below :



So, the DFA $M = (Q, \Sigma, \delta, q_0, A)$ where

$Q = \{ S, A, B, q_f, q_1, q_2, q_3 \}$, $\Sigma = \{0,1\}$

$q_0 = S$, $A = \{q_f\}$

δ is as obtained from the above table.

The additional vertices introduced are q_1, q_2, q_3.

Example 2 : Construct a DFA to accept the language generated by the following grammar.

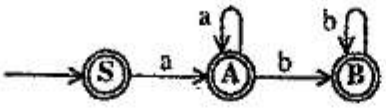
S	→	**aA** | ∈
A	→	**aA| bB** | ∈
B	→	**bB** | ∈

Solution :

Note that for each production of the form $A \to wB$, the corresponding transition will be $\delta(A, w) = B$. Also, for each production $A \to w$, we can introduce the transition $\delta(A, w) = q_f$ where q_f is the final state. The transitions obtained from grammar G is shown using the following table:

Productions			Transitions
S	→	aA	$\delta(S, a) = A$
S	→	∈	S is the final state
A	→	aA	$\delta(A, a) = A$
A	→	bB	$\delta(A, b) = B$
A	→	∈	A is the final state
B	→	bB	$\delta(B, b) = B$
B	→	∈	B is the final state.

Note : For each transition of the form $A \to \in$, make A as the final state.
The FA corresponding to the transitions obtained is shown below :



So, the DFA $M = (Q, \Sigma, \delta, q_0, A)$ where

$Q = \{S, A, B\}$, $\Sigma = \{a, b\}$

$q_0 = S$, $A = \{S, A, B\}$

δ is as obtained from the above table .

4.3 REGULAR GRAMMAR FROM FA

Theorem : Let $M = (Q, \Sigma, \delta, q_0, A)$ be a finite automaton. If L is the regular language accepted by FA, then there exists a right linear grammar G = (V, T, P, S) so that L = L(G).

Proof : Let $M = (Q, \Sigma, \delta, q_0, A)$ be a finite automata accepting L where

$Q = \{q_0, q_1, \ldots q_n\}$

$\Sigma = \{a_1, a_2, \ldots a_m\}$

A regular grammar G = (V, T, P, S) can be constructed where

$V = \{q_0, q_1, \ldots q_n\}$

$T = \Sigma$

$S = q_0$

The productions P from the transitions can be obtained as shown below :

Step 1 : For each transition of the form $\delta(q_i, a) = q_j$

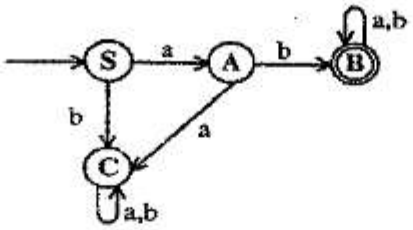
the corresponding production defined will be $q_i \to aq_j$

Step 2 : If $q \in A$ i. e., if q is the final state in FA, then introduce the production

$q \to \in$

As these productions are obtained from the transitions defined for FA, the language accepted by FA is also accepted by the grammar.

Example 1 : Construct a regular grammar from the following FA.



Solution : Note that for each transition of the form $\delta(A, a) = B$, introduce the production $A \rightarrow aB$. If $q \in A$ i. e., if q is the final state, introduce the production $A \rightarrow \in$. The productions obtained from the transitions defined for FA is shown using the following table :

Transitions	Productions
$\delta(S, a) = A$	S → aA
$\delta(S, b) = C$	S → bC
$\delta(A, a) = C$	A → aC
$\delta(A, b) = B$	A → bB
$\delta(B, a) = B$	B → aB
$\delta(B, b) = B$	B → bB
$\delta(C, a) = C$	C → aC
$\delta(C, b) = C$	C → bC

It is very important to note that B is the final state. So, we have to introduce the production $B \rightarrow \in$. The grammar G corresponding to the productions obtained is shown below :

Grammar G = (V, T, P, S) where

V = { S, A, B, C }

T = { a, b }

P = {

$S \rightarrow aA \mid bC$

$A \rightarrow aC \mid bB$

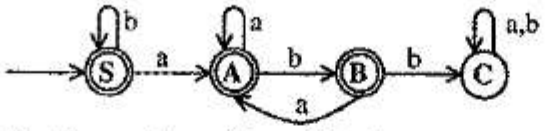
$B \rightarrow aB \mid bB \mid \in$

$C \rightarrow aC \mid bC$

}

S is the start symbol

Example 2 : Construct a regular grammar for the following FA.



Solution : Note that for each transition of the form $\delta(A, a) = B$, introduce the production $A \rightarrow aB$. If $q \in A$ i.e., if q is the final state, introduce the production $A \rightarrow \in$. The productions obtained from the transitions defined for FA is shown using the following table :

Transitions	Productions
$\delta(S, a) = A$	S → aA
$\delta(S,b)=S$	S → bS
$\delta(A,a)=A$	A → aA
$\delta(A, b) = B$	A → bB
$\delta(B,a)=A$	B → aA
$\delta(B,b)=C$	B → bC
$\delta(C, a) = C$	C → aC
$\delta(C, b) = C$	C → bC

It is very important to note that S, A and B are final states. So, we have to introduce the productions $S \rightarrow \in$, $A \rightarrow \in$ and $B \rightarrow \in$. The grammar G corresponding to the productions obtained is shown below :

Grammar G = (V, T, P, S) where

V = { S, A, B, C }

T = { a, b }

P = {
S → aA | bS | ∈
A → aA | bB | ∈
B → aA | bC | ∈
C → aC | bC
}

S is the start symbol

Note : The FA in this problem accepts strings of a's and b's except those containing the substring abb. So, from the grammar G we can obtain a regular language which consists of strings of a's and b's without the substring abb.

REVIEW QUESTIONS

Q1. What is regular grammar ? Explain with examples.

Answer :

For Answer refer to Topic 4.1 , Page No : 4.1.

Q2. Explain procedure to construct FA from regular grammar.

Answer :

For Answer refer to Topic : 4.2 , Page No : 4.2.

Q3. Construct a DFA to accept the language generated by the following grammar

$$S \rightarrow 01A$$

$$A \rightarrow 10B$$

$$B \rightarrow 0A \mid 11$$

Answer :

For Answer refer to example - 1 , Page No : 4.3.

Q4. Construct a DFA to accept the language generated by the following grammar .

$$S \rightarrow aA \mid \in$$

$$A \rightarrow aA \mid bB \mid \in$$

$$B \rightarrow bB \mid \in$$

Answer :

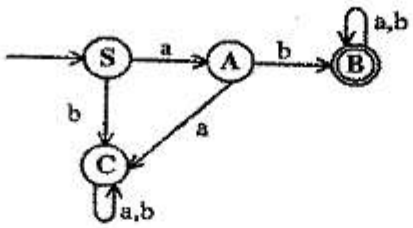
For Answer refer to example - 2 , Page No : 4.4.

Q5. Explain procedure to obtain Regular grammar from FA.

Answer :

For Answer refer to Topic : 4.3 , Page No : 4.5.

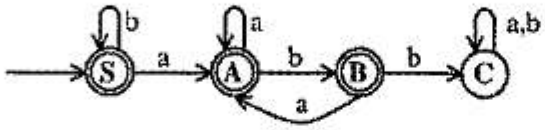
Q6. Construct a regular grammar from the following FA.



Answer :

For Answer refer to example - 1 , Page No : 4.6.

Q7. Construct a regular grammar for the following FA.



Answer :

For Answer refer to example - 2 , Page No : 4.7.

OBJECTIVE TYPE QUESTIONS

1. The grammar generated by production rule $S \to aCa, C \to aCab$ is,

   (a) $a^n a^n, n > 0$ (b) $a^n b a^n, n > 0$

   (c) $a^n a^n, n \geq 0$ (d) None of the above

2. The language $L\{0^n 1^n 2^k 3^k\}$ is a

   (a) Recursively enumerable language. (b) Regular language

   (c) CSL (d) CFL

3. The set $\{a^n b^n\}$ can be generated by the CFG

   (a) $S \to ab \mid asB \mid E$ (b) $S \to ab \mid aSb$

   (c) $S \to aaSbb$ (d) None of the above

4. Chomsky hierarchy from type 0 to type 3 is:

   (a) $L_{RL}, L_{CSL}, L_{CF}, L_{RE}$ (b) $L_{RE}, L_{CFL}, L_{CSL}, L_{RL}$

   (c) $L_{RE}, L_{CSL}, L_{CFL}, L_{RL}$ (d) $L_{RL}, L_{CFL}, L_{CSL}, L_{RE}$

5. Which of the following relationship holds?

   (a) $L_{CF} \subset L_{DC} \supset L_{RL}$ (b) $L_{CF} \supset L_{DC} \subset L_{RL}$

   (c) $L_{CF} \supseteq L_{DC} \supseteq L_{RL}$ (d) $L_{CFL} \subseteq L_{DCS} \subseteq L_{RL}$

6. Which of the following statements is true?

   (a) $L_{RE} \subset L_{RC} \supset L_{CSL}$ (b) $L_{RE} \supset L_{RC} \subset L_{CSL}$

   (c) $L_{RE} \supset L_{RC} \supset L_{CSL}$ (d) $L_{RE} \subset L_{RC} \subset L_{CSL}$

7. Which of the following is true?

   (a) $L_0 \subseteq L_{CFF} \subseteq L_{CSL} \subseteq L_{RL}$ (b) $L_{RL} \subseteq L_{CSL} \subseteq L_{CFL} \subseteq L_0$

   (c) $L_{RL} \subseteq L_{CFL} \subseteq L_{CSL} \subseteq L_0$ (d) $L_0 \subseteq L_{CSL} \subseteq L_{CFL} \subseteq L_{RL}$

8. Which of the following is true?

   (a) L_0 is subset of L_{CFL} (b) L_{RL} is subset of L_{CFL}

   (c) L_0 is subset of L_{CSL}. (d) None of the above.

9. A grammar $G=\langle V,T,S,P\rangle$ is said to be context sensitive if all Productions are the form $x \to y$, where $x, y \in (V \cup T)^+$ and

(a) $|x| \geq |y|$ (b) $|x| = |y|$ (c) $|x| \leq |y|$ (d) None of the above

10. A grammar $G=\langle V,T,S,P\rangle$ is called unrestricted if all the productions are of the form $u \to v$, where,

(a) $u \in (V \cup T)$ *and* $v \in (V \cup T)^+$ (b) $u \in (V \cup T)^*$ *and* $v \in (V \cup T)^*$

(c) $u \in (V \cup T)^+$ *and* $v \in (V \cup T)^*$ (d) $u \in (V \cup T)^*$ *and* $v \in (V \cup T)$

11. The grammar that generates $L=\{a^n b^n c^i \mid n \geq 1, i \geq 0\}$ is,

(a) $S \to a \in b \mid Sc, \in \to ab / \in b$ (b) $S \to \in \mid Sc, \in ab \mid a \in b$

(c) Any one of the two (d) None of the two.

12. The grammar that generates $L=\{wew^l \mid w \in \{a,b\}^*\}$ *is,*

(a) $S \to aSa \mid bSb \mid c$ (b) $S \to aSa \mid bSb \mid aca \mid bcb$

(c) Any one of the two (d) None of the two.

13. Let G be the grammar $S \to aA, A \to A\,bb \mid b$, sentential forms of G are,

(a) aAb^{2n}, ab^{2n} *where*, $n \geq 0$ (b) aAb^{2n}, ab^{2n+1} *where* , $n \geq 0$

(c) aAb^{2n}, ab^{2n+1} *where*, $n \geq 1$ (d) None of above.

14. Which of the following grammars can generate $w = aabbb$

(a) $S \to AB, A \to BB \mid a, B \to AB \mid b$ (b) $S \to AB, A \to aA \mid u, B \to bB \mid b$

(c) Both (d) None.

15. The grammar having productions as $A \to B$, where $A \in B, B \in (V \cup \Sigma)$ is

(a) Type 3 (b) Type 2 (c) Type 1 (d) Type 0

16. The grammar generated by production rules $S \to aSBC \mid abc$, $cB \to Bc, aB \to aB \to aa$ is

(a) $a^n b^n c^n, n \leq 0$ (b) $a^n b^n c^n, n > 0$

(c) $a^n b^n c^n, n \geq 0$ (d) $a^n b^n c^n, n > 1$

17. If we have an algorithm to determine whether a given element belongs to a set X or not, then this set is called

(a) Context-sensitive (b) Complete

(c) Recursively Enumerable (d) Recursive

18. A production of the form $A \to \alpha$ where $A \in V$ *and* $\alpha \in (V \cup \Sigma)^*$ represents which type of grammar?

(a) Three (b) Two (c) One (d) Zero

19. What is the highest type number to grammar given by these production rules $S \to ASB \mid d, A \to aA$.

(a) Three (b) Two (c) One (d) Zero

20. The productions: $E \to E + E, E \to E - E, E \to E * E, E \to id$

(a) Are unambiguous

(b) Generated an inherently ambiguous language

(c) Generate an ambiguous language

(d) None.

21. The set $L = \{a^n b^n c^n\}$ is an example of grammar that is

(a) not context free (b) regular

(c) context free (d) none.

22. Choose the correct statements

(a) some regular languages can't be generated by an CFG.

(b) Some non regular languages can't be generated by an CFG

(c) Any regular language has not an equivalent CFG

(d) all languages can be generated by CFG

23. The grammar having production as $A \to B$, *Where* $A \in (V \cup \Sigma)^+, \mathbf{B} \in (V \cup \Sigma)^*$ is

(a) Type 3 (b) Type 2 (c) Type 1 (d) Type 0

24. The grammar having production as $A \to xB$, *Where* $A \in V$, $\mathbf{B} \in (V \cup \in), x \in \Sigma^*$ is

(a) Type 3 (b) Type 2 (c) Type 1 (d) Type 0

25. Chomsky Hierarchy is representation of
(a) Parsers (b) Grammars
(c) Machines (d) None of these

26. Find language generated by the following grammar $S \rightarrow 0S1 \mid 0A1 A \rightarrow 1A0 \mid 10$
(a) $\{0^n1^m0^m1^n \mid m,n \geq 1\}$ (b) $\{0^n1^n0^n \mid n > 1\}$
(c) $\{1^m0^n \mid m,n \geq 1\}$ (d) None of the above.

27. Type 2 grammar is also called
(a) context-free (b) Context-sensitive
(c) regular (d) none of these.

28. Which of the following language is context free?
(a) $a^nb^j : n = j^2$ (b) a (c) a^nb^n (d) $a^nb^nc^n$

29. Which of the following language is regular?
(a) phi (b) a^nb^n (c) $a^nb^nc^n$ (d) a^nb^n

30. A grammar is context-sensitive if all production of the form $x > y$ are such that
(a) $|x|>|y|$ (b) $|x|<|y|$ (c) $|x|\leq|y|$ (d) $|x|=|y|$

31. A recursive lenguage is also:
(a) deterministic (b) CFL
(c) recursive enumerable (d) regular

32. Which of the following grammars can generate $w = aabbb$
(a) $S \rightarrow AB, A \rightarrow BB \mid a, B \rightarrow AB \mid b$
(b) $S \rightarrow AB, A \rightarrow aA \mid a, B \rightarrow bB \mid b$
(c) Both
(d) None.

33. Which of the following is true for the language generated by $S \rightarrow AB, A \rightarrow BB \mid a, B \rightarrow AB \mid b$
(a) aab belongs to this language.
(b) ab doesn't belongs to this language
(c) aabb belongs to this language
(d) aabbb doesn't belongs to this language

34. Let $G_1 = (V_1, \Sigma, S_1, P_1)$ be right linear and $G_2 = (V_2, \Sigma, S_2, P_2)$ be left linear grammar and assume that V_1 and V_2 are disjoint. Consider, the linear grammar

$G = (\{s\} \cup V_1 \cup V_2, \Sigma, S, P)$, where S is not in $V_1 \cup V_2$

and $P = \{s \to s_1 \mid s_2\} \cup P_1 \cup P_2$, *then* $L(G)$ *is*

(a) not regular. (b) regular (c) left linear (d) right linear

35. A regular grammar is

(a) Either left linear or right linear (b) Neither left linear nor right linear
(c) Right linear and not left linear (d) Both left linear and right linear

36. A grammar $G = (V, T, S, P)$ is right linear if

(a) CF < context sensitive < right linear (b) $A \to xB \mid x; A, B \in V$ *and* $x \in T^*$
(c) None of these

37. A language L is accepted by a finite automaton if and only if

(a) recursive. (b) context sensitive
(c) primitive recursive (d) right linear

38. The correct relationship is given by

(a) right linear < CF < context sensitive. (b) context sensitive < CF < right linear
(c) CF < context sensitive < right linear (d) CF < right linear < context sensitive

39. Which of the following can be generated by: $S \to aS \mid bA, a \to d \mid ccA$

(a) ababccd (b) aabccd (c) abbbd (d) bccddd

40. Language generated by the grammar: $S \to 0A \mid 1s \mid 0 \mid 1, a \to 1A \mid 1S \mid 1$ *is*

(a) all strings of 0's and 1's that neither contain 2 consecutive 0's nor contains two consecutive 0's.
(b) all strings of 0's and 1's that does no contain 2 consecutive 1's
(c) all strings of 0's and 1's that does no contain 2 consecutive 0's
(d) all strings of 0's and 1's

41. Consider the grammar: $S \to aAB \mid abc$, $A \to aa \mid e$ B $\to b$ then equivalent representation is

(a) $S \to aB \mid abc$, $A \to aa$, B $\to b$ (b) $S \to aaaB \mid aB \mid abc$, $A \to aa$, B $\to b$
(c) $S \to aaaB \mid abc$, $A \to aa$, B $\to b$ (d) $S \to aAB \mid abc$, $A \to aa$, B $\to b$

42. Give the regular grammar that generates the following set : $\{(ab)^n \mid n \geq 1\}$

(a) $S \to S_1S_2, S_1 \to bS, S \to aS_2, S_2 \to b$ (b) $S \to aS_1, S_1 \to bS, S \to aS_2, S_2 \to b$

(c) $S \to aS_1S_2, S_1 \to bS, S_2 \to b$ (d) None of the above.

43. Give the regular grammar that generates the following set: $\{a^l b^m c^n \mid l, m, n\, n \geq 1\}$

(a) $S \to aS_1S_2, S_1 \to aS_1, S_1 \to bS_2, S_2 \to bS_2, S_2 \to cS_2S_3, S_3 \to c$

(b) $S \to aS_1, S_1 \to aS_1, S_1 \to bS_2, S_2 \to bS_2, S_2 \to cS_3, S_3 \to c$

(c) $S \cup aSS_1, S_1 \cup aS_1S_2, S_1 \cup bS_2, S_2 \cup bS_2, S_2 \cup cS_3, S_3 \cup c$

(d) None of the above.

44. Give the regular grammar that generates the following set: $\{a^{2n} \mid n \geq 1\}$

(a) $S \to aS_1S_2, S_1 \to aS, S_2 \to a$ (b) $S \to aS_1, S_1 \to aS, S \to aS_2, S_2 \to a$

(c) $S \to aS, S \to a$ (d) None of the above.

45. Find language generated by the following grammar: $S \to 0A \mid 1S \mid 1 \mid 0 \;\; A \to 1A \mid 1S \mid 1$

(a) $\{x \in \{0,1\}^*\}$

(b) $\{x \in \{0,1\}^* \mid x$ does not contain any two consecutive zeroes.$\}$

(c) None of the above.

46. A production of the form $A \to a$ or $A \to aB$ represents which type of grammar?

(a) Three. (b) Two (c) One (d) Zero

47. What is the highest type number to grammar given by these production rules: $S \to aS \mid ab$.

(a) Three. (b) Two (c) One (d) Zero

48. What is the highest type number to the grammar given by these production rules: $S \to Aa,\ A \to c \mid Ba,\ B \to abc$.

(a) Three. (b) Two (c) One (d) Zero

49. $L(G) = \{ww^R : w \in \{a,b\}^*\}$ *is*

(a) not context free (b) regular

(c) context free (d) none of these.

50. What language is generated by the following grammar $G=\langle S, \in, B\}, \{a,b\}, P, S\rangle$ where P is :

$S \rightarrow aB|bA$

$A \rightarrow a \mid as \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

(a)All words consisting of one b more than the number of a's.

(b) All words consisting of equal numbers of a's and b's.

(c) All words consisting of one a more than the number of b's

(d) None of the above

51. Find the false statement for k-equivalent states

(a) If q_1 and q_2 are k-equivalent for all $k \geq 0$, then they are equivalent.

(b) If q_1 and q_2 are $(k+1)$-equivalent then they are k-equivalent.

(c) Two states q_1 and q_2 are k-equivalent if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or both non-final states for all strings x of length k or more.

(d) The k-equivalence is an equivalence relation.

52. The language generated by the grammar $S \rightarrow 0S1|0A1$, $A \rightarrow 1A0|10$ is

(a) $\{1^m 0^n : n, m \geq 1\}$ (b) $\{0^m 1^n : n, m \geq 1\}$

(c) $\{0^n 1^m 0^m 1^n : m, n \geq 1\}$ (d) None of the above

53. The language generated by the grammar $S \rightarrow 0s1|0A1\}$, $A \rightarrow 1A|1$ is

(a) $\{0^m 1^n : n > m > 1\}$ (b) $\{0^m 1^n : n > m \geq 1\}$

(c) $\{0^m 1^n : m > n > 1\}$ (d) None of the above

54. Match the language with the corresponding machine:

Language	*Machine*
(i) Regular language	(A) Non-deterministic pushdown automaton
(ii) DCFL	(B) Turing machine
(iii) CFL	(C) Deterministic pushdown automaton
(iv) Context-sensitive language	(D) (Non)Deterministic finite-state acceptor
(v) Recursive language	(E) Turing machine that halts
(vi) Recursively enumerable language	(F) Linear-bounded automaton

(a) DACFEB (b) DCAFEB (c) DCABEF (d) DCFFAB

55. When Minimizing a $DFA\ M = \langle Q, \Sigma, \delta, q_0, F \rangle$ we try to minimize

(a) The cardinality of the language L such that $L = L(M)$

(b) The cardinality of Q

(c) The out-degree of each vertex.

(d) The length of the strings accepted by M

(e) None of the above.

ANSWER KEY

1.(b)	2.(d)	3.(b)	4.(c)	5.(c)	6.(d)	7.(c)	8.(b)	9.(d)	10.(c)
11.(b)	12.(a)	13.(b)	14.(b)	15.(b)	16.(b)	17.(d)	18.(b)	19.(b)	20.(a)
21.(c)	22.(b)	23.(d)	24.(a)	25.(b)	26.(a)	27.(a)	28.(c)	29.(a)	30.(c)
31.(c)	32.(c)	33.(a)	34.(b)	35.(a)	36.(b)	37.(d)	38.(a)	39.(b)	40.(c)
41.(b)	42.(b)	43.(a)	44.(b)	45.(b)	46.(a)	47.(b)	48.(b)	49.(c)	50.(b)
51.(c)	52.(c)	53.(b)	54.(b)	55.(b)					

Formal Languages And Automata Theory

UNIT 3

COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF AERONAUTICAL ENGINEERING

DUNDIGAL, HYDERABAD - 500 043

5

CONTEXT FREE GRAMMARS

After going through this chapter, you should be able to understand :

- Context free grammars
- Left most and Rightmost derivation of strings
- Derivation Trees
- Ambiguity in CFGs
- Minimization of CFGs
- Normal Forms (CNF & GNF)
- Pumping Lemma for CFLs
- Enumeration properties of CFLs

5.1 CONTEXT FREE GRAMMARS

A grammar $G = (V, T, P, S)$ is said to be a CFG if the productions of G are of the form :

$$A \to \alpha, \text{ where } \alpha \in (V \cup T)^*$$

The right hand side of a CFG is not restricted and it may be null or a combination of variables and terminals. The possible length of right hand sentential form ranges from 0 to ∞ i.e., $0 \le |\alpha| \le \infty$.

As we know that a CFG has no context neither left nor right. This is why, it is known as CONTEXT - FREE. *Many programming languages have recursive structure that can be defined by CFG's.*

Example 1 : Consider the grammar $G = (V, T, P, S)$ having productions :
$S \to aSa \mid bSb \mid \in$. Check the productions and find the language generated.

Solution :

Let $P_1 : S \to aSa$ (RHS is terminal variable terminal)

$P_2 : S \to bSb$ (RHS is terminal variable terminal)

$P_3 : S \to \in$ (RHS is null string)

Since, all productions are of the form $A \to \alpha$, where $\alpha \in (V \cup T)^*$, hence G is a CFG.

Language Generated :

$S \Rightarrow aSa \text{ or } bSb$

$\Rightarrow a^nSa^n \text{ or } b^nSb^n$ (Using n step derivation)

$\Rightarrow a^nb^mSb^ma^n \text{ or } b^na^mSa^mb^n$ (Using m step derivation)

$\Rightarrow a^nb^mb^ma^n \text{ or } b^na^ma^mb^n$ (Using $S \rightarrow \in$)

So, $L(G) = \{ww^R : w \in (a + b)^*\}$

Example 2 : Let G = (V, T, P, S) where V = { S , C }, T = { a , b }
P = { S → aCa
C → aCa | b
} S is the start symbol
What is the language generated by this grammar ?

Solution : Consider the derivation

$S \Rightarrow aCa \Rightarrow$ aba (By applying the 1^{st} and 3^{rd} production)

So, the string aba $\in L(G)$

Consider the derivation

$S \Rightarrow aCa$	By applying	$S \rightarrow aCa$	
$\Rightarrow aaCaa$	By applying	$C \rightarrow aCa$	
$\Rightarrow aaaCaaa$	By applying	$C \rightarrow aCa$	
.....			
.....			
$\Rightarrow a^nCa^n$	By applying	$C \rightarrow aCa$	n times
$\Rightarrow a^nba^n$	By applying	$C \rightarrow b$	

So, the language L accepted by the grammar G is $L(G) = \{ a^n\, ba^n \mid n \geq 1 \}$
i. e., the language L derived from the grammar G is "The string consisting of n number of a's followed by a 'b' followed by n number of a's.

Example 3 : What is the language generated by the grammar

$S \rightarrow 0A \mid \in$

$A \rightarrow 1S$

Solution : The null string $\in$ can be obtained by applying the production $S \rightarrow \in$ and the derivation is shown below :

$S \Rightarrow \in$ (By applying $S \rightarrow \in$)

Consider the derivation

$S \Rightarrow 0A$ (By applying $S \rightarrow 0A$)

$\Rightarrow 01S$ (By applying $A \rightarrow 1S$)

$\Rightarrow 010A$ (By applying $S \rightarrow 0A$)

$\Rightarrow 0101S$ (By applying $A \rightarrow 1S$)

$\Rightarrow 0101$ (By applying $S \rightarrow \in$)

So, alternatively applying the productions $S \rightarrow 0A$ and $A \rightarrow 1S$ and finally applying the production $S \rightarrow \in$, we get string consisting of only of 01's. So, both null string i.e., $\in$ and string consisting 01's can be generated from this grammar. So, the language generated by this grammar is

$$L = \{w \mid w \in \{01\}^*\} \text{ or } L = \{(01)^n \mid n \geq 0\}$$

Example 4 : Show that the language $L = \{a^m b^n \mid m \neq n\}$ is context free.

Solution :

If it is possible to construct a CFG to generate this language then we say that the language is context free. Let us construct the CFG for the language defined. Assume that m = n i. e., m number of a's should be followed by m number of b's. The CFG for this can be

$$S \rightarrow aSb \mid \in \quad \text{.....(1)}$$

But, $L = \{a^m b^n \mid m \neq n\}$ means, a's should be followed by b's and number of a's should not be equal to number of b's i. e., $m \neq n$.

Let us see the different cases when m > n and when m < n.

Case 1 :

m > n : This case occurs if the number of a's are more compared to number of b's. The extra a's can be generated using the production

$$A \rightarrow aA \mid a$$

and the extra a's generated from this production should be appended towards left of the string generated from the production shown in production 1. This can be achieved by introducing one more production.

$$S_1 \rightarrow AS$$

So, even though from S we get n number of a's followed by n number of b's since it is preceded by a variable A from which we could generate extra a's, number of a's followed by number of b's are different.

Case 2 :

m < n : This case occurs if the number of b's are more compared to number of a's. The extra b's can be generated using the production.

$$B \rightarrow bB|b$$

and the extra b's generated from this production should be appended towards right of the string generated from the production shown in production (1). This can be achieved by introducing one more production

$$S_1 \rightarrow SB$$

The context free grammar $G = (V, T, P, S)$ where

$$V = \{S_1, S, A, B\} \ , \ T = \{a, b\}$$

$P = \{$

$$S_1 \rightarrow AS|SB$$
$$S \rightarrow aSb \mid \in$$
$$A \rightarrow aA \mid a$$
$$B \rightarrow bB \mid b$$

$\}$ S_1 is the start symbol

generates the language $L = \{ a^m b^n \mid m \neq n \}$. Since a CFG exists for the language, the language is context free.

Example 5 : Draw a CFG to generate a language consisting of equal number of a's and b's.

Solution : Note that initial production can be of the form

$$S \rightarrow aB \mid bA$$

If the first symbol is **'a'**, the second symbol should be a non - terminal from which we can obtain either **'b'** or one more **'a'** followed by two **B's** denoted by **aBB** or a **'b'** followed by S denoted by **bS** .

Note that from all these symbols definitely we obtain equal number of a's and b's. The productions corresponding to these can be of the form

$$B \rightarrow b|aBB|bS$$

On similar lines we can write A - productions as

$$A \rightarrow a \mid bAA \mid aS$$

from which we obtain a 'b' followed by either

1. 'a' or
2. a 'b' followed by AA's denoted by b AA or
3. symbol 'a' followed by S denoted by aS

The context free grammar G = (V, T, P, S) where

$$V = \{ S, A, B \}, \quad T = \{ a, b \}$$

$$P = \{ \; S \rightarrow aB \mid bA$$

$$A \rightarrow aS \mid bAA \mid a$$

$$B \rightarrow bS \mid aBB \mid b$$

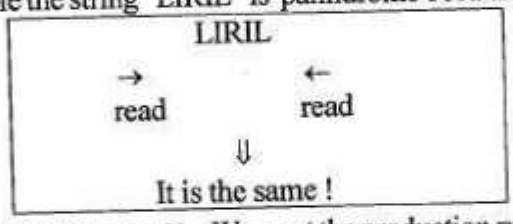
$\}$ S is the start symbol

generates the language consisting of equal number of a's and b's.

Example 6 : Construct CFG for the language L which has all the strings which are all palindromes over $T = \{a, b\}$

Solution : As we know the strings are palindrome if they posses same alphabets from forward as well as from backward.

For example the string "LIRIL" is palindrome because



Since the language L is over $T = \{a, b\}$. We want the production rules to be build a's and b's. As $\in$ can be the palindrome, a can be palindrome even b can be palindrome. So we can write the production rules as

$$G = (\{ S \}, \{a, b\}, P, S)$$

P can be $S \rightarrow a \; S \; a$

$S \rightarrow b \; S \; b$

$S \rightarrow a$

$S \rightarrow b$

$S \rightarrow \in$

The string **abaaba** can be derived as

$$S \rightarrow aSa$$

$$\rightarrow ab\,Sba$$

$$\rightarrow ab\,a\,Sa\,ba$$

$$\rightarrow ab\,a \in a\,ba$$

$$\rightarrow ab\,a\;a\,ba$$

which is a palindrome.

Example 7 : Obtain a CFG to generate integers .

Solution :

The sign of a number can be '+' or '-' or $\in$. The production for this can be written as

$$S \rightarrow +|-|\in$$

A number can be formed from any of the digits 0, 1, 2, ...9. The production to obtain these digits can be written as $D \rightarrow 0|1|2|...|9$

A number N can be recursively defined as follows .

1. A number N is a digit D (i. e., $N \rightarrow D$)
2. The number N followed by digit D is also a number (i. e., $N \rightarrow ND$)

The productions for this recursive definition can be written as

$$N \rightarrow D$$
$$N \rightarrow ND$$

An integer number I can be a number N or the sign S of a number followed by number N. The production for this can be written as $I \rightarrow N|SN$

So, the grammar G to obtain integer numbers can be written as $G = (V, T, P, S)$ where

V = { D, S, N, I } , T = { +, -, 0, 1, 2, ... 9 }

p = {

I → N | SN

N → D | ND

S → + | - | $\in$

D → 0 | 1 | 2 | ..... | 9

}

S = I which is the start symbol

Example 8 : Obtain the grammar to generate the language

$$L = \{ 0^m 1^m 2^n | m \geq 1 \text{ and } n \geq 0 \}.$$

Solution : In the language $L = \{ 0^m 1^m 2^n \}$, if n = 0, the language L contains m number of 0's and m number of 1's. The grammar for this can be of the form

$$A \rightarrow 01 | 0A1$$

If n is greater than zero, the language L should contain m number of 0's followed by m number of 1's followed by one or more 2's i. e., the language generated from the non - terminal A should be followed by n number of 2's. So, the resulting productions can be written as

$$S \rightarrow A | S2$$
$$A \rightarrow 01 | 0A1$$

Thus, the grammar G to generate the language

$$L = \{ 0^m 1^n 2^n \mid m \geq 1 \text{ and } n \geq 0 \}$$

can be written as G = (V, T, P, S) where

V = { S, A } , T = { 0, 1, 2 }

P = {

$S \rightarrow A \mid S2$

$A \rightarrow 01 \mid 0A1$

} S is the start symbol

Example 9 : Obtain a grammar to generate the language $L = \{0^n 1^{n+1} \mid n \geq 0\}$.

Solution :

Note : It is clear from the language that total number of 1's will be one more than the total number of 0's and all 0's precede all 1's. So, first let us generate the string $0^n 1^n$ and add the digit 1 at the end of this string.

The recursive definition to generate the string $0^n 1^n$ can be written as

$$A \rightarrow 0A1 \mid \in$$

If the production $A \rightarrow 0A1$ is applied n times we get the sentential form as shown below.

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow \quad \ldots\ldots 0^n A 1^n$$

Finally if we apply the production

$$A \rightarrow \in$$

the derivation starting from the start symbol A will be of the form

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 0^n A 1^n \Rightarrow 0^n 1^n$$

Thus, using these productions we get the string $0^n 1^n$. But, we should get the string $0^n 1^{n+1}$ i. e., an extra 1 should be placed at the end. This can be achieved by using the production

$$S \rightarrow A1$$

Note that from A we get string $0^n 1^n$ and 1 is appended at the end resulting in the string $0^n 1^{n+1}$.

So, the final grammar G to generate the language $L = \{ 0^n 1^{n+1} \mid n \geq 0 \}$ will be G = (V, T, P, S) where

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

P = {

S → A 1

A → 0 A 1 | ∈

} S is the start symbol

Example 10 : Obtain the grammar to generate the language

$$L = \{w \mid n_a(w) = n_b(w)\}$$

Solution :

Note : $n_a(w) = n_b(w)$ means, number of a's in the string w should be equal to number of b's in the string w. To get equal number of a's and b's, we know that there are three cases :

1. There are no a's and b's present in the string w.
2. The symbol 'a' can be followed by the symbol 'b'
3. The symbol 'b' can be followed by the symbol a'

The corresponding productions for these three cases can be written as

$$S \rightarrow \in$$

$$S \rightarrow aSb$$

$$S \rightarrow bSa$$

Using these productions the strings of the form $\in$, ab, ba, abab, baba etc., can be generated. But, the stirngs such as abba, baab, etc., where the string starts and ends with the same symbol, can not be generated from these productions (even though they are valid strings).

So, to obtain in the producitons to generate such strings, let us divide the string into two substrings. For example, let us take the string 'abba'. This string can be split into two substrings 'ab' and 'ba'. The substring 'ab' can be generated from S and the derivation is shown below :

$$S \Rightarrow aSb \qquad (\text{By applying } S \rightarrow aSb)$$

$$\Rightarrow ab \qquad (\text{By applying } S \rightarrow \in)$$

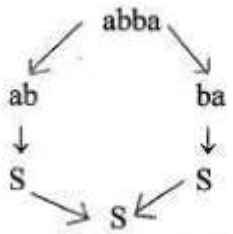
Similarly, the substring 'ba' can be generated from S and the derivation is shown below :

$$S \Rightarrow bSa \qquad (\text{By applying } S \rightarrow bSa)$$

$$\Rightarrow ba \qquad (\text{By applying } S \rightarrow \in)$$

i. e., the first sub string 'ab' can be generated from S as shown in the first derivation and the second sub string 'ba' can also be generated from S as shown in second derivation.

So, to get the string 'abba' from S, perform the derivation in reverse order as shown below :



So, to get a string such that it starts and ends with the same symbol, the production to be used is

$$S \rightarrow SS$$

So, the final grammar to generate the language $L = \{ w \mid n_a(w) = n_b(w) \}$ is G = (V, T, P, S) where

$$V = \{ S \} \quad , \quad T = \{ a, b \}$$

$$P = \{ \; S \to \in$$

$$S \to aSb$$

$$S \to bSa$$

$$S \to SS$$

$$\} \quad S \text{ is the start symbol}$$

5.2 LEFTMOST AND RIGHTMOST DERIVATIONS

Leftmost derivation :

If $G = (V, T, P, S)$ is a CFG and $w \in L(G)$ then a derivation $S \underset{L}{\overset{*}{\Rightarrow}} w$ is called leftmost derivation if and only if all steps involved in derivation have leftmost variable replacement only.

Rightmost derivation :

If $G = (V, T, P, S)$ is a CFG and $w \in L(G)$, then a derivation $S \underset{R}{\overset{*}{\Rightarrow}} w$ is called rightmost derivation if and only if all steps involved in derivation have rightmost variable replacement only.

Example 1 : Consider the grammar $S \to S + S \mid S * S \mid a \mid b$. Find leftmost and rightmost derivations for string $w = a * a + b$.

Solution :

Leftmost derivation for $w = a * a + b$

$S \underset{L}{\Rightarrow} S * S$ (Using $S \to S * S$)

$\underset{L}{\Rightarrow} a * S$ (The first left hand symbol is a, so using $S \to a$)

$\underset{L}{\Rightarrow} a * S + S$ (Using $S \to S + S$, in order to get $a + b$)

$\underset{L}{\Rightarrow} a * a + S$ (Second symbol from the left is a, so using $S \to a$)

$\underset{L}{\Rightarrow} a * a + b$ (The last symbol from the left is b, so using $S \to b$)

Rightmost derivation for $w = a * a + b$

$S \underset{R}{\Rightarrow} S * S$ (Using $S \rightarrow S * S$)

$\underset{R}{\Rightarrow} S * S + S$ (Since, in the above sentential form second symbol from the right is * so, we can not use $S \rightarrow a|b$. Therefore, we use $S \rightarrow S + S$)

$\underset{R}{\Rightarrow} S * S + b$ (Using $S \rightarrow b$)

$\underset{R}{\Rightarrow} S * a + b$ (Using $S \rightarrow a$)

$\underset{R}{\Rightarrow} a * a + b$ (Using $S \rightarrow a$)

Example 2 : Consider a CFG $S \rightarrow bA|aB$, $A \rightarrow aS|aAA|a$, $B \rightarrow bS|aBB|b$. Find leftmost and rightmost derivations for $w = aaabbabbba$.

Solution :

Leftmost derivation for $w = aaabbabbba$:

$S \Rightarrow aB$ (Using $S \rightarrow aB$ to generate first symbol of w)

$\Rightarrow aaBB$ (Since, second symbol is a, so we use $B \rightarrow aBB$)

$\Rightarrow aaaBBB$ (Since, third symbol is a, so we use $B \rightarrow aBB$)

$\Rightarrow aaabBB$ (Since fourth symbol is b, so we use $B \rightarrow b$)

$\Rightarrow aaabbB$ (Since, fifth symbol is b, so we use $B \rightarrow b$)

$\Rightarrow aaabbaBB$ (Since, sixth symbol is a, so we use $B \rightarrow aBB$)

$\Rightarrow aaabbabB$ (Since, seventh symbol is b, so we use $B \rightarrow b$)

$\Rightarrow aaabbabbS$ (Since, eighth symbol is b, so we use $B \rightarrow bS$)

$\Rightarrow aaabbabbbA$ (Since, ninth symbol is b, so we use $S \rightarrow bA$)

$\Rightarrow aaabbabbba$ (Since, the tenth symbol is a, so using $A \rightarrow a$)

Rightmost derivation for $w = aaabbabbba$

$S \Rightarrow aB$ (Using $S \rightarrow aB$ to generate first symbol of w)

$\Rightarrow aaBB$ (We need a as the rightmost symbol and second symbol from the left side, so we use $B \rightarrow aBB$)

$\Rightarrow aaBbS$ (We need a as rightmost symbol and this is obtained from A only, we use $B \rightarrow bS$)

$\Rightarrow aaBbbA$ (Using $S \rightarrow bA$)

$\Rightarrow aaBbba$ (Using $A \rightarrow a$)

$\Rightarrow aaaBBbba$ (We need b as the fourth symbol from the right)

$\Rightarrow aaaBbbba$ (Using $B \rightarrow b$)

$\Rightarrow aaabSbbba$ (Using $B \rightarrow bS$)

$\Rightarrow$ *aaabbAbbba* (Using $S \rightarrow bA$)

$\Rightarrow$ *aaabbabbba* (Using $A \rightarrow a$)

5.3 DERIVATION TREES

Let $G = (V, T, P, S)$ is a CFG. Each production of G is represented with a tree satisfying the following conditions:

1. If $A \rightarrow \alpha_1\alpha_2\alpha_3 \ldots \alpha_n$ is a production in G, then A becomes the parent of nodes labeled $\alpha_1, \alpha_2, \alpha_3, \ldots \alpha_n$, and
2. The collection of children from left to right yields $\alpha_1\alpha_2\alpha_3 \ldots \alpha_n$

Example : Consider a CFG $S \rightarrow S + S \mid S * S \mid a \mid b$ and construct the derivation trees for all productions.

Solution :

For the production $S \rightarrow S + S$

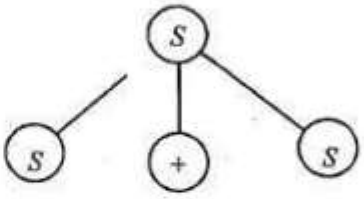


Figure (a)

For the production $S \rightarrow S * S$

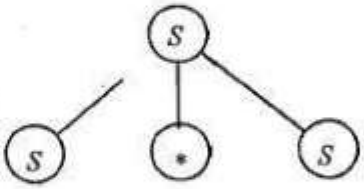


Figure (b)

For the production $S \rightarrow a$

Figure (c)

For the production $S \rightarrow b$

Figure (d)

If $w \in L(G)$ then it is represented by a tree called **derivation tree or parse tree** satisfying the following conditions :

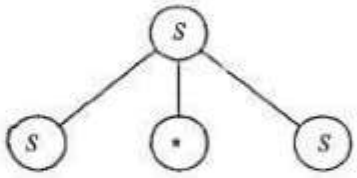
1. The root has label S (the starting symbol),
2. The all internal vertices (or nodes) are labeled with variables,
3. The leaves or terminal nodes are labeled with $\in$ or terminal symbols,
4. If $A \rightarrow \alpha_1\alpha_2\alpha_3 \ldots \alpha_n$ is a production in G, then A becomes the parent of nodes labeled $\alpha_1, \alpha_2, \alpha_3, \ldots \alpha_n$, and
5. The collection of leaves from left to right yields the string w.

Example 1 : Consider the grammar $S \rightarrow S + S | S * S | a | b$. Construct derivation tree for string $w = a * b + a$.

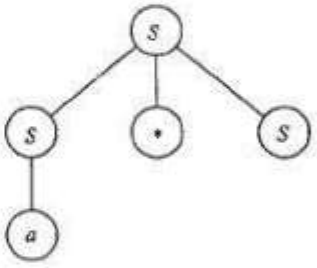
Solution : The derivation tree or parse tree is shown in below figure.

Leftmost derivation for $w = a * b + a$:

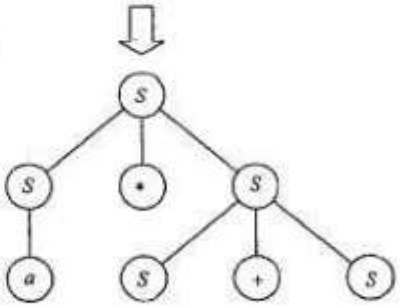
$S \Rightarrow S * S$ (Using $S \rightarrow S * S$)



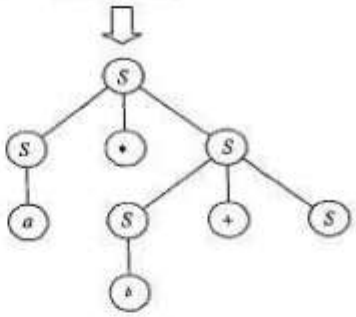
$\Rightarrow a * S$ (The first left hand symbol is a, so using $S \rightarrow a$)



$\Rightarrow a * S + S$ (Using $S \rightarrow S + S$, in order to get $b + a$)



$\Rightarrow a * b + S$ (Second symbol from the left is b, so using $S \rightarrow b$)



$\Rightarrow a * b + a$ (The last symbol from the left is a, so using $S \rightarrow a$)

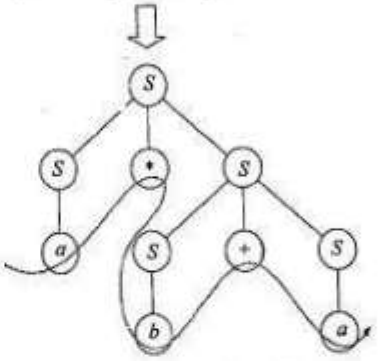


Figure : Parse tree for $a * b + a$

Example 2 : Consider a grammar G having productions $S \rightarrow aAS|a$, $A \rightarrow SbA|SS|ba$.

Show that $S \overset{*}{\Rightarrow} aabbaa$ and construct a derivation tree whose yield is aabbaa.

Solution :

$S \Rightarrow aAS$
$\Rightarrow aSbAS$
$\Rightarrow$ aabAS
$\Rightarrow aabbaS$
$\Rightarrow aabbaa$

Hence, $S \overset{*}{\Rightarrow} aabbaa$
Parse tree is shown in figure.

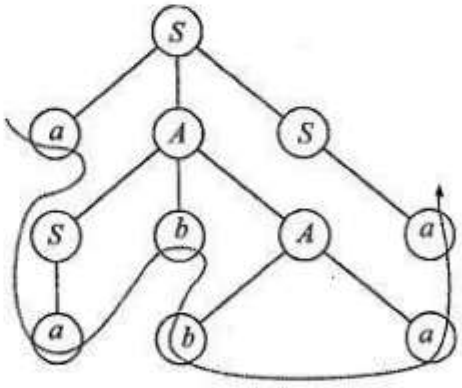


Figure : Parse tree yielding $aabbaa$

Example 3 : Consider the grammar G whose productions are

$S \rightarrow 0B|1A,\ A \rightarrow 0|0S|1AA,\ B \rightarrow 1|1S|0BB$. Find

(a) Leftmost and(b) Rightmost derivation for string 00110101, and construct derivation tree also.

Solution :

(a) Leftmost derivation :

$S \Rightarrow 0B \Rightarrow 00BB$

$\Rightarrow 001B \Rightarrow 0011S$

$\Rightarrow 00110B \Rightarrow 001101S$

$\Rightarrow 0011010B \Rightarrow 00110101$

(b) Rightmost derivation :

$S \Rightarrow 0B \Rightarrow 00BB$

$\Rightarrow 00B1 \Rightarrow 001S1$

$\Rightarrow 0011A1 \Rightarrow 00110S1$

$\Rightarrow 001101A1 \Rightarrow 00110101$

(c) Derivation tree :

Derivation tree is shown in below figure.

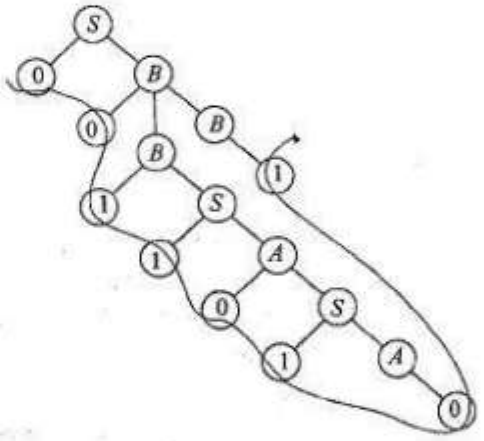


Figure : Derivation tree for 00110101

5.4 AMBIGUITY IN CFGs

A grammar G is *ambiguous if there exists some string* $w \in L(G)$ *for which there are two or more distinct derivation trees, or there are two or more distinct leftmost derivations.*

Example 1 : Consider the CFG $S \rightarrow S + S \mid S * S \mid a \mid b$ and string $w = a * a + b$, and derivations as follows:

Solution :

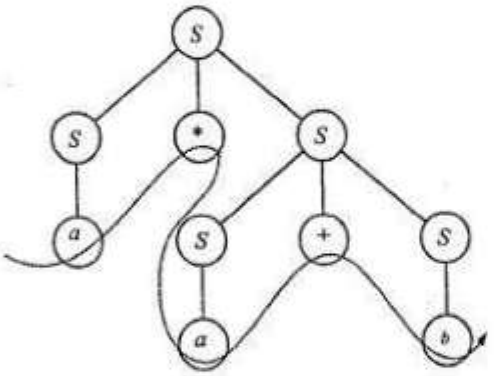
First leftmost derivation for $w = a * a + b$

$S \Rightarrow S * S$ (Using $S \rightarrow S * S$)
$\Rightarrow a * S$ (Using $S \rightarrow a$)
$\Rightarrow a * S + S$ (Using $S \rightarrow S + S$)
$\Rightarrow a * a + S$ (Using $S \rightarrow a$)
$\Rightarrow a * a + b$ (Using $S \rightarrow b$)

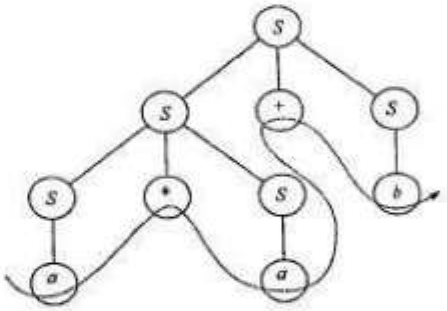
Second leftmost derivation for $w = a * a + b$

$S \Rightarrow S + S$ (Using $S \rightarrow S + S$)
$\Rightarrow S * S + S$ (Using $S \rightarrow S * S$)
$\Rightarrow a * S + S$ (Using $S \rightarrow a$)
$\Rightarrow a * a + S$ (Using $S \rightarrow a$)
$\Rightarrow a * a + b$ (Using $S \rightarrow b$)

Two distinct parse trees are shown in figure (a) and figure (b)



Figure(a) Parse tree for $a * a + b$



Figure(b) Parse tree for $a * a + b$

Since, there are two distinct leftmost derivations (two parse trees) for string w, hence w is ambiguous and there is ambiguity in grammar G.

Example 2 : Show that the following grammars are ambiguous.

(a) $S \rightarrow SS \mid a \mid b$

(b) $S \rightarrow A \mid B \mid b,\ A \rightarrow aAB \mid ab,\ B \rightarrow abB \mid \in$

Solution :

(a) Consider the string $w = bbb$, two leftmost derivations are as follows :

$S \underset{L}{\Rightarrow} SS$ (Using $S \to SS$)

$\underset{L}{\Rightarrow} bS$ (Using $S \to b$)

$\underset{L}{\Rightarrow} bSS$ (Using $S \to SS$)

$\underset{L}{\Rightarrow} bbS$ (Using $S \to b$)

$\underset{L}{\Rightarrow} bbb$ (Using $S \to b$)

$S \underset{L}{\Rightarrow} SS$ (Using $S \to SS$)

$\underset{L}{\Rightarrow} SSS$ (Using $S \to SS$)

$\underset{L}{\Rightarrow} bSS$ (Using $S \to b$)

$\underset{L}{\Rightarrow} bbS$ (Using $S \to b$)

$\underset{L}{\Rightarrow} bbb$ (Using $S \to b$)

Two parse trees are shown in figure(a) and figure(b).

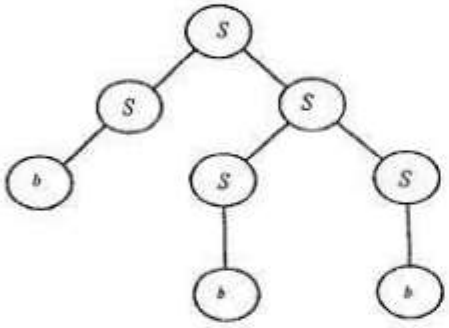


Figure (a) Parse tree for bbb

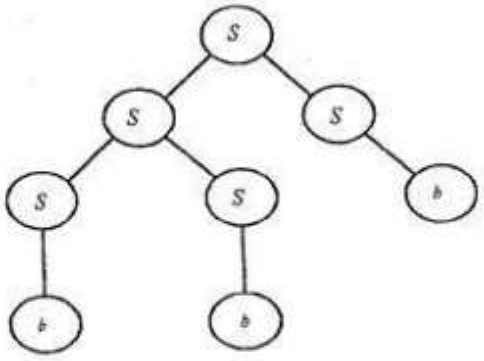


Figure (b) Parse tree for bbb

So, the given grammar is ambiguous.

(b) Consider the string $w = ab$, we get two leftmost derivations for w as follows :

$S \underset{L}{\Rightarrow} A$

$\underset{L}{\Rightarrow} ab$ (Using $A \to ab$)

$S \underset{L}{\Rightarrow} B$

$\underset{L}{\Rightarrow} abB$ (Using $B \to abB$)

$\underset{L}{\Rightarrow} ab$ (Using $B \to \in$)

Two parse trees are shown in figure (c) and figure (d).

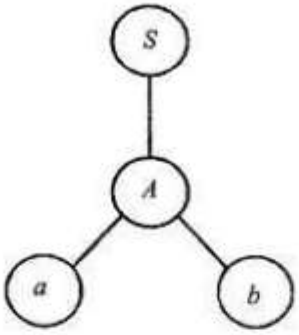


Figure (c) Parse tree for $w = ab$

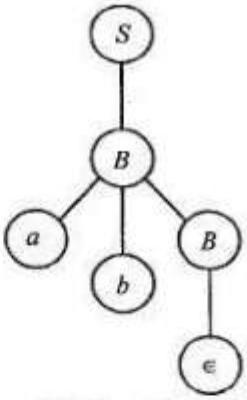


Figure (d) Parse tree for $w = ab$

So, the given grammar is ambiguous.

5.4.1 Removal of Ambiguity

5.4.1.1 Left Recursion

A grammar can be changed from one form to another accepting the same language. If a grammar has left recursive property, it is undesirable and left recursion should be eliminated. The left recursion is defined as follows.

Definition : A grammar G is said to be left recursive if there is some non terminal A such that $A \Rightarrow^{+} A\alpha$. In other words, in the derivation process starting from any non - terminal A, if a sentential form starts with the same non - terminal A, then we say that the grammar is having left recursion.

Elimination of Left Recursion

The left recursion in a grammar G can be eliminated as shown below. Consider the A - production of the form $A \rightarrow A\alpha_1 | A\alpha_2 | A\alpha_3 ..... A\alpha_n | \beta_1 | \beta_2 | \beta_3 ..... \beta_m$

where β_i's do not start with A. Then the A productions can be replaced by

$$A \rightarrow \beta_1 A^1 | \beta_2 A^1 | \beta_3 A^1 | ..... \beta_m A^1$$

$$A^1 \rightarrow \alpha_1 A^1 | \alpha_2 A^1 | \alpha_3 A^1 | ..... | \alpha_n A^1 | \in$$

Note that α_i's do not start with A^1.

Example 1 : Eliminate left recursion from the following grammar

$$E \rightarrow E + T | T$$

$$T \rightarrow T * F | F$$

$$F \rightarrow (E) | id$$

Solution : The left recursion can be eliminated as shown below :

Given	Substitution	Without left recursion
$A \to A\alpha_i \mid \beta_i$		$A \to \beta_i A^1$ and $A^1 \to \alpha_i A^1 \mid \in$
$E \to E+T \mid T$	$A = E$ $\alpha_1 = +T$ $\beta_1 = T$	$E \to TE^1$ $E^1 \to +TE^1 \mid \in$
$T \to T*F \mid F$	$A = T$ $\alpha_1 = *F$ $\beta_1 = F$	$T \to FT^1$ $T^1 \to *FT^1 \mid \in$
$F \to (E) \mid id$	Not applicable	$F \to (E) \mid id$

The grammar obtained after eliminating left recursion is

$$E \to TE^1$$
$$E^1 \to +TE^1 \mid \in$$
$$T \to FT^1$$
$$T^1 \to *FT^1 \mid \in$$
$$F \to (E) \mid id$$

Example 2 : Eliminate left recursion from the following grammar

$$S \to Ab \mid a$$
$$A \to Ab \mid Sa$$

Solution :

The non terminal S, even though is not having immediate left recursion, it has left recursion because $S \Rightarrow Ab \Rightarrow Sab$ i. e., $S \Rightarrow^+ Sab$. Substituting for S in the A - production can eliminate the indirect left recursion from S. So, the given grammar can be written as

$$S \to Ab \mid a$$
$$A \to Ab \mid Aba \mid aa$$

Now, A - production has left recursion and can be eliminated as shown below :

Given	Substitution	Without left recursion
$A \rightarrow A\alpha_i \mid \beta_i$		$A \rightarrow \beta_i A^1$ and $A^1 \rightarrow \alpha_i A^1 \mid \in$
$S \rightarrow Ab \mid a$	Not applicable	$S \rightarrow Ab \mid a$
$A \rightarrow Ab \mid Aba \mid aa$	A = A $\alpha_1 = b$ $\alpha_2 = ba$ $\beta_1 = aa$	$A \rightarrow aaA^1$ $A^1 \rightarrow bA^1 \mid baA^1 \mid \in$

The grammar obtained after eliminating left recursion is

$$S \rightarrow Ab \mid a$$

$$A \rightarrow aaA^1$$

$$A^1 \rightarrow bA^1 \mid baA^1 \mid \in$$

5.4.1.2 Left Factoring

Definition :

Two or more productions of a variable A of the grammar $G = (V, T, P, S)$ are said to have left factoring if A - productions are of the form $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n$, where $\beta_i \in (V \cup T)^*$ and does not start (prefix) with α . All these A - productions have common left factor α .

Elimination of Left Factoring

Let the variable A has (left factoring) productions as follows :

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_m$, where $\beta_1, \beta_2, \beta_3 \ldots \beta_n$ and $\gamma_1, \gamma_2, \ldots \gamma_m$ do not contain α as a prefix, then we replace A - productions by :

$A \rightarrow \alpha A' \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_m$, where $A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$.

Example : Consider the grammar $S \rightarrow aSa \mid aa$ and remove the left factoring (if any).

Solution :

$S \rightarrow aSa$ and $S \rightarrow aa$ have $a = a$ as a left factor, so removing the left factoring, we get the productions : $S \rightarrow aS'$, $S' \rightarrow Sa \mid a$.

The problem associated with left factoring and left recursive grammars is back - tracking. We can find α as a prefix in RHS in many ways and a string having α as a prefix can create problem. In worst condition, to get appropriate remaining part of the string we have to search the entire production list. We take the first production, if it is not suitable then take second production and so on. This situation is known as back - tracking . For example, consider the above S - productions $S \rightarrow aSa \mid aa$ and a string w = aa. We have choice of the both productions looking at the first symbol on the RHS.

Iteration First :

$S \Rightarrow aSa$

$\Rightarrow aaaa \neq w$

Iteration Second :

$S \Rightarrow aa = w$

So, if we follow the iteration first, then we can not get the string w and we will have to return to the iteration second i. e. the starting symbol. The problem, in which we proceed further and do not get the desired string and we come to the previous step, is known as back - tracking. This problem is a fundamental problem in designing of compilers (parser).

Procedure for Removal of Ambiguity :

We have no obvious rule or method defined for removing ambiguity as we have for left recursion and left factoring. So, we will have to concentrate on heuristic approach most of the time.

Let us consider the ambiguous grammar $S \rightarrow S+S \mid S*S \mid a \mid b$. Now, if we analyze the productions, then we find that two productions are left recursive. So, first we try to remove the left recursion.

$S \rightarrow S+S$ and $S \rightarrow S*S$ is replaced by $S \rightarrow aS' \mid bS'$, $S' \rightarrow +SS' \mid *SS' \mid \in$

Now, we check the derivation for ambiguous string $w = a*a+a$. We have only one left most derivation or only one parse tree given as follows :

$$
\begin{aligned}
S &\Rightarrow aS' \\
&\Rightarrow a*SS' \\
&\Rightarrow a*aS'S' \\
&\Rightarrow a*a+SS'S' \\
&\Rightarrow a*a+aS'S'S' \\
&\Rightarrow a*a+a \in S'S' \\
&\Rightarrow a*a+a \in S' \\
&\Rightarrow a*a+a \in (\equiv a*a+a)
\end{aligned}
$$

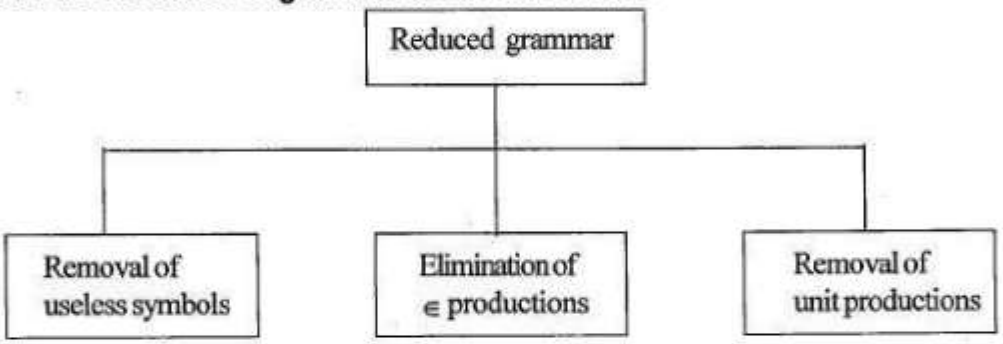
So, we conclude that removal of left recursion (and left factoring also) helps in removal of ambiguity of the ambiguous grammars.

5.5 MINIMIZATION OF CFGs

As we have seen various languages can effectively be represented by context free grammar. All the grammars are not always optimized. That means grammar may consists of some extra symbols (non - terminals). Having extra symbols unnecessary increases the length of grammar. Simplification of grammar means reduction of grammar by removing useless symbols. The properties of reduced grammar are given below :

1. Each variable (i. e. non - terminal) and each terminal of G appears in the derivation of some word in L.
2. There should not be any production as $X \to Y$ where X and Y are non - terminals.
3. If $\in$ is not in the language L then there need not be the production $X \to \in$.

We see the reduction of grammar as shown below :



5.5.1 Removal of useless symbols

Definition : A symbol X is useful if there is a derivation of the form

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* w$$

Otherwise, the symbol X is useless. Note that in a derivation, finally we should get string of terminals and all these symbols must be reachable from the start symbol S. Those symbols and productions which are not at all used in the derivation are useless.

Theorem 5.5.1 : Let G = (V, T, P, S) be a CFG. We can find an equivalent grammar $G_1 = (V_1, T_1, P_1, S)$ such that for each A in $(V_1 \cup T_1)$ there exists α and β in $(V_1 \cup T_1)^*$ and x in T^* for which $S \Rightarrow^* \alpha A \beta \Rightarrow^* x$.

Proof : The grammar G_1 can be obtained from G in two stages.

STAGE 1 :

Obtain the set of variables and productions which derive only string of terminals i. e., Obtain a grammar $G_1 = (V_1, T_1, P_1, S)$ such that V_1 contains only the set of variables A for which $A \Rightarrow^* x$ where $x \in T^+$.

The algorithm to obtain a set of variables from which only string of terminals can be derived is shown below.

Step 1 : [Initialize old _ variables denoted by ov to ϕ]

$$ov = \phi$$

Step 2 : Take all productions of the form $A \rightarrow x$ where $x \in T^+$ i. e., if the R. H. S of the production contains only string of terminals consider those productions and corresponding non terminals on L. H. S are added to new _ variables denoted by nv. This can be expressed using the following statement :

$$nv = \{ A \mid A \rightarrow x \quad and \quad x \in T^+ \}$$

Step 3 : Compare ov and nv. As long as the elements in ov and nv are not equal, repeat the following statements. Otherwise goto step 4.

a. [Copy new_varialbes to old _variables]
ov = nv

b. Add all the elements in ov to nv. Also add the variables which derive a string consisting of terminals and non terminals which are in ov.

$$nv = ov \cup \{ A \mid A \rightarrow y \; and \; y \in (ov \cup T)^* \}$$

Step 4 : When the loop is terminated, nv (or ov) contains all those non terminals from which only the string of terminals are derived and add those variables to V_1.

i. e., $V_1 = ov$

Step 5 : [Terminate the algorithm]

return V_1

Note that the variable V_1 contains only those variables from which string of terminals are obtained. The productions used to obtain V_1 are added to P_1 and the terminals in these productions are added to T_1. The grammar $G_1 = (V_1, T_1, P_1, S)$ contains those variables A in V_1 such that $A \Rightarrow^* x$ for some x in T^+. Since each derivation in G_1 is a derivation of G, $L(G_1) = L(G)$.

STAGE 2 :

Obtain the set of variables and terminals which are reachable from the start symbol and the corresponding productions. This can be obtained as shown below :

Given a CFG $G = (V, T, P, S)$, we can find an equivalent grammar $G_1 = (V_1, T_1, P_1, S)$ such that for each X in $V_1 \cup T_1$ there exists α such that $S \Rightarrow^* \alpha$ and X is a symbol in α i. e., if X is a variable $X \in V_1$ and if X is terminal $X \in T_1$. Each symbol X in $V_1 \cup T_1$ is reachable from the start symbol S. The algorithm for this is shown below.

$V_1 = \{S\}$

For each A in V_1

if $A \rightarrow \alpha$ then

Add the variables in A to V_1

Add the terminals in α to T_1

Endif

Endfor

Using this algorithm all those symbols (whether variables or terminals) that are not reachable from the start symbol are eliminated. The grammar G_1 does not contain any useless symbol or production. For each $X \in L(G_1)$ there is a derivation.

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* x$$

Using these two steps we can effectively find G_1 such that $L(G) = L(G_1)$ and the two grammars G and G_1 are equivalent.

Example 1 : Eliminate the useless symbols in the grammar

S → aA| bB
A → aA | a
B → bB
D → ab | Ea
E → aC | d

Solution :

Stage 1 : Applying the algorithm shown in stage 1 of the theorem 5.5.1, we can obtain a set of variables from which we get only string of terminals and is shown below.

oV	nV	Productions
ϕ	A, D, E	A → a D → ab E → d
A, D, E	A, D, E, S	S → aA A → aA D → Ea
A, D, E, S	A, D, E, S	

The resulting grammar $G_1 = (V_1, T_1, P_1, S)$ where

V_1 = {A, D, E, S}

T_1 = {a, b, d}

P_1 = {

A → a | aA
D → ab | Ea
E → d
S → aA

} S is the start symbol

contains all those variables in V_1 such that $A \Rightarrow^+ w$ where $w \in T^+$.

Stage 2 :

Applying the algorithm given in stage 2 of the theorem 5.5.1, we obtain the symbols such that each symbol X is reachable from the start symbol S as shown below.

P_1	T_1	V_1
-	-	S
$S \to aA$	a	S, A
$A \to a \mid aA$	a	S, A

The resulting grammar $G_1 = (V_1, T_1, P_1, S)$ where $V_1 = \{S, A\}$, $T_1 = \{a\}$

P_1 = {

$S \to aA$
$A \to a \mid aA$

} S is the start symbol

such that each symbol X in $(V_1 \cup T_1)$ has a derivation of the form $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$.

Example 2 : Eliminate the useless symbols in the grammar

$$
\begin{array}{lcl}
S & \rightarrow & aA \mid a \mid Bb \mid cC \\
A & \rightarrow & aB \\
B & \rightarrow & a \mid Aa \\
C & \rightarrow & cCD \\
D & \rightarrow & ddd
\end{array}
$$

Solution :

Stage 1 :

Applying the algorithm shown in stage1 of theorem 5.5.1 , we can obtain a set of variables from which we get only string of terminals and is shown below.

ov	nv	Productions
ϕ	S, B, D	S → a B → a D → ddd
S, B, D	S, B, D, A	S → Bb A → aB
S, B, D, A	S, B, D, A	S → aA B → Aa

The resulting grammar $G_1 = (V_1, T_1, P_1, S)$ where

$V_1 = \{ S, B, D, A \}$

$T_1 = \{ a, b, d \}$

$P_1 = \{$

$$
\begin{array}{lcl}
S & \rightarrow & a \mid Bb \mid aA \\
B. & \rightarrow & a \mid Aa \\
D & \rightarrow & ddd \\
A & \rightarrow & aB
\end{array}
$$

$\}$

S is the start symbol contains all those variables in V_1 such that $A \Rightarrow^* w$.

Stage 2 :

Applying the algorithm given in stage 2 of the theorem 5.5.1, we obtain the symbols such that each symbol X is reachable from the start symbol S as shown below.

P_1	T_1	V_1
-	-	S
S → a | Bb | Aa	a, b	S, A, B
A → a B	a, b	S, A, B
B → a | A a	a, b	S, A, B

The resulting grammar $G_1 = (V_1, T_1, P_1, S)$ where

V_1 = { S, A, B }

T_1 = { a, b }

P_1 = {

S → a | Bb | aA

A → aB

B → a | Aa

} S is the start symbol

such that each symbol X in $(V_1 \cup T_1)$ has a derivation of the form $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$.

5.5.2 Eliminating ϵ - productions

A production of the form $A \rightarrow \epsilon$ is undesirable in a CFG, unless an empty string is derived from the start symbol. Suppose, the language generated from a grammar G does not derive any empty string and the grammar consists of ϵ- productions. Such ϵ - productions can be removed. An ϵ - production is defined as follows :

Definition 1 : Let G = (V, T, P, S) be a CFG. A production in P of the form

$$A \rightarrow \epsilon$$

is called an ϵ - production or NULL production. After applying the production the variable A is erased. For each A in V, if there is a derivation of the form

$$A \Rightarrow^* \epsilon$$

then A is a nullable variable.

Example : Consider the grammar

S → ABCa | bD

A → BC | b

B → b | ϵ

$$C \rightarrow c \mid \in$$
$$D \rightarrow d$$

In this grammar, the productions

$$B \rightarrow \in$$
$$C \rightarrow \in$$

are $\in$ - productions and the variables B, C are nullable variables. Because there is a production

$$A \rightarrow BC$$

and both B and C are nullable variables, then A is also a nullable variable.

Definition 2 : Let G = (V, T, P, S) be a CFG where V is set of variables, T is set of terminals, P is set of productions and S is the start symbol. A nullable variable is defined as follows.

1. If $A \rightarrow \in$ is a production in P, then A is a nullable variable.
2. If $A \rightarrow B_1 B_2 \ldots\ldots B_n$ is a production in P, and if $B_1 B_2 \ldots\ldots B_n$ are nullable variables, then A is also a nullable variable
3. The variables for which there are productions of the form shown in step 1 and step 2 are nullable variables.

Even though a grammar G has some $\in$ - productions, the language may not derive a language containing empty string. So, in such cases, the $\in$ - productions or NULL productions are not needed and they can be eliminated.

Theorem 5.5.2 : Let G = (V, T, P, S) where $L(G) \neq \in$. We can effectively find an equivalent grammar G_1 with no $\in$– productions such that $L(G_1) = L(G) - \in$.

Proof : The grammar G_1 can be obtained from G in two steps.

Step 1 : Find the set of nullable variables in the grammar G using the following algorithm.

$ov = \phi$
$nv = \{A \mid A \rightarrow \in\}$
while (ov ! = n v)
{
    ov = nv
    nv = ov $\cup$ { $A \mid A \rightarrow \alpha$ *and* $\alpha \in ov^*$ }
}
V = ov

Once the control comes out of the while loop, the set V contains only the nullable variables.

Step 2 : Construction of productions P_1. Consider a production of the form

$$A \to X_1 X_2 X_3 \ldots\ldots\ldots X_n,\ n \geq 1$$

where each X_i is in $(V \cup T)$. In a production, take all possible combinations of nullable variables and replace the nullable variables with $\in$ one by one and add the resulting productions to P_1. If the given production is not an $\in$ - production, add it to P_1.

Suppose, A and B are nullable variables in the production, then

1. First add the production to P_1.
2. Replace A with $\in$ and add the resulting production to P_1
3. Replace B with $\in$ and the resulting production to P_1.
4. Replace A and B with $\in$ and add the resulting production to P_1.
5. If all symbols on right side of production are nullable variables, the resulting production is an $\in$ production and do not add this to P_1.

Thus, the resulting grammar G_1 obtained, generates the same language as generated by G without $\in$ and the proof is straight forward.

Example 1 : Eliminate all $\in$ - productions from the grammar

S	→	ABCa | bD
A	→	BC | b
B	→	b | $\in$
C	→	c | $\in$
D	→	d

Solution :

Step 1 :

Obtain the set of nullable variables from the grammar. This can be done using step 1 of theorem 5.5.2 as shown below.

ov	IIV	Productions
ϕ	B, C	B → $\in$ C → $\in$
B, C	B, C, A	A → BC
B, C, A	B, C, A	-

V = { B, C, A } are all nullable variables.

Step 2 : Construction of productions P_1.

Productions	Resulting productions (P_1)
$S \rightarrow ABCa$	$S \rightarrow ABCa$ | BCa |ACa | ABa | Ca | A a | Ba | a
$S \rightarrow bD$	$S \rightarrow bD$
$A \rightarrow BC \mid b$	$A \rightarrow BC$| B | C | b
$B \rightarrow b \mid \in$	$B \rightarrow b$
$C \rightarrow c \mid \in$	$C \rightarrow c$
$D \rightarrow d$	$D \rightarrow d$

The grammar $G_1 = (V_1, T_1, P_1, S)$ where

V_1 = { S, A, B, C, D }

T_1 = { a, b, c, d }

P_1 = {
S → ABCa | BCa | ACa | ABaCa | Aa | Ba | a | bD
A → BC | B | C | b
B → b
C → c
D → d
} S is the start symbol

Example 2 : Eliminate all $\in$- productions from the grammar

S → BAAB
A → 0A2 | 2A0 | $\in$
B → AB | 1B | $\in$

Solution :

Step 1 : Obtain the set of nullable variables from the grammar. This can be done using step 1 of theorem 5.5.2 as shown below.

ov	nv	Productions
ϕ	A, B	$A \rightarrow \in$ $B \rightarrow \in$
A, B	A, B, S	$A \rightarrow BAAB$
A, B, S	A, B, S	-

V = { S, A, B } are all nullable variables.

Step 2 : Construction of productions P_1. Add a non $\in$-production in P to P_1. Take all the combinations of nullable variables in a production, delete subset of nullable variables one by one and add the resulting productions to P_1.

Productions	Resulting productions (P_1)
S → BAAB	S → BAAB | AAB | BAB | BAA | AB | BB | BA | AA | A | B
A → 0A 2	A → 0A2 | 02
A → 2A0	A → 2A0 | 20
B → AB	B → AB | B | A
B → 1 B	B → 1 B | 1

We can delete the productions of the form A → A. In P_1, the production $B \to B$ can be deleted and the final grammar obtained after eliminating $\in$-productions is shown below.

The grammar $G_1 = (V_1, T_1, P_1, S)$ where

V_1 = { S, A, B, C, D }

T_1 = { a, b, c, d }

P_1 = { S → BAAB \| AAB \| BAB \| BAA \| AB \| BB \| BA \| AA \| A \| B
A → 0A2 \| 02 \| 2A0 \| 20
B → AB \| A \| 1B \| 1
} S is the start symbol

5.5.3 Eliminating unit productions

Consider the production $A \to B$. The left hand side of the production and right hand side of the production contains only one variable. Such productions are called unit productions. Formally, a unit production is defined as follows.

Definition : Let G = (V, T, P, S) be a CFG. Any production in G of the form

$$A \to B$$

where A, $B \in V$ is a unit production.

In any grammar, the unit productions are undesirable. This is because one variable is simply replaced by another variable.

Example : Consider the productions.

$A \rightarrow B$

$B \rightarrow aB \mid b$

In this example,

$B \rightarrow aB$

$B \rightarrow b$

are non unit productions. Since B is generated from A, whatever is generated by B, the same things can be generated from A also. So, we can have

$A \rightarrow aB$

$A \rightarrow b$ and the production $A \rightarrow B$ can be deleted.

Theorem 5.5.3 : Let G = (V, T, P, S) be a CFG and has unit productions and no $\in$ – productions. An equivalent grammar G_1 without unit productions can be obtained such that $L(G) = L(G_1)$ i. e., any language generated by G is also generated by G_1 . But , the grammar G_1 has no unit productions.

Proof :

A unit production in grammar G can be eliminated using the following steps :

1. Remove all the productions of the form $A \rightarrow A$
2. Add all non unit productions to P_1.
3. For each variable A find all variables B such that

   $A \Rightarrow^* B$

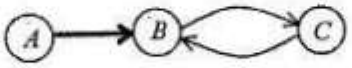
   i. e., in the derivation process from A, if we encounter only one variable in a sentential form say B (no terminals should be there), obtain all such variables.
4. Obtain a dependency graph. For example, if we have the productions

   $A \rightarrow B$

   $B \rightarrow C$

   $C \rightarrow B$

   the dependency graph will be of the form



5. Note from the dependency graph that

   a. $A \Rightarrow^* B$ i. e., B can be obtained from A
   So, all non - unit productions generated from B can also be generated from A

   b. $A \Rightarrow^* C$ i. e., C can be obtained from A
   So, all non - unit productions generated from C can also be generated from A

c. $B \Rightarrow^* C$ i. e., C can be obtained from B

So, all non - unit productions generated from C can also be generated from B

d. $C \Rightarrow^* B$ i. e., B can be obtained from C

So , all non - unit productions generated from B can also be generated from C.

6. Finally, the unit productions can be deleted from the grammar G.

7. The resulting grammar G_1 , generates the same language as accepted by G.

Example1 : Eliminate all unit productions from the grammar

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow C|b \\ C &\rightarrow D \\ D &\rightarrow E|bC \\ E &\rightarrow d|Ab \end{aligned}$$

Solution : The non unit productions of the grammar G are shown below :

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow b \\ D &\rightarrow bC \\ E &\rightarrow d|Ab \qquad \text{..... (1)} \end{aligned}$$

The unit productions of the grammar G are shown below :

$$\begin{aligned} B &\rightarrow C \\ C &\rightarrow D \\ D &\rightarrow E \end{aligned}$$

The dependency graph for the unit productions is shown below :

It is clear from the dependency graph that all non unit productions from E can be generated from D. The non unit productions from E are

$$E \rightarrow d|Ab \qquad \text{..... (2)}$$

Since $D \Rightarrow^* E$,

$$D \rightarrow d \mid Ab$$

The resulting D productions are

$$D \rightarrow bC \text{ (from production(1))}$$
$$D \rightarrow d \mid Ab \quad \text{.....(3)}$$

From the dependency graph it is clear that, $C \Rightarrow^* E$. So, the non unit productions from E shown in (production(2)) can be generated from C. Therefore,

$$C \rightarrow d \mid Ab$$

From the dependency graph it is clear that, $C \Rightarrow^* D$. So, the non unit productions from D shown in (production(3)) can be generated from C. Therefore,

$$C \rightarrow bC$$
$$C \rightarrow d \mid Ab \quad \text{.....(4)}$$

From the dependency graph it is clear that $B \Rightarrow^* C$, $B \Rightarrow^* D$, $D \Rightarrow^* E$. So, all the productions obtained from B can be obtained using (productions (1), (2), (3) and (4)) and the resulting productions are :

$$B \rightarrow b$$
$$B \rightarrow d \mid Ab$$
$$B \rightarrow bC \quad \text{..... (5)}$$

The final grammar obtained after eliminating unit productions can be obtained by combining the productions (Productions (1), (2), (3), (4), and (5)) and is shown below :

V_1 = { S, A, B, C, D, E }
T_1 = { a, b, d }
P_1 = {

S	→	AB
A	→	a
B	→	b | d | Ab | bC
C	→	bC | d | Ab
D	→	bC | d | Ab
E	→	d | Ab

} S is the start symbol

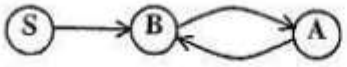
Example 2 : Eliminate unit productions from the grammar

S	→	A 0 | B
B	→	A | 11
A	→	0 | 12 | B

Solution : The unit productions of the grammar G are shown below :

S → B
B → A
A → B

The dependency graph for the unit productions is shown below.



The non unit productions are :

S → A0
B → 11
A → 0 | 12 ..... (1)

It is clear from the dependency graph that $S \Rightarrow^* B, S \Rightarrow^* A$, $B \Rightarrow^* A$ and $A \Rightarrow^* B$. So, the new productions from S, A and B are

S → 11 | 0 | 12
B → 0 | 12
A → 11 .....(2)

The resulting grammar without unit productions can be obtained by combining Productions (1) and (2) and is shown below :

V_1 = { S, A, B } , T_1 = { 0, 1, 2 }
P_1 = { S → A0 | 11 | 0 | 12
A → 0 | 12 | 11
B → 11 | 0 | 12
} S is the start symbol

Note : Given any grammar, all undesirable productions can be eliminated by removing

1. ϵ – productions using theorem 6.5.2
2. unit productions using theorem 6.5.3.
3. useless symbols and productions using theorem 6.5.1

in sequence. The final grammar obtained does not have any undesirable productions.

5.6 NORMAL FORMS

As we have seen the grammar can be simplified by reducing the ϵ production, removing useless symbols, unit productions. There is also a need to have grammar in some specific form. As you have seen in CFG at the right hand of the production there are any number of terminal or non - terminal symbols in any combination. We need to normalize such a grammar. That means we want the grammar in some specific format. That means there should be fixed number of terminals and non - terminals, in the context free grammar.

In a CFG, there is no restriction on the right hand side of a production. The restrictions are imposed on the right hand side of productions in a CFG resulting in normal forms. The different normal forms are :

1. Chomsky Normal Form (CNF)
2. Greiback Normal Form (GNF)

5.6.1 Chomsky Normal Form (CNF)

Chomsky normal form can be defined as follows.

Non - terminal → Non - terminal.Non - terminal
Non - terminal → terminal

The given CFG should be converted in the above format then we can say that the grammar is in CNF. Before converting the grammar into CNF it should be in reduced form. That means remove all the useless symbols, ϵ productions and unit productions from it. Thus this reduced grammar can be then converted to CNF.

Definition :
Let G = (V, T, P, S) be a CFG. The grammar G is said to be in CNF if all productions are of the form

$$A \rightarrow BC$$
or
$$A \rightarrow a$$

where A, B and $C \in V$ and $a \in T$.

Note that if a grammar is in CNF, the right hand side of the production should contain two symbols or one symbol. If there are two symbols on the right hand side those two symbols must be non - terminals and if there is only one symbol, that symbol must be a terminal.

Theorem 5.6.1 : Let G = (V, T, P, S) be a CFG which generates context free language without ϵ. We can find an equivalent context free grammar $G_1 = (V_1, T, P_1, S)$ in CNF such that $L(G) = L(G_1)$ i. e., all productions in G_1 are of the form

$$A \rightarrow BC$$
or
$$A \rightarrow a$$

Proof : Let the grammar G has no $\in$ – productions and unit productions. The grammar G_1 can be obtained using the following steps.

Step 1 : Consider the productions of the form

$$A \rightarrow X_1 X_2 X_3 \ldots\ldots\ldots X_n$$

where $n \geq 2$ and each $X_i \in (V \cup T)$ i. e., consider the productions having more than two symbols on the right hand side of the production. If X is a terminal say a, then replace this terminal by a corresponding non terminal B_a and introduce the production

$$B_a \rightarrow a$$

The non - terminals on the right hand side of the production are retained. The resulting productions are added to P_1. The resulting context free grammar $G_1 = (V_1, T, P_1, S)$ where each production in P_1 is of the form

$$A \rightarrow A_1 A_2 \ldots\ldots\ldots A_n$$

or

$$A \rightarrow a$$

generates the same language as accepted by grammar G. So, $L(G) = L(G_1)$.

Step 2 : Restrict the number of variables on the right hand side of the production. Add all the productions of G_1 which are in CNF to P_1. Consider a production of the form

$$A \rightarrow A_1 A_2 \ldots\ldots\ldots A_n$$

where $n \geq 3$ (Note that if n = 2, the production is already in CNF and n can not be equal to 1. Because if n = 1, there is only one symbol and it is a terminal which again is in CNF). The A - production can be written as

$$\begin{aligned} A &\rightarrow A_1 D_1 \\ D_1 &\rightarrow A_2 D_2 \\ D_2 &\rightarrow A_3 D_3 \\ &\vdots \\ D_{n-2} &\rightarrow A_{n-1} D_{n-1} \end{aligned}$$

These productions are added to P_1 and new variables are added to V_1. The grammar thus obtained is in CNF. The resulting grammar $G_1 = (V_1, T, P_1, S)$ generates the same language as accepted by G i. e. $L(G) = L(G_1)$.

Example 1 : Consider the grammar

S → 0A | 1 B
A → 0AA | 1 S | 1
B → 1 BB | 0 S | 0 Obtain the grammar in CNF :

Solution :

Step 1 : All productions which are in CNF are added to P_1. The productions which are in standard form and added to P_1 are :

A → 1
B → 0 ..... (1)

Consider the productions, which are not in CNF. Replace the terminal a on right hand side of the production by a non - terminal A and introduce the production A → a. This step has to be carried out for each production which are not in CNF.

The table below shows the action taken indicating which terminal is replaced by the corresponding non - terminal and what is the new production introduced. The last column shows the resulting productions.

Given Productions	Action	Resulting productions
S → 0A |1B	Replace 0 by B_0 and introduce the production $B_0 \to 0$ Replace 1 by B_1 and introduce the production $B_1 \to 1$	$S \to B_0 A \mid B_1 B$ $B_0 \to 0$ $B_1 \to 1$
A → 0AA/1S	Replace 0 by B_0 and introduce the production $B_0 \to 0$ Replace 1 by B_1 and introduce the production $B_1 \to 1$	$A \to B_0 AA / B_1 S$ $B_0 \to 0$ $B_1 \to 1$
B → 1BB/0S	Replace 0 by B_0 and introduce the production $B_0 \to 0$ Replace 1 by B_1 and introduce the production $B_1 \to 1$	$B \to B_1 BB / B_0 S$ $B_1 \to 1$ $B_0 \to 0$

The grammar $G_1 = (V_1, T, P_1, S)$ can be obtained by combining the productions obtained from the last column in the table and the productions shown in (1).

V_1 = { S, A, B, B_0, B_1 }

T_1 = { 0, 1 }

P_1 = {

S → $B_0A | B_1B$

A → $B_0AA | B_1S | 1$

B → $B_1BB | B_0S | 0$

B_0 → 0

B_1 → 1

} S is the start symbol

Step 2 :

Restricting the number of variables on the right hand side of the production to 2. The productions obtained after step 1 are :

$S \rightarrow B_0A \mid B_1B$

$A \rightarrow B_0AA \mid B_1S \mid 1$

$B \rightarrow B_1BB \mid B_0S \mid 0$

$B_0 \rightarrow 0$

$B_1 \rightarrow 1$

In the above productions, the productions which are in CNF are

$S \rightarrow B_0A \mid B_1B$

$A \rightarrow B_1S \mid 1$

$B \rightarrow B_0S \mid 0$

$B_0 \rightarrow 0$

$B_1 \rightarrow 1$ ..... (2)

and add these productions to P_1 . The productions which are not in CNF are

$A \rightarrow B_0AA$

$B \rightarrow B_1BB$

The following table shows how these productions are changed to CNF so that only two variables are present on the right hand side of the production.

Given Productions	Action	Resulting productions
$A \to B_0 AA$	Replace AA on R.H.S with variable D_1 and introduce the production $D_1 \to AA$	$A \to B_0 D_1$ $D_1 \to AA$
$B \to B_1 BB$	Replace BB on R. H. S with variable D_2 and introduce the production $D_2 \to BB$	$B \to B_1 D_2$ $D_2 \to BB$ .....(3)

The final grammar which is in CNF can be obtained by combining the productions in (2) and (3).

The grammar $G_1 = (V_1, T, P_1, S)$ is in CNF where

V_1 = { S, A, B, B_0, B_1, D_1, D_2 }

T_1 = { 0, 1 }

P_1 = {
$S \to B_0 A \mid B_1 B$
$A \to B_1 S \mid 1 \mid B_0 D_1$
$B \to B_0 S \mid 0 \mid B_1 D_2$
$B_0 \to 0$
$B_1 \to 1$
$D_1 \to AA$
$D_2 \to BB$
} S is the start symbol

Example 2 : Find a grammar in CNF equivalent to the grammar :

$$S \to -S \mid [\, S \uparrow S \,] \mid a \mid b$$

Solution : Given, grammar is :

$$S \to -S \mid [\, S \uparrow S \,] \mid a \mid b \qquad \text{.....(A)}$$

where, terminals are :

$-, [, \uparrow,], a$ *and* b

In the given grammar (A) there is no any $\in$–production, no any unit - production and no any useless symbols.

Now, in the given grammar (A), following are the productions which is already in the form of CNF:

$S \to a$

$S \to b$

Also, in the given grammar (A), following are the productions which are not in the form of CNF:

$$S \to -S$$

$$S \to [S \uparrow S]$$

Thus: (a) Considering the production:

$$S \to -S$$

We can write this production as:

$$S \to V_1 S \quad \text{.... (1)}$$

$$V_1 \to - \quad \text{.... (2)}$$

where V_1 is a new variable.

(b) Now, considering the production:

$$S \to [S \uparrow S]$$

We can write this production as:

$$S \to V_2\, SV_3\, SV_4 \quad \text{.... (3)}$$

$$V_2 \to [\quad \text{.... (4)}$$

$$V_3 \to \uparrow \quad \text{.... (5)}$$

$$V_4 \to] \quad \text{.... (6)}$$

where V_2, V_3 and V_4 are new variables.

Thus, from (1), ..., (6), the result grammar becomes:

$$S \to V_1 S \mid V_2 S\, V_3\, SV_4 \mid a \mid b$$

$$V_1 \to -$$

$$V_2 \to [\quad \text{.....(B)}$$

$$V_3 \to \uparrow$$

$$V_4 \to]$$

Now, in the resultant grammar (B), following is the production which is not in the form of CNF:

$$S \to V_2\, SV_3\, SV_4$$

(c) Now, considering the production:

$$S \to V_2 SV_3 SV_4$$

We can write this production as:

$$S \to V_2 V_5 V_6 \quad \text{.....(7)}$$

$$V_5 \to SV_3 \quad \text{.....(8)}$$

$$V_6 \to SV_4 \quad \text{.....(9)}$$

Thus, from (7), (8) and (9), the resultant grammar becomes :

$$S \rightarrow V_1\ S \mid V_2 V_5 V_6 \mid a \mid b$$
$$V_1 \rightarrow -$$
$$V_2 \rightarrow [$$
$$V_5 \rightarrow SV_3 \qquad \text{.....(C)}$$
$$V_6 \rightarrow SV_4$$
$$V_3 \rightarrow \uparrow$$
$$V_4 \rightarrow]$$

Now, in the resultant grammar (C), following is the production which is not in the form of CNF:

$$S \rightarrow V_2 V_5 V_6$$

We can write this production as :

$$S \rightarrow V_2 V_7 \qquad \text{.....(10)}$$
$$V_7 \rightarrow V_5 V_6 \qquad \text{.....(11)}$$

Thus, from (10) and (11), the resultant grammar becomes :

$$S \rightarrow V_1 S \mid V_2 V_7 \mid a \mid b$$
$$V_1 \rightarrow -$$
$$V_2 \rightarrow [$$
$$V_7 \rightarrow V_5 V_6 \qquad \text{.....(D)}$$
$$V_5 \rightarrow SV_3$$
$$V_6 \rightarrow SV_4$$
$$V_3 \rightarrow \uparrow$$
$$V_4 \rightarrow]$$

Thus, the resultant grammar (D) is in the form of CNF, which is the required solution.

5.6.2 Greibach Normal form (GNF)

Greibach normal form can be defined as follows :

Non - terminal → one terminal. Any number of non - terminals

Example :

$S \rightarrow aA$ is in GNF
$S \rightarrow a$ is in GNF

But $S \to AA$ is not in GNF

$S \to Aa$ is not in GNF

Definition : A CFG $G = (V, T, P, S)$ is in Greibach normal form (GNF) if its all productions are of type $A \to a\alpha$, where $\alpha \in V^*$ (String of variables including null string) and $a \in T$. A grammar in GNF is the natural generalization of a regular grammar (right - linear).

Theorem 5.6.2 : Every CFL L without $\in$ is generated by grammar, where productions are of type $A \to a\alpha$, where $\alpha \in V^*$ and $a \in T$.

Proof : We use removal of left recursion (without null productions) as given below.

Let the variable A has left recursive productions given as follows :

$A \to A\alpha_1 | A\alpha_2 | A\alpha_3 | .... | A\alpha_n | \beta_1 | \beta_2 | \beta_3 | .... | B_m$, where $\beta_1, \beta_2, \beta_3, ... B_m$ do not begin with A, then we replace A - productions by the productions given below .

$$A \to \beta_1 A' | \beta_2 A' | .... | \beta_m A' | \beta_1 | \beta_2 | \beta_3 | ... | B_m, \text{ where}$$

$$A' \to \alpha_1 A' | \alpha_2 A' | \alpha_3 A' | .... | \alpha_n A' | \alpha_1 | \alpha_2 | \alpha_3 | ... | \alpha_n$$

Method for Converting a CFG into GNF :

We consider CFG $G = (V, T, P, S)$.

Step 1 : Rename all the variables of G as $A_1, A_2, A_3, ..., A_n$

Step 2 : Repeat Step 3 and Step 4 for $i = 1, 2, ..., n$

Step 3 : If $A_i \to a\alpha_1\alpha_2\alpha_3 ..... \alpha_m$, where $a \in T$, and α_j is a variable or a terminal symbol,

Repeat for $j = 1, 2, ..., m$

If α_j is a terminal then replace it by a variable A_{n+j} and add production $A_{n+j} \to \alpha_j$, and $n = n + 1$. Consider the next A_i – production and go to step 3.

Step 4 : If $A_i \to \alpha_1\alpha_2\alpha_3 .....\alpha_m$, where α_1 is a variable, then perform the following :

If α_1 is same as A_i, then remove the left recursion and go to Step 3.

Else replace α_1 by all RHS of α_1 -productions one by one. Consider the remaining A_i -productions, which are not in GNF and go to Step 3.

Step 5 : Exit

Advantages of GNF :

1. Avoids left recursion.
2. Always has terminal in leftmost position in RHS of each production.
3. Helps select production correctly.
4. Guarantees derivation length no longer than string length.

Example 1 : Consider the CFG $S \to S+S \mid S*S \mid a \mid b$ and find an equivalent grammar in GNF.

Solution: Let G_1 is the equivalent grammar in GNF.

Renaming the variable, we get

$P_1 : S_1 \to S_1 + S_1$ (Not in GNF)

$P_2 : S_1 \to S_1 * S_1$ (Not in GNF)

$P_3 : S_1 \to a$ (In GNF)

$P_4 : S_1 \to b$ (In GNF)

P_1 and P_2 are left recursive productions, so removing the left recursion, we get

$S_1 \to a S_2 \mid b S_2 \mid a \mid b$, where

$S_2 \to + S_1 S_2 \mid * S_1 S_2 \mid + S_1 \mid * S_1$

Now, all productions are in GNF.

Example 2 : Consider the grammar $G = (\{A_1, A_2, A_3\}, \{a,b\}, P, A_1)$, where P consists of following production rules.

$A_1 \to A_2 A_3, A_2 \to A_3 A_1 \mid b, A_3 \to A_1 A_2 \mid a$ Convert it into GNF.

Solution : (Renaming is not required)

Consider A_1 - productions :

$A_1 \to A_2 A_3$ (Not in GNF)

Replacing A_2 by its RHS, we get

$A_1 \to b A_3$ (In GNF)

$A_1 \to A_3 A_1 A_3$ (Not in GNF)

Now , consider $A_1 \to A_3 A_1 A_3$, and replacing A_3 by its RHS, we get

$A_1 \to a A_1 A_3$ (In GNF)

$A_1 \to A_1 A_2 A_1 A_3$ (Not in GNF)

So, A_1 - productions are $A_1 \to b A_3 \mid a A_1 A_3 \mid A_1 A_2 A_1 A_3$

Now, consider $A_1 \to A_1 A_2 A_1 A_3$ and removing left recursion, we get

$A_1 \to b A_3 A_4 \mid b A_3$ (In GNF)

$A_1 \to a A_1 A_3 A_4 \mid a A_1 A_3$ (In GNF), where

$A_4 \to A_2 A_1 A_3 A_4 \mid A_2 A_1 A_3$

(A_4 is a new variable and its production is not in GNF)

So, now all A_1 - productions are in GNF .

Consider the A_2 - productions :

$A_2 \to b$ (In GNF)

$A_2 \to A_3 A_1$ (Not in GNF)

Now, consider $A_2 \to A_3 A_1$ and replacing A_3 by its RHS, we get

$A_2 \to aA_1$ (In GNF)

$A_2 \to A_1 A_2 A_1$ (Not in GNF)

Now, consider $A_2 \to A_1 A_2 A_1$ and replacing A_1 by its RHS, we get

$A_2 \to bA_3 A_4 A_2 A_1$, (In GNF)

$A_2 \to bA_3 A_2 A_1$ (In GNF)

$A_2 \to aA_1 A_3 A_4 A_2 A_1$ (In GNF)

$A_2 \to aA_1 A_3 A_2 A_1$ (In GNF)

So , all A_2 - productions are in GNF.

Consider A_3 - productions :

$A_3 \to a$ (In GNF)

$A_3 \to A_1 A_2$ (Not in GNF)

Now, consider $A_3 \to A_1 A_2$ and replacing A_1 by its RHS, we get

$A_3 \to bA_3 A_4 A_2 \mid bA_3 A_2 \mid aA_1 A_3 A_4 A_2 \mid aA_1 A_3 A_2$ (In GNF)

Consider A_4 - productions :

$A_4 \to A_2 A_1 A_3 A_4 \mid A_2 A_1 A_3$ (Not in GNF)

Replacing A_2 by its RHS, we get

$A_4 \to bA_1 A_3 A_4 \mid bA_1 A_3 \mid aA_1 A_1 A_3 A_4 \mid aA_1 A_1 A_3$,

$A_4 \to bA_3 A_4 A_2 A_1 A_1 A_3 A_4$,

$A_4 \to bA_3 A_4 A_2 A_1 A_1 A_3$,

$A_4 \to bA_3 A_2 A_1 A_1 A_3 A_4$,

$A_4 \to bA_3 A_2 A_1 A_1 A_3$,

$A_4 \to aA_1 A_3 A_4 A_2 A_1 A_1 A_3 A_4$,

$A_4 \to aA_1 A_3 A_4 A_2 A_1 A_1 A_3$,

$A_4 \to aA_1 A_3 A_2 A_1 A_1 A_3 A_4$, and

$A_4 \to aA_1 A_3 A_2 A_1 A_1 A_3$

Now, all A_4 - productions are in GNF.

Productions in GNF are :

$A_1 \to aA_1A_3 \mid bA_3A_4 \mid bA_3 \mid aA_1A_3A_4 \mid aA_1A_3$

$A_2 \to b \mid aA_1 \mid bA_3A_4A_2A_1 \mid bA_3A_2A_1 \mid aA_1A_3A_4A_2A_1 \mid aA_1A_3A_2A_1$,

$A_3 \to a \mid bA_3A_4A_2 \mid bA_3A_2 \mid aA_1A_3A_4A_2 \mid aA_1A_3A_2$,

$A_4 \to bA_1A_3A_4 \mid bA_1A_3 \mid aA_1A_1A_3A_4 \mid aA_1A_1A_3 \mid bA_3A_4A_2A_1A_1A_3A_4$,

$A_4 \to bA_3A_2A_1A_1A_3A_4 \mid aA_1A_3A_4A_2A_1A_1A_3A_4 \mid aA_1A_3A_2A_1A_1A_3A_4$,

$A_4 \to bA_3A_4A_2A_1A_1A_3 \mid bA_3A_2A_1A_1A_3 \mid aA_1A_3A_4A_2A_1A_1A_3 \mid aA_1A_3A_2A_1A_1A_3$

Example 3 : Find equivalent grammar in GNF.

(a) S → aB | b A, A → aS | b AA | a, B → bS | a BB | b
(b) S → abSb | a | a Ab, A → bS | a AAb
(c) S → AA | 0 , A → SS | 1

Solution :

(a) Renaming S, A and B by A_1, A_2, and A_3 respectively, we get the following productions.

$A_1 \to aA_3 \mid bA_2, A_2 \to aA_1 \mid bA_2A_2 \mid a, A_3 \to bA_1 \mid aA_3A_3 \mid b$

Since, all productions are in GNF, so there is no need of any modification.

(b) Renaming S, and A by A_1 and A_2 respectively, we get the following productions.

$A_1 \to abA_1b \mid a \mid aA_2b, A_2 \to bA_1 \mid aA_2A_2b$

Consider the A_1 - productions one by one.

$A_1 \to abA_1b$ (Not in GNF)

Replacing all the RHS terminals except the first by new variables, we get

$A_1 \to aA_3A_1A_3$ where $A_3 \to b$ (In GNF)

Considering the next A_1 - production :

$A_1 \to a$ (In GNF)

Considering the next A_1 - production :

$A_1 \to aA_2b$ (Not in GNF)

Replacing b by variable A_3 (since, we have already defined $A_3 \to b$), we get

$A_1 \to aA_2A_3$ (In GNF)

Consider the A_2 - production :

$A_2 \to bA_1$ (In GNF)

Considering the next A_2 - production :

$A_2 \to aA_2A_2b$ (Not in GNF)

Replacing b by variable A_3 (since, we have already defined $A_3 \to b$), we get

$A_2 \to aA_2A_2A_3$ (In GNF)

Now, all productions given following are in GNF.

$A_1 \to aA_3A_1A_3 | a | aA_2A_3$, $A_2 \to bA_1 | aA_2A_2A_3$, and $A_3 \to b$

(c) Renaming S, and A by A_1, and A_2 respectively, we get the following productions

$A_1 \to A_2A_2 | 0$, $A_2 \to A_1A_1 | 1$

Consider the A_1 - productions one by one.

$A_1 \to A_2A_2$ (Not in GNF)

Replacing leftmost A_2 by A_1A_1 and 1, we get

$A_1 \to A_1A_1A_2 | 1A_2$

Considering the production $A_1 \to A_1A_1A_2$, this is not in GNF and has left recursion. Considering the all A_1 - productions $A_1 \to A_1A_1A_2 | 1A_2 | 0$ and removing left recursion from the production $A_1 \to A_1A_1A_2$, we get $A_1 \to 1A_2A_3 | 0A_3$ (In GNF),

Where $A_3 \to A_1A_2A_3 | A_1A_2$

Considering A_2 - production $A_2 \to A_1A_1$ and replacing left most A_1 by $1A_2A_3$ and $0A_3$, we get

$A_2 \to 1A_2A_3A_1 | 0A_3A_1$ (In GNF)

Considering A_3 - productions $A_3 \to A_1A_2A_3 | A_1A_2$ and replacing A_1 by $1A_2A_3$ and $0A_3$, we get

$A_3 \to 1A_2A_3A_2 | 0A_3A_2 | 1A_2A_3A_2A_3 | 0A_3A_2A_3$ (In GNF)

Now, the productions in GNF are following .

$A_1 \to 1A_2A_3 | 0A_3$, $A_2 \to 1A_2A_3A_1 | 0A_3A_1 | 1$,

and $A_3 \to 1A_2A_3A_2 | 0A_3A_2 | 1A_2A_3A_2A_3 | 0A_3A_2A_3$

5.7 PUMPING LEMMA FOR CFLs

The pumping lemma for CFLs states that there are always two short substrings close together that can be pumped same number of times as we like and the result is a string in the same CFL.

Lemma :

Let L be a CFL and a long string z is in L, then there exists a constant n such that $|z| \geq n$ and z can be written as uvwxy such that

(i) $|vx| \geq 1$

(ii) $|vwx| \leq n$, and

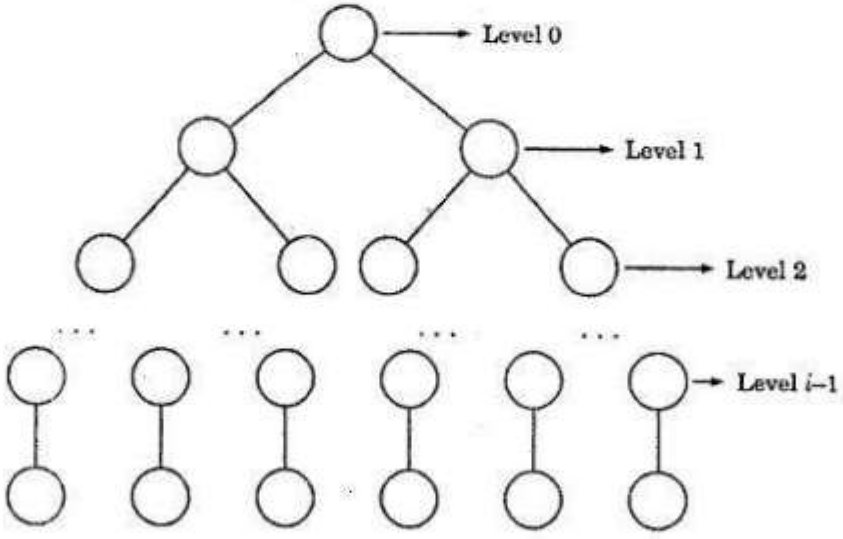
(iii) $uv^i wx^i y$ is in L for i = 0, 1, 2,.....

Proof :

Let G be a CFG in CNF and generates $L - \{\in\}$. Since, z is a long string, so parse tree for z must contain a long path. Suppose, the longest path in parse tree of z has length h. In the parse tree, no word can be greater that the length 2^{h-1} or in other words, the maximum length word would of length 2^{h-1}.

We see the proof as follows :

Since, the grammar G is in CNF (productions are of types $A \to a$ or $A \to XY$), so parse tree for z is a binary tree. The parse tree yields longest word if and only if its all levels except the last level contain two children as shown in below figure .



Since, the number of leaves is the length of longest string and it is equal to the number of nodes at level $i-1$ as shown in above figure .The number of nodes at level $i-1=2^{i-1}$

So, the longest word has the length 2^{h-1}, where h is the longest path length. In other words, we say that no word can be greater than 2^{h-1} length.

Let G has k variables and $n=2^k$. If z is in L(G) and $|z|\geq 2^k$. So, the longest path in the parse tree of z has length $k+1$ and this path contains $k+2$ vertices ($k+1$ internal vertices and one terminal vertex). Since, all the vertices except the terminal are variables, so the longest path contains $k+1$ variables. It means, one variable appears twice in the longest path. Let variable A appears twice, So $A \underset{G}{\overset{*}{\Rightarrow}} z_3\ Az_4 \underset{G}{\overset{*}{\Rightarrow}} (z_3)^i\ A(z_4)^i$, where z_3 and z_4 are two substrings of z. Let $A \underset{G}{\overset{*}{\Rightarrow}} z_2$ then $A \underset{G}{\overset{*}{\Rightarrow}} (z_3)^i z_2 (z_4)^i$. We say that z_3 and z_4 can be pumped same number of times as we like.

Example : Consider a CFG $S \rightarrow SS \mid a$ and $z = aaaa$. The parse tree for z is shown in figure(a) .

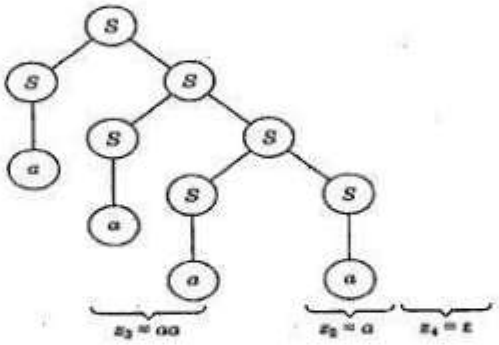


Figure (a)

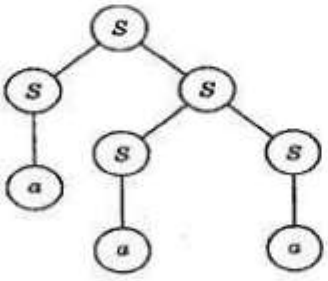


Figure (b)

Figure (c)

From the subtree shown in figure (b), we get $S \overset{*}{\Rightarrow} aaS \in$ or $S \overset{*}{\Rightarrow} z_3 S z_4$ and considering the subtree shown in figure(c), we get $S \overset{*}{\Rightarrow} a$ or $S \overset{*}{\Rightarrow} z_2$.

The subtree shown in figure (b) can be added as many times as we like in the parse tree shown in figure (a). So, $S \overset{*}{\Rightarrow} z_3^i S z_4^i \overset{*}{\Rightarrow} z_3^i z_2 z_4^i$

Therefore, string z can be written as $uz_3z_2z_4y$ for some u and y substrings of z. The substrings z_3 and z_4 can be pumped as many times as we like. Replacing z_3, z_2 and z_4 by v, w and x respectively, we get z = uvwxy and $S \overset{*}{\Rightarrow} uv^iwx^iy$ for some i = 0, 1, 2, ...
Hence, the statement of theorem is proved.

Application of Pumping Lemma for CFLs

We use the pumping lemma to prove certain languages are not CFL. We proceed as we have seen in application of pumping lemma for regular sets and get contradiction. The result of this lemma is always negative.

Procedure for Proving Language is not Context - free

The following steps are considered to show a given language is not context - free.

Step 1 :

Suppose that L is context - free. Let l be the natural number obtained by using pumping lemma.

Step 2 :

Choose a string $x \in L$ such that $|x| \geq l$ using pumping lemma principle write z = uvwxy.

Step 3 :

Find suitable i so that $uv^iwx^iy \notin L$. This is a contradiction. So L is not context - free.

Example 1 : Consider the language $L = \{a^n\ b^n\ c^n : n \geq 1\}$ and prove that L is not CFL.

Solution : All the words of L contain equal number of a's, b's and c's. Let L is a CFL and z is a long string in L such that $|z| \geq n$. Using Pumping Lemma for L, we write $z = uvwxy$ and uv^iwx^iy is in L for some i = 0, 1, 2, ... and $|vx| \geq 1$ and $|vwx| \leq n$.

The substring vx may be a^p, b^q, c^r, a^pb^q, b^qc^r but not a^pc^r.

Consider i = 0, so uwy is in L.

Case 1 : $vx = a^p$, so $z = uwy = a^{n-p}b^nc^n$ is in L.

The number of a's is fewer than the number of b's and c's for $p \geq 1$, which is a contradiction.

Case 2 : $vx = b^q$, so z = uwy = $a^nb^{n-q}c^n$ is in L.

The number of b's is fewer than the number of a's and c's for $q \geq 1$, which is a contradiction.

Case 3 : $vx = c^r$, so $z = uwy = a^nb^nc^{n-r}$ is in L.

The number of c's is fewer than the number of a's and b's for $r \geq 1$, which is a contradiction.

Case 4 : $vx = a^pb^q$, so $z = uwy = a^{n-p}b^{n-q}c^n$ is in L.

The number of a's and b's are fewer than the number of c's for $p,q \geq 1$, which is a contradiction.

Case 5 : $vx = b^qc^r$, so $z = uwy = a^nb^{n-q}c^{n-r}$ is in L.

The number of b's and c's are fewer than the number of a's for $q,r \geq 1$, which is a contradiction.

Since, we get contradiction for all values of vx, so L is not a CFL.

Example 2 : Prove that following languages are not CFL

(a) $\mathbf{L = \{a^p : p \text{ is a prime number}\}}$

(b) $\mathbf{L = \{a^nb^mc^nd^m : m, n \geq 1\}}$

(c) $\mathbf{L = \{a^nb^nc^m : m \geq n\}}$

Solution :

(a) All the words of L have length prime. Let L be a CFL and z is a long string in L. Using Pumping Lemma for L, we write $z = uvwxy$ and uv^iwx^iy is in L for some $i = 0, 1, 2, ...$ and $|vx| = m$ and $|uwy| = n$ where n is a prime number then $|uv^nwx^ny| = n + mn$. As $n + mn$ is not a prime number, so $uv^nwx^ny \notin L$ and this is a contradiction. Therefore, L is not a CFL.

(b) Let L be a CFL and z is a long string in L such that $z = uvwxy$ for $|vx| = 1$ and $|vwx| = k$, where k is some constant.

In L, all words have equal number of a's and c's and equal number of b's and d's. The value of vx may be combination of two consecutive symbols like $a^p b^q$, $b^q c^r$, $c^r d^s$.

According to pumping lemma $uv^i wx^i y$ is in L for some $i = 0, 1, 2, ...$

Consider $i = 0$, then $z = uwy$ is in L.

Case 1 : $vx = a^p b^q$, then

$$z = a^{n-p} b^{m-q} c^n d^m$$

The number of a's and b's are fewer than the number of c's and d's for $p, q \geq 1$, which is a contradiction.

Case 2 : $vx = b^q c^r$, then $z = a^n b^{m-q} c^{n-r} d^m$

The number of b's and c's are fewer than the number of d's and a's for $q, r \geq 1$, which is a contradiction.

Case 3 : $vx = c^r d^s$, then $z = a^n b^m c^{n-r} d^{m-s}$

The number of c's and d's are fewer than number of a's and b's for $r, s \geq 1$, which is a contradiction.

Since, we are getting contradiction in all cases, so L is not a CFL.

(c) All the words of L contain equal number of a's, b's and number of c's is greater than number of a's (or b's). Let L is a CFL and z is a long string in L such that $|z| \geq n$. Using pumping lemma for L, we write $z = uvwxy$ and $uv^i wx^i y$, which are in L for some $i = 0, 1, 2, ...$ and $|vx| \geq 1$ and $|vwx| \leq n$.

The substring vx may be a^p, b^q, c^r, $a^p b^q$, $b^q c^r$ but not $a^p c^r$.

Consider $i = 0$, so uwy is in L.

Case 1 : $vx = a^p$, so $z = uwy = a^{n-p} b^n c^n$ is in L.

The number of a's is fewer than the number of b's for $p \geq 1$, which is a contradiction.

Case 2 : $vx = b^q$, so $z = uwy = a^n b^{n-q} c^n$ is in L.

The number of b's is fewer than the number of a's for $q \geq 1$, which is a contradiction.

Case 3 : $vx = c^r$, so $z = uwy = a^n b^n c^{n-r}$ is in L.

The number of c's may be equal or less than the number of a's (or b's) for $r \geq 1$, which is a contradiction.

Since, we are getting contradiction in all cases, so L is not a CFL.

Example 3 : Show that the following language is not context free $\mathbf{L = \{a^{n^2} / n \geq 1\}}$

Solution :

Method - I : Assume L is context - free and n is the pumping lemma constant

Let $Z = a^{n^2}$

write $Z = uvwxy$, where $|vwx| \leq n$ and $|vx| \geq 1$

Let $|vx| = m$, $m \leq n$

As $|uv^2wx^2y| > n^2$, $|uv^2wx^2y| = k^2$, where k *is* $\geq n + 1$

But $|uv^2wx^2y| = n^2 + m < n^2 + 2n + 1$

So $|uv^2wx^2y|$ strictly lies between n^2 and $(n + 1)^2$ which means $uv^2wx^2y \notin L$, a contradiction. Hence $\{a^{n^2} : n \geq 1\}$ is not context-free

Method - II : We can also show that

$L = \{a, aaaa, aaaaaaaaa, ...\}$

$Z = \underset{v}{a}\ \underset{w}{a}\ \underset{x}{a}\ \underset{y}{a}$

$uv^2wx^2y = \in a^2aa^2a = aaaaaa$

$uv^2wx^2y \notin L$

$\therefore$ L is not context-free.

Example 4 : Show that the following language is not context-free

$$L = \{0^m 1^m 2^n / m \le n \le 2m\}.$$

Solution :

Method - I :

Assume L is context-free and n is the pumping lemma constant.

Let $Z = 0^m 1^m 2^n$

Then $Z = uvwxy$, where $1 \le |vx| \le n$

So vx cannot contain all the three symbols 0, 1 and 2. If vx contains only 0's and 1's then we can choose i such that $uv^i wx^i y$ has more than $2n$ occurrences of a 0 (or 1) and exactly $2n$ occurrences of L. This means $uv^i wx^i y \notin L$, a contradiction.

In other cases also we can get a contradiction by proper choice of i. Thus the given language is not context - free.

Method - II :

Consider the accepted set of strings from the given language

$L = \{0122, 0011222, 00112222, ...\}$

$$Z = \underset{u}{0}\ \underset{v}{01}\ \underset{w}{1}\ \underset{x}{22}\ \underset{y}{2}$$

$uv^2wx^2y = 0\,(01)^2 1\,(22)^2 2 = 0\ 01011\ 22222 \notin L$

$\therefore$ L is not context-free.

5.7. 2 Ogden's Lemma and Its Applications

There exist some non - context free languages which cannot be proved using the lemma of section 5.7. We need a stronger result. Ogden's lemma is more powerful than the pumping lemma. This lemma allows us to fix 'distinguished positions' in the sentence z and puts some conditions for v, x, y with respect to these positions. Proof of Ogden's lemma is beyond the scope of this book. However, we present the statement of Ogden's lemma and illustrate its application.

Statement of Ogden's Lemma

Let L be a context free language. There exists a constant n such that for any sentence $z, |z| \geq n$, we can fix at least n distinguished positions, and z can be written as uvwxy such that

i. vx contains at least one distinguished position,
ii. vwx contains at most n distinguished positions ; and
iii. any string of the form $uv^i wx^i y, i \geq 0$ is in L.

Note :

1. Pumping lemma of 5.7 is a special case of Ogden's lemma in which every position in z is distinguished.
2. In applying Ogden's lemma, choice of distinguished positions is under our control.

Example :

Prove that $L = \{ a^i b^j c^k \mid i \neq j,\ j \neq k$ and $i \neq k \}$ is not context free.

Solution :

If L is context free we can apply Ogden's lemma. Let n be the constant of the lemma. Consider the sentence $z = a^n b^{n!+n} c^{2n!+n}$. We will choose all positions in the block of a's as distinguished. z can be split as uvwxy such that (i) vx has at least one distinguished position and (ii) vwx has at most n distinguished positions : By (i), vx should contain at least one a. These are different cases.

Case 1 :

$v \in a^+$ and $x \in b^*$. Let $u = a^p$ such that $1 \leq p < n$. Then, p is divisor of n!. Let q be the integer such that $pq = n!$.

Consider $z' = uv^{2q+1} wx^{2q+1} y$.

y consists of $(2n!+n)$ $c's$ (remains unchanged).

$$v^{2q+1} = a^{2pq+p} = a^{2n!+p}$$
$$uv^{2q+1} = a^{n-p} a^{2n!+p} = a^{2n!+n}$$

Hence in z', number of a's = number of c's.

Case 2 :

$v \in a^+$ and $x \in c^*$. Let $v = a^p$ and pq=n!. Pumping v and x, $(q+1)$ times, we get : $z' = uv^{q+1}wx^{q+1}y$.

In z', no. of a's will be $n - p + n! + p = n! + n$.

No.of b's in z' will remain n! + n. Hence, no. of a's = no. of b's in z'.

Similarly, in other cases, we can arrive at strings not as per specification of L.

Hence, L is not context free.

5.8 CLOSURE PROPERTIES OF CFLs

The closure properties that hold for regular languages do not always hold for context free languages. Consider those operations which preserve CFL.

The purpose of these operations are to prove certain languages are CFL and certain languages are not CFL.

Context-free languages are closed under following properties.

1. Union
2. Concatenation and
3. Kleene Closure (Context-free languages **may** or **may not** close under following properties)
4. Intersection
5. Complementation

Theorem 5.8.1 : If L_1 and L_2 are two CFLs, then union of L_1 and L_2 denoted by $L_1 + L_2$ or $L_1 \cup L_2$ is also a CFL.

Proof :

Let CFG $G_1 = (V_1, T_1, P, S)$ generates L_1 and CFG $G_2 = (V_2, T_2, P, S)$ generates L_2 and $G = (V, T, P, S)$ generates $L = L_1 + L_2$.

We construct G as follows :

Step 1 : Rename the variables of CFG G_1

If $V_1 = \{S, A, B, ..., X\}$, then the renamed variables are $\{S_1, A_1, B_1, ...X_1\}$. This modification should be reflected in productions also.

Step 2 : Rename the variables of CFG G_2

If $V_2 = \{S, A, B, \ldots X\}$, then the renamed variables are $\{S_2, A_2, B_2 \ldots X_2\}$. This modification should be reflected in production also.

Step 3 : We get of the productions of G_1 and G_2 to get productions of G as follows :

$S \rightarrow S_1 \mid S_2$, where S_1 and S_2 are starting symbols of grammars G_1 and G_2 respectively and S_1 - productions and S_2 - productions remain unchanged.

$T = T_1 \cup T_2$,

$V = \{S_1, A_1, B_1, \ldots X_1\} \cup \{S_2, A_2, B_2, \ldots X_2\}$

Since, all productions of G_1 and G_2 including $S \rightarrow S_1 \mid S_2$ are in context-free form, so G is a CFG.

Language generated by G :

$L(G)$ = Language generated from (S_1 *or* S_2)

= Language generated from S_1 or language generated from S_2

= $L(G_1)$ or $L(G_2)$ (Since, S_1 and S_2 are starting symbols of G_1 and G_2 respectively.)

= L_1 *or* L_2 (Since, G_1 produces L_1 and G_2 produces L_2.)

= $L_1 + L_2$

Hence, statement of the theorem is proved.

Example : Consider the CFGs $S \rightarrow aSb \mid ab$ and $S \rightarrow cSdd \mid cdd$, which generate languages L_1 and L_2 respectively. Construct grammar for $L = L_1 + L_2$.

Solution :

Let G_1 generates L_1 and G_2 generates L_2 and $G = (V, T, P, S)$ generates $L = L_1 + L_2$.

Renaming the variables of G_1 and G_2, we get

$V_1 = \{S_1\}$ and $V_2 = \{S_2\}$, where S_1 - productions are $S_1 \rightarrow aS_1b \mid ab$, and S_2 - productions are $S_2 \rightarrow cS_2dd \mid cdd$

We define G as follows :

$V = \{S, S_1, S_2\}$,

$T = \{Terminals\ of\ G_1\ or\ G_2\} = \{a, b, c, d\}$,

P includes : $S \to S_1 \mid S_2$, $S_1 \to aS_1b \mid ab$, and $S_2 \to cS_2dd \mid cdd$.

$L = L_1 + L_2$

$= \{a^m b^n : m, n \geq 1\} \cup \{c^n d^{2n} : n \geq 1\}$

Theorem 5.8.2 : If L_1 and L_2 are two CFLs, then concatenation of L_1 and L_2 denoted by L_1L_2 is also a CFL .

Proof : Let CFG $G_1 = (V_1, T_1, P, S)$ generates L_1 and CFG $G_2 = (V_2, T_2, P, S)$ generates L_2 and $G = (V, T, P, S)$ generates $L = L_1L_2$.

We construct G as follows :

Step 1 : Rename the variables of CFG G_1.

If $V_1 = \{S, A, B, ..., X\}$, then the renamed variables are $\{S_1, A_1, B_1, ...X_1\}$. This modification is reflected in productions also.

Step 2 : Rename the variables of CFG G_2 :

If $V_2 = \{S, A, B, ... X\}$, then the renamed variables are $\{S_2, A_2, B_2, ...X_2\}$. This modification is reflected in productions also.

Step 3 : The productions of G_1 are followed by the productions of G_2 to get productions of G as follows.

$S \to S_1S_2$, where S_1 and S_2 are starting symbols of grammars G_1 and G_2 respectively and S_1 - productions and S_2 - productions remain unchanged.

$T = T_1 \cup T_2$,

$V = \{S_1, A_1, B_1, ..., X_1\} \cup \{S_2, A_2, B_2, ..., X_2\}$

Since, all productions of G_1 and G_2 including $S \to S_1S_2$ are in context-free form, so G is a CFG.

Language Generated by G :

$L(G)$ = Language generated from S_1 followed by language generated from S_2

$= L(G_1)\ L(G_2)$ (Since, S_1 and S_2 are starting symbols of G_1 and G_2 respectively).

$= L_1L_2$ (Since, G_1 produces L_1 and G_2 produces L_2 .)

Hence, statement of the theorem is proved.

Example : Consider the CFGs $S \rightarrow aSb \mid ab$ and $S \rightarrow cSdd \mid cdd$, which generate languages L_1 and L_2 respectively. Construct grammar for $L = L_1L_2$.

Solution :

Let G_1 generates L_1 and G_2 generates L_2 and $G = (V, T, P, S)$ generates $L = L_1L_2$. Renaming the variables of G_1 and G_2, we get

$V_1 = \{S_1\}$ and $V_2 = \{S_2\}$, where S_1 - productions are : $S_1 \rightarrow aS_1b \mid ab$, and S_2 - productions are : $S_2 \rightarrow cS_2dd \mid cdd$.

We define G as follows :

$V = \{S, S_1, S_2\}$, $\Sigma = \{Terminals\ of\ G_1\ or\ G_2\} = \{a, b, c, d\}$,

P includes : $S \rightarrow S_1S_2$, $S_1 \rightarrow aS_1b \mid ab$, and $S_2 \rightarrow cS_2dd \mid cdd$

$L = L_1L_2 = \{a^m b^n : m, n \geq 1\}\ \{c^n d^{2n} : n \geq 1\}$.

Theorem 5.8.3 : If L is a CFL generated by grammar $G = (V, T, P, S)$, then Kleene closure of L denoted by L^* is also a CFL.

Proof : Let grammar $G' = (V, T, P', S')$ generates L^*. We define G' based on given grammar G

$L^* = \{\in, L, LL, LLL, ...\}$, since L^* includes null string, so G' has production : $S' \rightarrow \in$ and from other productions, G' has to generate multiples of L. So, we have two recursive S' - productions : $S' \rightarrow SS' \mid S'S$, where S is the starting symbol of G

So, $P' = \{S' \rightarrow \in \mid SS' \mid S'S\} \cup \{S$ – productions of grammar $G\}$

Since, all productions of G' are in context-free form, so G' is a CFG.

Language generated by G' :

$L(G') = \{\in, L, LL, LLL, ...\} = L^*$

Thus, statement of theorem is proved.

Example : Consider the CFGs $S \rightarrow aSa \mid aa$, which generates $L = \{a^{2n} : n \geq 1\}$. Construct a grammar, which generates L^*.

Solution :

Let $G' = (V, T, P', S')$ generates L^*. We define the productions of G' as follows :

$S' \rightarrow \in \mid SS' \mid S'S$, where $S \rightarrow aSa \mid aa$

Language generated by G' :

$S' \Rightarrow \in$

Hence, $\in$ is in $L(G')$.

$S' \Rightarrow S'S$ (Using $S' \rightarrow S'S$)

$\Rightarrow S'SS$ (Using S' → S'S)

.....

.....

$\Rightarrow S'SS \ldots n$ times (Using $S' \rightarrow S'S$ n times)

$\Rightarrow \in SS \ldots n$ times (Using $S' \rightarrow \in$)

$\Rightarrow SS \ldots n$ times

$\Rightarrow LL \ldots n$ times (Since, G generates language L and S is the starting symbol of G.)

$= L^+$

So, $L(G') = \{\in\} \cup L^+ = L^*$

Theorem 5.8. 4 : If L_1 and L_2 are two CFLs, then intersection of L_1 and L_2 denoted by $L_1 \cap L_2$ may or may not be a CFL.

Proof : We will discuss some examples, which prove the theorem.

Example 1 : Consider the CFLs $L_1 = \{a^n b^n c^m : m, n \geq 1\}$ and $L_2 = \{a^m b^n c^n : m, n \geq 1\}$, then intersection of L_1 and L_2 is not a CFL.

Solution:

$L_1 = \{abc, aabbcc, aaabbbccc, \ldots\}$ and $L_2 = \{abc, abbcc, aabbcc, aabbbccc, aaabbbccc, \ldots\}$

So, $L_1 \cap L_2 = \{abc, aabbcc, aaabbbccc, \ldots\}$

$= \{a^n b^n c^n : n \geq 1\}$

Clearly, $L_1 \cap L_2$ is not a CFL.

Example 2 : Consider the CFLs $L_1 = \{a^n b^n : n \geq 1\}$ and $L_2 = \{a^p b^q : p, q \geq 1\}$, then intersection of L_1 and L_2 is a CFL.

Solution :

$L_1 = \{ab, aabb, aaabbb, \ldots\}$ and $L_2 = \{ab, aab, aabb, abbb, aabbb, aaabbb, \ldots\}$

So, $L_1 \cap L_2 = \{ab, aabb, aaabbb, \ldots\} = \{a^k b^k : k \geq 1\}$

Clearly, $L_1 \cap L_2$ is a CFL.

Theorem 5.8. 5 : If L is a CFL over some alphabet T, then complement of L denoted by $T^* - L$ may or may not be a CFL.

Proof :

We will discuss some mathematical identities to prove this theorem. Let us assume that complement of a CFL is also CFL. It means, $\overline{L} = T^* - L$ is CFL.

Let R and S are two CFLs over T, then we know that

$R \cap S = T^* - (\overline{R} \cup \overline{S})$ (De Morgan's law)

Since, we have assumed that complement of CFL is also a CFL, so $\overline{R}$ and $\overline{S}$ are CFLs and hence $P = \overline{R} \cup \overline{S}$ is a CFL (P is union of two CFLs).

So, $R \cap S = T^* - P$

or, $R \cap S = \overline{P}$

Since, P is a CFL, so $\overline{P}$ is a CFL.

Thus, $R \cap S$ is a CFL i.e., intersection of CFLs R and S is a CFL.

But, according to Theorem 5.8.4, $R \cap S$ may or may not be a CFL. So, our assumption about complement of a CFL is not hundred percent correct.

Since, intersection and complement are interchangeable using De Morgan's law, so whatever the truth about intersection we have proved that is also applicable to complement.

Therefore, we conclude that complement of a CFL may or may not be a CFL.

We will discuss some examples, which prove the theorem.

Example 1 :

Consider a CFL L over $T = \{a, b\}$ which contains all the strings that not have the number of a's and b's equal or if number of a's and b's are equal then no two a's or b's are consecutive, then $T^* - L$ is a CFL.

Solution :

L = { All strings over $\{a, b\}$ not having number of a's and b's equal } or {All strings over $\{a, b\}$ which have number of a's and b's equal but no two a's and b's are consecutive}

So, $L = \{\in, aab, baa, aaab, ...\} \cup \{ab, abab, baba, ...\}$

$= \{\in, ab, aab, baa, aaab, baaa, abab, baba, ...\}$

Simply, $L = (a + b)^* - \{a^k b^k : k \geq 2\}$

So, $T^* - L = (a + b)^* - ((a + b)^* - \{a^n b^n : n \geq 2\})$

= {All the words over $\{a, b\}$ having equal number of a's and b's and all a's and b's are consecutive}

$= \{a^k b^k : k \geq 2\}$

Clearly, $T^* - L$ is a CFL.

Example 2 :

Consider a CFL L over $\{a, b, c\}$ having all the strings in which number of a's, number of b's and number of c's are not equal or if number of a's, b's and c's are equal then no two a's, b's and c's are consecutive, then T* - L is not a CFL.

Solution :

$L = \{\in, a, b, c, ab, ba, ac, ca, ...\} \cup \{abc, abcabc, acabcb, ...\}$

$= \{\in, a, b, c, ab, ba, ac, ca, aaa, bbb, ccc, abc, ...\}$

Simply, $L = (a + b + c)^* - \{a^n b^n c^n : n \geq 2\}$

Let $T = \{a, b, c\}$ then

$T^* - L = \{aabbcc, aaabbbccc, ...\}$

= { All the words over $\{a, b, c\}$ having equal number of a's, b's and c's and all a's, b's and c's are consecutive}

$= \{a^n b^n c^n : n \geq 2\}$

Clearly, $T^* - L$ is not a CFL.

REVIEW QUESTIONS

Q1. Define context free grammar.

Answer :

For Answer refer to Topic : 5.1, Page No : 5.1.

Q2. Consider the grammar $G = (V, T, P, S)$ having productions :

$S \rightarrow aSa \mid bSb \mid \in$. Check the productions and find the language generated.

Answer :

For Answer refer to example - 1 , Page No : 5.1.

Q3. Let G = (V, T, P, S) where V = { S , C } , T = { a , b }

P = { S $\rightarrow$ aCa

$C \rightarrow$ aCa | b

} S is the start symbol

What is the language generated by this grammar ?

Answer :

For Answer refer to example - 2 , Page No : 5.2.

Q4. What is the language generated by the grammar

$S \rightarrow 0A \mid \in$

$A \rightarrow 1S$

Answer :

For Answer refer to example - 3 , Page No : 5.2.

Q5. Show that the language $L = \{ a^m b^n \mid m \neq n \}$ is context free.

Answer :

For Answer refer to example - 4 , Page No : 5.3.

Q6. Draw a CFG to generate a language consisting of equal number of a's and b's.

Answer :

For Answer refer to example - 5 , Page No : 5.4.

Q7. Construct CFG for the language L which has all the strings which are all palindromes over $T = \{a, b\}$

Answer :

For Answer refer to example - 6 , Page No : 5.5.

Q8. Obtain a CFG to generate integers .

Answer :

For Answer refer to example - 7 , Page No : 5.6.

Q9. Obtain the grammar to generate the language $L = \{ 0^m 1^m 2^n \mid m \geq 1 \text{ and } n \geq 0 \}$.

Answer :

For Answer refer to example - 8 , Page No : 5.6.

Q10. Obtain a grammar to generate the language $L = \{0^n 1^{n+1} \mid n \geq 0 \}$.

Answer :

For Answer refer to example - 9 , Page No : 5.7.

Q11. Obtain the grammar to generate the language $L = \{ w \mid n_a(w) = n_b(w) \}$

Answer :

For Answer refer to example - 10 , Page No : 5.8.

Q12. Explain about leftmost and right most derivations.

Answer :

For Answer refer to Topic : 5.2 , Page No : 5.9.

Q13. Consider the grammar $S \rightarrow S + S \mid S * S \mid a \mid b$. Find leftmost and rightmost derivations for string $w = a * a + b$.

Answer :

For Answer refer to example - 1 , Page No : 5.9.

Q14. Consider a CFG $S \rightarrow bA \mid aB$, $A \rightarrow aS \mid aAA \mid a$, $B \rightarrow bS \mid aBB \mid b$. Find leftmost and rightmost derivations for $w = aaabbabbba$.

Answer :

For Answer refer to example - 2 , Page No : 5.10.

Q15. Explain derivation tree with an example.

Answer :

For Answer refer to Topic : 5.3 , Page No : 5.11.

Q16. Consider the grammar $S \rightarrow S + S | S * S | a | b$. Construct derivation tree for string $w = a * b + a$.

Answer :

For Answer refer to example - 1 , Page No : 5.12.

Q17. Consider a grammar G having productions $S \rightarrow aAS | a$, $A \rightarrow SbA | SS | ba$.

Show that $S \overset{*}{\Rightarrow} aabbaa$ and construct a derivation tree whose yield is aabbaa.

Answer :

For Answer refer to example - 2 , Page No : 5.13.

Q18. Consider the grammar G whose productions are

$S \rightarrow 0B | 1A$, $A \rightarrow 0 | 0S | 1AA$, $B \rightarrow 1 | 1S | 0BB$. Find

(a) Leftmost and(b) Rightmost derivation for string 00110101, and construct derivation tree also.

Answer :

For Answer refer to example - 3 , Page No : 5.14.

Q19. What is ambiguity in CFGs ? Explain .

Answer :

For Answer refer to Topic : 5.4, Page No : 5.14.

Q20. Consider the CFG $S \rightarrow S + S | S * S | a | b$ and string $w = a * a + b$, and derivations as follows:

Answer :

For Answer refer to example - 1 , Page No : 5.15.

Q21. Show that the following grammars are ambiguous.

(a) $S \rightarrow SS | a | b$

(b) $S \rightarrow A | B | b$, $A \rightarrow aAB | ab$, $B \rightarrow abB | \in$

Answer :

For Answer refer to example - 2 , Page No : 5.15.

Q22. What is left recursion ? Explain procedure to eliminate left recursion.

Answer :

For Answer refer to Topic : 5.4.1.1 , Page No : 5.17.

Q23. Eliminate left recursion from the following grammar

$$E \rightarrow E+T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid id$$

Answer :

For Answer refer to example - 1 , Page No : 5.17.

Q24. Eliminate left recursion from the following grammar

$$S \rightarrow Ab \mid a$$
$$A \rightarrow Ab \mid Sa$$

Answer :

For Answer refer to example - 2 , Page No : 5.18.

Q25. What is left factoring? Explain procedure to eliminate left factoring.

Answer :

For Answer refer to Topic : 5.4.1.2, , Page No : 5.19.

Q26. Consider the grammar $S \rightarrow aSa \mid aa$ and remove the left factoring (if any).

Answer :

For Answer refer to example , Page No : 5.19.

Q27. What is useless symbol ? Explain procedure to removal of useless symbols.

Answer :

For Answer refer to Topic : 5.5.1, Page No : 5.21.

Q28. Eliminate the useless symbols in the grammar

S → aA| bB
A → aA | a
B → bB
D → ab | Ea
E → aC | d

Answer :

For Answer refer to example - 1 , Page No : 5.23.

Q29. Eliminate the useless symbols in the grammar

$$
\begin{array}{lcl}
S & \rightarrow & aA \mid a \mid Bb \mid cC \\
A & \rightarrow & aB \\
B & \rightarrow & a \mid Aa \\
C & \rightarrow & cCD \\
D & \rightarrow & ddd
\end{array}
$$

Answer :

For Answer refer to example - 2 , Page No : 5.25.

Q30. What is $\in$ – production ? Explain procedure to removal of $\in$ – productions.

Answer :

For Answer refer to Topic : 5.5.2 , Page No : 5.26.

Q31. Eliminate all $\in$ - productions from the grammar

$$
\begin{array}{lcl}
S & \rightarrow & ABCa \mid bD \\
A & \rightarrow & BC \mid b \\
B & \rightarrow & b \mid \in \\
C & \rightarrow & c \mid \in \\
D & \rightarrow & d
\end{array}
$$

Answer :

For Answer refer to example - 1 , Page No : 5.28.

Q32. Eliminate all $\in$ - productions from the grammar

$$
\begin{array}{lcl}
S & \rightarrow & BAAB \\
A & \rightarrow & 0A2 \mid 2A0 \mid \in \\
B & \rightarrow & AB \mid 1B \mid \in
\end{array}
$$

Answer :

For Answer refer to example - 2 , Page No : 5.29.

Q33. What is unit production? Explain procedure to elimination of unit productions.

Answer :

For Answer refer to Topic : 5.5.3, Page No : 5.30.

Q34. Eliminate all unit productions from the grammar

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow C|b \\ C &\rightarrow D \\ D &\rightarrow E|bC \\ E &\rightarrow d|Ab \end{aligned}$$

Answer :

For Answer refer to example - 1 , Page No : 5.32.

Q35. Eliminate unit productions from the grammar

$$\begin{aligned} S &\rightarrow A\,0 \mid B \\ B &\rightarrow A \mid 11 \\ A &\rightarrow 0 \mid 12 \mid B \end{aligned}$$

Answer :

For Answer refer to example - 2 , Page No : 5.33.

Q36. State and prove CNF.

Answer :

For Answer refer to Topic : 5.6.1 , Page No : 5.35.

Q37. Consider the grammar

$$\begin{aligned} S &\rightarrow 0A \mid 1\,B \\ A &\rightarrow 0AA \mid 1\,S \mid 1 \\ B &\rightarrow 1\,BB \mid 0\,S \mid 0 \end{aligned}$$

Obtain the grammar in CNF .

Answer :

For Answer refer to example - 1 , Page No : 5.37.

Q38. Find a grammar in CNF equivalent to the grammar :

$$S \rightarrow -S \mid [\,S \uparrow S\,] \mid a \mid b$$

Answer :

For Answer refer to example - 2 , Page No : 5.39.

Q39. State and prove GNF.

Answer :

For Answer refer to Topic : 5.6.2 , Page No : 5.41.

Q40. Consider the CFG $S \rightarrow S+S|S*S|a|b$ and find an equivalent grammar in GNF.

Answer :

For Answer refer to example - 1 , Page No : 5.43.

Q41. Consider the grammar $G = (\{A_1, A_2, A_3\}, \{a, b\}, P, A_1)$, where P consists of following production rules.

$A_1 \rightarrow A_2 A_3, A_2 \rightarrow A_3 A_1 | b, A_3 \rightarrow A_1 A_2 | a$ Convert it into GNF.

Answer :

For Answer refer to example - 2 , Page No : 5.43.

Q42. Find equivalent grammar in GNF.

(a) S → aB | b A, A → aS | b AA | a, B → bS | a BB | b

(b) S → abSb | a | a Ab, A → bS | a AAb

(c) S → AA | 0 , A → SS | 1

For Answer refer to example - 3 , Page No : 5.45.

Q43. State and prove Pumping lemma for CFL's.

Answer :

For Answer refer to Topic : 5.7, Page No : 5.46.

Q44. Consider the language $L = \{a^n b^n c^n : n \geq 1\}$ and prove that L is not CFL.

Answer :

For Answer refer to example - 1 , Page No : 5.50.

Q45. Prove that following languages are not CFL

(a) $L = \{a^p : p \text{ is a prime number}\}$

(b) $L = \{a^n b^m c^n d^m : m, n \geq 1\}$

(c) $L = \{a^n b^n c^m : m \geq n\}$

Answer :

For Answer refer to example - 2 , Page No : 5.50.

Q46. Show that the following language is not context free $L = \{a^{n^2} / n \geq 1\}$.

Answer :

For Answer refer to example - 3 , Page No : 5.52.

Q47. Show that the following language is not context-free $L = \{0^m 1^m 2^n / m \leq n \leq 2m\}$.

Answer :

For Answer refer to example - 4 , Page No : 5.53.

Q48. State ogden's lemma .

Answer :

For Answer refer to Topic : 5.7.2 , Page No : 5.53.

Q49. Prove that $L = \{ a^i b^j c^k | i \neq j, j \neq k \text{ and } i \neq k \}$ is not context free.

Answer :

For Answer refer to example , Page No : 5.54.

Q50. Write and prove closure properties of CFLs.

Answer :

For Answer refer to Topic : 5.8 , Page No : 5.55.

OBJECTIVE TYPE QUESTIONS

1. A context free grammar which generates $\{a^l b^m c^n \mid l+m=n\}$ has its production rules given by
   (a) $S \to aS, S \to aS_1, S_1 \to bS_1, S_1 \to bS_2, S_2 \to bS_3, S_3 \to cS_3 \mid c$
   (b) $S \to cSb, cS \to Sc, bS \to Sb, S \to aS_1 c, S_1 \to aS_1 c, S_1 \to ac$
   (c) $S \to aSc \mid ac \mid bc \mid bS_1 c, S_1 \to bS_1 c / bc$
   (d) $S \to bSc \mid aSc \mid ac \mid a \mid bS_1, S_1 \to S_1 c / c$
2. Let two kinds of grammars be defined as follows.
   (I) $A \to a, A \to BC$ (II) $A \to aa$, where $A, B, C \in V$, $A \in V$ and $\alpha \in V^*$
   (a) I denotes Chomsky Normal form but II does not denote Greibach Normal Form.
   (b) I denotes Chomsky Normal form and II denotes Greibach Normal Form.
   (c) II denotes Chomsky Normal form but I does not denote Greibach Normal Form.
   (d) II denotes Chomsky Normal form and I denotes denote Greibach Normal Form
3. The language $L=\{a^n b^m c^n d^m / n \geq 1, m \geq 1\}$
   (a) abstracts the problem of checking number of formal and external parameters
   (b) is context free (c) is not context free
   (d) (a) and (c)
4. CFG is not closed under
   (a) product (b) complementation (c) Kleene star (d) Union
5. The intersection of a CFL and regular language
   (a) Is always context free (b) Is always regular
   (c) need not be context free d) need not be regular
6. Which of the following languages cannot be produced by a Context Free Grammar?
   (a) $L=\{a^n b^n c^n : n \geq 0\}$ (b) $L=\{ww^R : w \in \{a, b\}^*\}$
   (c) $L=\{a^n b^k : k > n \geq 0\}$ d) $L=\{a^n b^n : n \geq 0\}$
7. Every CFG is ambiguous
   (a) some times false (b) some times true (c) false (d) true
8. Consider the language $L=\{a^p : p \text{ is prime}\}$
   (a) CFL (b) RL (c) both (d) None
9. A grammar G is known to have GNf representation then
   (a) G can't be written left or right linear
   (b) we can write G as left linear or right linear
   (c) G may be written as left linear or right linear (d) None of these.

10. $L=\{ww^R$ where w belongs to $(0,1)^*\}$ is
(a) Context Free (b) Context sensitive (c) both (d) None

11. A Grammar G has productions of type $A \to aB\ A \to Ba\ A \to a$ then G is
(a) CF (b) regular (c) both (d) none

12. Let G be the grammar $S \to aA, A \to Abb \mid b$, sentential forms of G are,
(a) aAb^{2n}, ab^{2n} where, $n \geq 0$ (b) aAb^{2n}, ab^{2n+1} where, $n \geq 0$
(c) aAb^{2n}, ab^{2n+1} where, $n \geq 1$ (d) None of above

13. Which of the following grammars can generate $w = aabbb$
(a) $S \to AB, A \to BB \mid a, B \to AB \mid b$ (b) $S \to AB, A \to aA \mid a, B \to bB \mid b$
(c) Both (d) None

14. A CFL is in CNF if every production is of form $S \to A$ or $S \to C$ where S is in V
(a) A is in Σ^* & B,C are in V^* (b) A is in Σ & B, C are in V
(c) A is in Σ & B, C are in V^* (d) A is in Σ^* & B, C are in V

15. The grammar having production as $A \to B$, where $A \in V, B \in (V \cup \Sigma)^+$, is
(a) Type 3 (b) Type 2 (c) Type 1 (d) Type 0

16. The grammar generated by production rules $S \to aSBc \mid abc, cB \to Bc, aB \to aa$ is
(a) $a^n b^n c^n, n \leq 0$ (b) $a^n b^n c^n, n > 0$
(c) $a^n b^n c^n, n \geq 0$ (d) $a^n b^n c^n\ n > 1$

17. Which of the following languages is context free?
(a) $L=\{a^{n^2} : 1 \leq n\}$ (b) $L=\{a^m b^m c^n : m \leq n \leq 2^m\}$
(c) $L=\{a^m b^n : n=m^2\}$ (d) None of the above

18. Let L be a context free language. Then for every $z \in L$, and some strings u, v, w, x and $y \in L$ Then we can find a natural number n such that
(a) $uv^k wx^k\ y \in L$ for all $k = 0$ (b) with $|z| \geq n$ can be written as uvwxy.
(c) $|vx| \geq 1.\ n \geq 1 vwx|$ (d) all of the above

19. What is the correct sequence of steps to construct an equivalent reduced grammar?
(A) Construct equivalent Grammar G_1 such that each variable derives some terminal string.
(B) Construct equivalent Grammar G' such that each symbol in G' appears in some sentential form.
(C) Eliminate null productions.
(D) Eliminate unit productions.
(a) B A D C (b) A B C D (c) BCDA (d) C D B A

20. Which of the following grammars is ambiguous?

(a) $S \to S+S|S*S|a|b$ (b) $S \to a|abSb|aAb, A \to bS|aAb$

(c) $S \to aB|ab, A \to aAB|a, B \to ABb|b$ (d) all of the above

21. Any string of terminals that can be generated by the following CFG

$S \to XY$

$X \to aX|bX|a$

$Y \to Ya|Yb|a$

(a) Has atleast two a (b) Has no consecutive a or b

(c) Should end in a (d) Has at least one b

22. The grammar $\{S \to as|bS|a|b\}$ is not equivalent to

(a) $(a+b)^*(a+b)$ (b) $(a+b)(a+b)^*$ (c) $(a^*+b)^*$ (d) None of the above

23. Which of the following statements is true?

(i) A regular set accepted by a deterministic finite automaton with n states is accepted to final state by a deterministic pda with n states and one pushdown symbol.

(ii) Every regular set accepted by a finite automaton with n states is accepted by a deterministic pda with one state and n pushdown symbols

(iii) If L is accepted by a deterministic pda A, then it can be shown that L is accepted by a deterministic pda A which never adds more than one symbol at a time.

(a) (i), (ii) and (iii) (b) (i) and (iii) only

(c) (ii) and (iii) only (d) (i) and (ii) only

24. Match the language with the corresponding machine :

Language	Machine
(i) Regular language	(A) Nondeterministic pushdown automaton
(ii) DCFL	(B) Turing machine
(iii) CFL	(C) Deterministic pushdown automaton
(iv) context-sensitive language	(D) (Non) Deterministic finite-state acceptor
(v) Recursive language	(E) Turing machine that halts
(vi) Recursively enumerable language	(F) Linear-bounded automaton

(a) D A C F E B (b) D C A F E B

(c) D C A B E F (d) D C F E A B

25. The grammar that generates $L=\{a^n b^n c^i | n=1, i=0\}$ is,

(a) $S \to aAb|Sc, A \to ab|aAb$ (b) $S \to A|Sc, A \to ab|aAb$

(c) Any one of the two (d) None of the two.

26. The grammar that generates $L=\{wcw^t | w \in \{a, b\}*\}$ is,
(a) $S \to aSa | bSb | c$
(b) $S \to aSa | bSb | aca | bcb$
(c) Any one of the two
(d) None of the two.

27. Let $T = \{0, 1, (,), +, *, \text{phi}, e\}$. We may think of T as set of symbols used regular expressions over alphabet $\{0,1\}$; the only different is that we use e for symbol epsilon, to avoid potential confusion in what follows. The CFG with set of terminals that generates exactly the regular expressions with a alphabet $\{0,1\}$ is :
(a) $S \to S+S | SS | S^* | (s) | 0 | 1 | phi | e$
(b) $S \to S+S | SS^* | S^* | (S) | 0 | 1 | \varphi | e$
(c) $S-S+S | SS | S | (S) | 0 | 1 | phi | e$
(d) none is possible

28. The CFG for the following language :
The set $\{0^n1^n | n>1\}$, that is the set of all strings of one or more 0's followed by an equal number of 1's is :
(a) $S \to 0S1 | 0 |$ epsilon.
(b) $S \to 0S1 | 01 |$ epsilon
(c) $S \to 0S1 | 01$
(d) None of the above

29. Which of the following is true
(a) $S \to aB, B \to Bb | b$ is ambiguous
(b) $S \to AB | aaB, A \to a | Aa, B \to b$ is unambiguous
(c) $S \to aB | ab, A \to aAB | a, B \to ABb | b$ is ambiguous
(d) $S \to aSbS | bSaS | \in$ is unambiguous

30. Choose the correct statements :
(a) some regular languages can't be generated by an CFG
(b) some non regular languages can't be generated by an CFG
(c) any regular language has not an equivalent CFG
(d) all languages can be generated by CFG

ANSWER KEY

1(c)	2(b)	3(d)	4(b)	5(a)	6(b)	7(c)	8(d)	9(c)	10(c)
11(c)	12(b)	13(a)	14(b)	15(b)	16(b)	17(d)	18(c)	19(d)	20(d)
21(a)	22(c)	23(a)	24(b)	25(b)	26(a)	27(a)	28(c)	29(c)	30(b)

6

PUSH DOWN AUTOMATA

After going through this chapter, you should be able to understand :

- Push down automata
- Acceptance by final state and by empty stack
- Equivalence of CFL and PDA
- Interconversion
- Introduction to DCFL and DPDA

6.1 INTRODUCTION

A PDA is an enhancement of finite automata (FA). Finite automata with a stack memory can be viewed as pushdown automata. Addition of stack memory enhances the capability of Pushdown automata as compared to finite automata. The stack memory is potentially infinite and it is a data structure. Its operation is based on last - in - first - out (LIFO). It means, the last object pushed on the stack is popped first for operation. We assume a stack is long enough and linearly arranged. We add or remove objects at the left end.

6.1.1 Model of Pushdown Automata (PDA)

A model of pushdown automata is shown in below figure. It consists of a finite tape, a reading head, which reads from the tape, a stack memory operating in LIFO fashion.

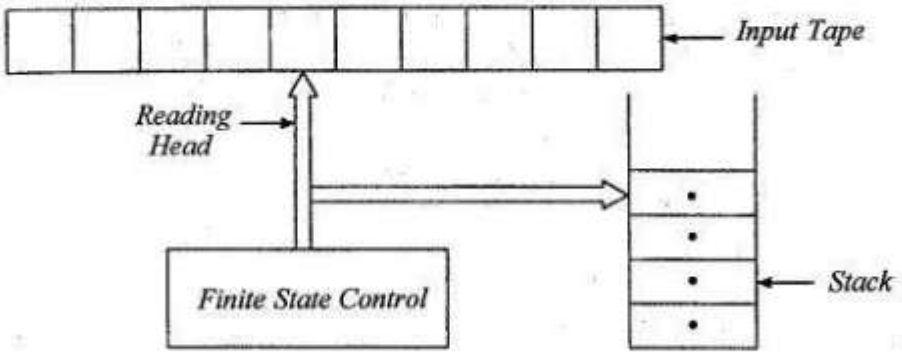


FIGURE : Model of Pushdown Automata

There are two alphabets ; one for input tape and another for stack. The stack alphabet is denoted by Γ and input alphabet is denoted by Σ. PDA reads from both the alphabets ; one symbol from the input and one symbol from the stack.

6.1.2 Mathematical Description of PDA

A pushdown automata is described by 7 - tuple $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$, where

1. Q is finite and nonempty set of states,
2. Σ is input alphabet,
3. Γ is finite and nonempty set of pushdown symbols,
4. δ is the transition function which maps
   From $Q \times (\Sigma \cup \{\in\}) \times \Gamma$ to (**finite subset** of) $Q \times \Gamma^*$,
5. $q_0 \in Q$, is the starting state,
6. $Z_0 \in \Gamma$, is the starting (top most or initial) stack symbol, and
7. $F \subseteq Q$, is the set of final states.

6.1.3 Moves of PDA

The move of PDA means that what are the options to proceed further after reading inputs in some state and writing some string on the stack. As we have discussed earlier that PDA is nondeterministic device having some finite number of choices of moves in each situation.

The **move** will be of two types :

1. In the first type of move, an input symbol is read from the tape, it means, the head is advanced and depending upon the topmost symbol on the stack and present state, PDA has number of choices to proceed further.
2. In the second type of move, the input symbol is not read from the tape, it means, head is not advanced and the topmost symbol of stack is used. The topmost of stack is modified without reading the input symbol. It is also known as an $\in$ - move.

Mathematically first type of move is defined as follows.

$\delta(q,a,Z) = \{(p_1,\alpha_1),(p_2,\alpha_2),\ldots(p_n,\alpha_n)\}$, where for $1 \le i \le n, q, p_i$ are states in Q, $a \in \Sigma$, $Z \in \Gamma$, *and* $\alpha_i \in \Gamma^*$.

PDA reads an input symbol a and one stack symbol Z in present state q and for any value(s) of i, enters state p_i, replaces stack symbol Z by string $\alpha_i \in \Gamma^*$, and head is advanced one cell on the tape. Now, the leftmost symbol of string α_i is assumed as the topmost symbol on the stack.

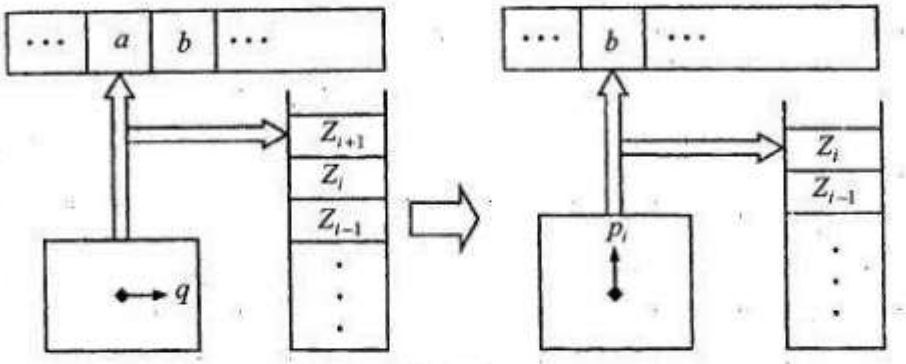
Mathematically second type of move is defined as follows.

$\delta(q,\in,Z) = \{(p_1,\alpha_1),(p_2,\alpha_2),\ldots(p_n,\alpha_n)\}$, where for $1 \le i \le n, q, p_i$ are states in $Q, a \in \Sigma$, $Z \in \Gamma$, *and* $\alpha_i \in \Gamma^*$.

PDA does not read input symbol but it reads stack symbol Z in present state q and for any value(s) of i, enters state p_i, replaces stack symbol Z by string $\alpha_i \in \Gamma^*$, and head is not advanced on the tape. Now, the leftmost symbol of string α_i is assumed as the topmost symbol on the stack.

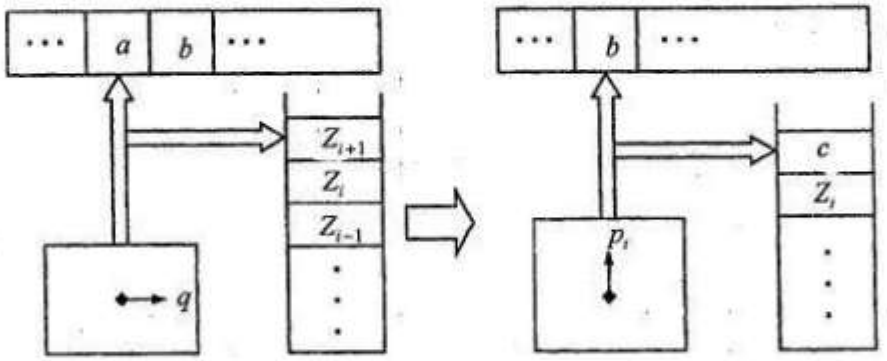
The **string** α_i be any one of the following :

1. $\alpha_i = \in$ in this case the topmost stack symbol Z_{i+1} is erased and second topmost symbol becomes the topmost symbol in the next move. It is shown in figure (a).



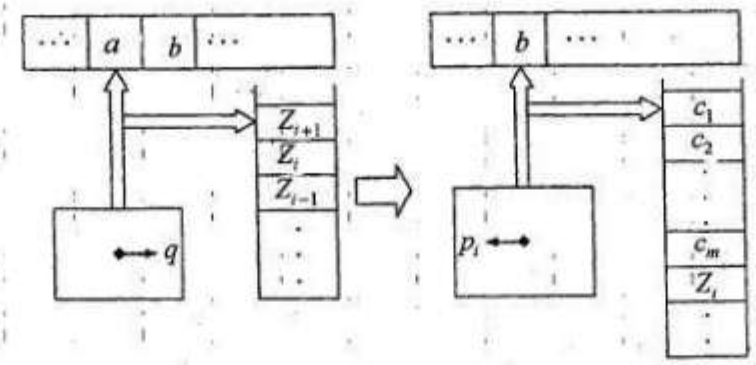
FIGURE(a): Move of PDA

2. $\alpha_i = c, c \in \Gamma$, in this case the topmost stack symbol Z_{i+1} is replaced by symbol c. It is shown in figure(b)



FIGURE(b): Move of PDA

3. $\alpha_i = c_1 c_2 \ldots c_m$, in this case the topmost stack symbol Z_{i+1} is replaced by string $c_1 c_2 \ldots c_m$. It is shown in figure(c).



FIGURE(c): Move of PDA

6.1.4 Instantaneous Description (ID) of PDA

Let PDA $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$, then its configuration at a given instant can be defined by instantaneous description (ID). An ID includes state, remaining input string, and remaining stack string (symbols). So, an ID is (q,x,α), where $q \in Q, x \in \Sigma^*, \alpha \in \Gamma^*$.

The relation between two consecutive IDs is represented by the sign $\vdash$.

We say $(q,ax,Z\beta) \vdash_M (p,x,\alpha\beta)$ if $\delta(q,a,Z)$ contains (p,α), where $Z,\beta,\alpha \in \Gamma^*$, a may be null or $a \in \Sigma$, $p,q \in Q$ for M

The reflexive and transitive closure of the relation $\vdash_M$ is denoted by $\vdash_M^*$

Properties :

1. If $(q,x,\alpha) \vdash_M^* (p,\in,\alpha)$, where $\alpha \in \Gamma^*, x \in \Sigma^*$, and $p,q \in Q$, then for all $y \in \Sigma^*$. $(q,xy,\alpha) \vdash_M^* (p,y,\alpha)$,
2. If $(q,xy,\alpha) \vdash_M^* (p,y,\alpha)$, where $\alpha \in \Gamma^*, x,y \in \Sigma^*$, and $p,q \in Q$, then $(q,x,\alpha) \vdash_M^* (p,\in,\alpha)$, and
3. If $(q,x,\alpha) \vdash_M^* (p,\in,\beta)$, where $\alpha,\beta \in \Gamma^*, x \in \Sigma^*$, and $p,q \in Q$, then $(q,x,\alpha\gamma) \vdash_M^* (p,\in,\beta\gamma)$, where $\gamma \in \Gamma^*$

6.1.5 Acceptance by PDA

Let M be a PDA, the accepted language is represented by N(M). We defined the acceptance by PDA in two ways.

1. Let $M = (Q,\Sigma,\Gamma,\delta, q_0, Z_0,F)$, then N(M) is accepted by final state such that

   $N(M)=\{w:(q_0,w,Z_0)\overset{*}{\underset{M}{\vdash}}(q_f,\in,\beta)$, where $q \in Q$, $w \in \Sigma^*, Z_0, \beta \in \Gamma^*$, and $q_f \in F\}$

   It is similar to the acceptance by FA discussed earlier. We define some final states and the accepted language *N(M)* is the set of all input strings for which some choice of moves leads to some final state.

2. Let $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,\phi)$, then *N(M)* is accepted by empty stack or null stack such that $N(M) = \{w:(q_0,w,Z_0)\overset{*}{\underset{M}{\vdash}}(p,\in,\in)$, where $p \in Q, w \in \Sigma^*\}$

   The language *N(M)* is the set of all input strings for which some sequence of moves causes the PDA to empty its stack.

Note : If acceptance is defined by empty stack then there is no meaning of final state and it is represented by ϕ.

Example : consider a PDA $M = (\{q_0,q_1,q_2\},\{a,c\},\{a,Z_0\},\delta,q_0,Z_0,\{q_2\})$ shown in below figure. Check the acceptability of string aacaa.

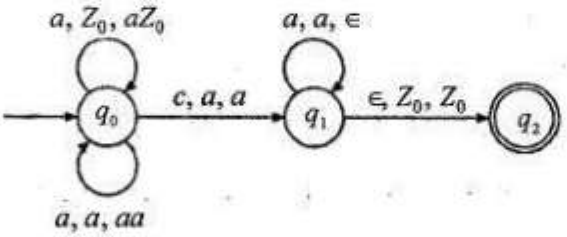


FIGURE : PDA accepting $\{a^n ca^n : n \geq 1\}$

Note : *Edges are labeled with Input symbol, stack symbol, written symbol on the stack.*

Solution :

The transition function δ is defined as follows :

$\delta(q_0,a,Z_0) = \{(q_0,aZ_0)\}$,

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

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

$\delta(q_1,a,a) = \{(q_1,\in)\}$, and

$\delta(q_1,\in,Z_0) = \{(q_2,Z_0)\}$

Following moves are carried out in order to check acceptability of string $aacaa$:

$$(q_0,aacaa,Z_0) \vdash (q_0,acaa,aZ_0)$$

$$\vdash (q_0,caa,aaZ_0)$$

$$\vdash (q_1,aa,aaZ_0)$$

$$\vdash (q_1,a,aZ_0)$$

$$\vdash (q_1,\in,Z_0)$$

$$\vdash (q_2,\in,Z_0)$$

Hence, $(q_0,aacaa,Z_0) \vdash_M^* (q_2,\in,Z_0)$.

Therefore, the string **aacaa** is accepted by M.

6. 2 CONSTRUCTION OF PDA

In this section, we shall see how PDA's can be constructed.

Example 1 : Obtain a PDA to accept the language $L(M) = \{ wCw^R \mid w \in (a+b)^* \}$ where W^R is reverse of W .

Solution:

It is clear from the language $L(M) = \{ wCw^R \}$ that if $w = abb$

then reverse of w denoted by W^R will be $W^R = bba$ and the language L will be wCw^R
i. e., abbCbba which is a string of palindrome.

So, we have to construct a PDA which accepts a palindrome consisting of a's and b's with the symbol C in the middle.

General Procedure :

To check for the palindrome, let us push all scanned symbols onto the stack till we encounter the letter C. Once we pass the middle string, if the string is a palindrome, for each scanned input symbol, there should be a corresponding symbol (same as input symbol) on the stack. Finally, if there is no input and stack is empty, we say that the given string is a palindrome.

Step 1 : Input symbols can be a or b .

Let q_0 be the initial state and Z_0 be the initial symbol on the stack. In state q_0 and when top of the stack is Z_0, whether the input symbol is a or b push it on to the stack, and remain in q_0. The transitions defined for this can be of the form

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

Once the first scanned input symbol is pushed on to the stack, the stack may contain either a or b. Now, in state q_0, the input symbol can be either a or b. Note that irrespective of what is the input or what is there on the stack, we have to keep pushing all the symbols on to the stack, till we encounter C. So, the transitions defined for this can be of the form

$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_0, ba)$$
$$\delta(q_0, a, b) = (q_0, ab)$$
$$\delta(q_0, b, b) = (q_0, bb)$$

Step 2 : Input symbol is C

Now, if the next input symbol is C, the top of the stack may be a or b. Another possibility is that, in state q_0, the first symbol itself can be C. In this case w is null string and Z_0 will be on the stack. In all these cases, the input symbol is C i. e., the symbol which is present in the middle of the string. So, change the state to q_1 and do not alter the contents of the stack. The transitions defined for this can be of the form

$$\delta(q_0, C, Z_0) = (q_1, Z_0)$$
$$\delta(q_0, C, a) = (q_1, a)$$
$$\delta(q_0, C, b) = (q_1, b)$$

Now, we have passed the center of the string.

Step 3 : Input symbols can be a or b .

To be a palindrome, for each input symbol there should be a corresponding symbol (same as input symbol) on the stack. So, whenever the input symbol is same as symbol on the stack, remain in state q_1 and delete that symbol from the stack and repeat the process. The transitions defined for this can be of the form

$$\delta(q_1, a, a) = (q_1, \in)$$
$$\delta(q_1, b, b) = (q_1, \in)$$

Step 4 : Finally, in state q_1 , if the string is a palindrome, there is no input symbol to be scanned and the stack should be empty i. e., the stack should contain Z_0. Now, change the state to q_2 and do not alter the contents of the stack. The transition for this can be of the form

$$\delta(q_1, \in, Z_0) = (q_2, Z_0)$$

So, the PDA M to accept the language $L(M) = \{wCw^R \mid w \in (a,b)^*\}$ is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

Where $Q = \{q_0, q_1, q_2\}$; $\Sigma = \{a, b, C\}$; $\Gamma = \{a, b, Z_0\}$

δ : is shown below,

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_0, ba)$$
$$\delta(q_0, a, b) = (q_0, ab)$$
$$\delta(q_0, b, b) = (q_0, bb)$$
$$\delta(q_0, C, Z_0) = (q_1, Z_0)$$
$$\delta(q_0, C, a) = (q_1, a)$$
$$\delta(q_0, C, b) = (q_1, b)$$
$$\delta(q_1, a, a) = (q_1, \in)$$
$$\delta(q_1, b, b) = (q_1, \in)$$
$$\delta(q_1, \in, Z_0) = (q_2, Z_0)$$

$q_0 \in Q$ is the start state of machine.

$Z_0 \in \Gamma$ is the initial symbol on the stack.

$F = \{q_2\}$ is the final state.

To accept the string :

The sequence of moves made by the PDA for the string **aabCbaa** is shown below.

Initial ID

$(q_0, aabCbaa, Z_0) \vdash (q_0, abCbaa, aZ_0)$

$\vdash (q_0, bCbaa, aaZ_0)$

$\vdash (q_0, Cbaa, baaZ_0)$

$\vdash (q_1, baa, baaZ_0)$

$\vdash (q_1, aa, aaZ_0)$

$\vdash (q_1, a, aZ_0)$

$\vdash (q_1, \in, Z_0)$

$\vdash (q_2, \in, Z_0)$

(Final Configuration)

Since q_2 is the final state and input string is $\in$ in the final configuration, the string **aabCbaa** is accepted by the PDA.

To reject the string :

The sequence of moves made by the PDA for the string **aabCbab** is shown below.

Initial ID

$(q_0, aabCbab, Z_0) \vdash (q_0, abCbab, aZ_0)$

$\vdash (q_0, bCbab, aaZ_0)$

$\vdash (q_0, Cbab, baaZ_0)$

$\vdash (q_1, bab, baaZ_0)$

$\vdash (q_1, ab, aaZ_0)$

$\vdash (q_1, b, aZ_0)$

(Final Configuration)

Since the transition $\delta(q_1, b, a)$ is not defined, the string **aabCbab** is not a palindrome and the machine halts and the string is rejected by the PDA.

Example 2 : Obtain a PDA to accept the language $L = \{ a^n b^n \mid n \geq 1 \}$ by a final state.

Solution :

The machine should accept n number of a's followed by n number of b's.

General Procedure :

Since n number of a's should be followed by n number of b's, let us push all the symbols on to the stack as long as the scanned input symbol is a. Once we encounter b's, we should see that for each b in the input, there should be corresponding a on the stack. When the input pointer reaches the end of the string, the stack should be empty. If stack is empty, it indicates that the string scanned has n number of a's followed by n number of b's.

Step 1 : Let q_0 be the start state and Z_0 be the initial symbol on the stack. As long as the next input symbol to be scanned is a, irrespective of what is there on the stack, keep pushing all the symbols onto the stack and remain in q_0. The transitions defined for this can be of the form

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$

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

Step 2 : In state q_0, if the next input symbol to be scanned is b and if the top of the stack is a, change the state to q_1 and delete one b from the stack. The transition for this can be of the form

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

Step 3 : Once the machine is in state q_1, the rest of the symbols to be scanned will be only b's and for each b there should be corresponding symbol a on the stack. So, as the scanned input symbol is b and if there is a matching a on the stack, remain in q_1 and delete the corresponding a from the stack. The transitions defined for this can be of the form

$$\delta(q_1, b, a) = (q_1, \in)$$

Step 4 : In state q_1, if the next input symbol to be scanned is $\in$ and if the top of the stack is Z_0, (it means that for each b in the input there exists corresponding a on the stack) change the state to q_2 which is an accepting state. The transition defined for this can be of the form

$$\delta(q_1, \in, Z_0) = (q_2, \in)$$

So, the PDA to accept the language $L = \{ a^n b^n \mid n \geq 1 \}$ is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

where $Q = \{q_0, q_1, q_2\}$; $\Sigma = \{a, b\}$; $\Gamma = \{a, Z_0\}$

δ : is shown below.

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$

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

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

$$\delta(q_1, b, a) = (q_1, \in)$$

$$\delta(q_1, \in, Z_0) = (q_2, \in)$$

$q_0 \in Q$ is the start state of machine

$Z_0 \in \Gamma$ is the initial symbol on the stack

$F = \{q_2\}$ is the final state.

To accept the string :

The sequence of moves made by the PDA for the string **aaabbb** is shown below.

Initial ID

$(q_0, aaabbb, Z_0)$ $\vdash$ $(q_0, aabbb, aZ_0)$

$\vdash$ $(q_0, abbb, aaZ_0)$

$\vdash$ $(q_0, bbb, aaaZ_0)$

$\vdash$ (q_1, bb, aaZ_0)

$\vdash$ (q_1, b, aZ_0)

$\vdash$ $(q_1, \in, Z_0)$

$\vdash$ $(q_2, \in, Z_0)$

(Final Configuration)

Since q_2 is the final state and input string is $\in$ in the final configuration, the string **aaabbb** is accepted by the PDA.

To reject the string :

The sequence of moves made by the PDA for the string **aabbb** is shown below.

Initial ID

$(q_0, aabbb, Z_0)$ $\vdash$ $(q_0, abbb, aZ_0)$

$\vdash$ (q_0, bbb, aaZ_0)

$\vdash$ (q_1, bb, aZ_0)

$\vdash$ (q_1, b, Z_0)

(Final Configuration)

Since the transition $\delta(q_1, b, Z_0)$ is not defined, the string **aabbb** is rejected by the PDA.

Example 3 : Obtain a PDA to accept the language $L(M) = \{ w | w \in (a+b)^* \text{ and } n_a(w) = n_b(w) \}$.

solution :

The language accepted by the machine should consist strings of a's and b's of any length. Only restriction is that number of a's in string w should be equal to number of b's. The order of a's and b's is irrelevant. For example aaabbb, ababab, aababb etc. are all the strings in the language L(M).

General Procedure :

The first scanned input symbol is pushed on to the stack. From this point onwards, if the scanned symbol is same as the symbol on to the stack, push the current input symbol on to the stack. If the input symbol is different from the symbol on the top of the stack, pop one symbol from the stack and repeat the process. Finally, when end of string is encountered, if the stack is empty, the string w has equal number of a's and b's, otherwise number of a's and b's are different.

Step 1 : Let q_0 be the start state and Z_0 be the initial symbol on the stack. When the machine is in state q_0 and when top of the stack contains Z_0, scan the input symbol (either a or b) and push it on to the stack. The transitions defined for this can be of the form.

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

Step 2 : Once the first input symbol is pushed on to the stack, the top of stack may contain either a or b and the next input symbol to be scanned may be a or b. If the input symbol is same as the symbol on top of the stack, push the current input symbol on to the stack and remain in state q_0 only. Otherwise, pop an element from the stack. The transitions defined for this can be of the form

$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, b) = (q_0, bb)$$
$$\delta(q_0, a, b) = (q_0, \in)$$
$$\delta(q_0, b, a) = (q_0, \in)$$

Step 3 : In state q_0, if the next symbol to be scanned is $\in$ (empty string) and top of the stack is Z_0, it means that for each symbol a there exists a corresponding b and for each symbol b, there exists a symbol a. So, the string w consists of equal number of a's and b's and change the state to q_1. The transition defined for this can be of the form

$$\delta(q_0, \in, Z_0) = (q_1, Z_0)$$

So, the PDA to accept the language $L=\{w|n_a(w)=n_b(w)\}$ is given by $M=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$

where $Q=\{q_0, q_1\}$; $\Sigma=\{a, b\}$; $\Gamma=\{a, b, Z_0\}$

δ : is shown below

$$\delta(q_0,a,Z_0)=(q_0,aZ_0)$$
$$\delta(q_0,b,Z_0)=(q_0,bZ_0)$$
$$\delta(q_0,a,a)=(q_0,aa)$$
$$\delta(q_0,b,b)=(q_0,bb)$$
$$\delta(q_0,a,b)=(q_0,\in)$$
$$\delta(q_0,b,a)=(q_0,\in)$$
$$\delta(q_0,\in,Z_0)=(q_1,Z_0)$$

$q_0 \in Q$ is the start state of machine.

$Z_0 \in \Gamma$ is the initial symbol on the stack.

$F=\{q_1\}$ is the final state.

To accept the string :

The sequence of moves made by the PDA for the string **abbbaa** is shown below.

Initial ID

$(q_0, abbbaa, Z_0) \vdash (q_0, bbbaa, aZ_0)$

$\vdash (q_0, bbaa, Z_0)$

$\vdash (q_0, baa, bZ_0)$

$\vdash (q_0, aa, bbZ_0)$

$\vdash (q_0, a, bZ_0)$

$\vdash (q_0, \in, Z_0)$

$\vdash (q_1, \in, Z_0)$

(Final Configuration)

Since q_1 is the final state and input string is $\in$ in the final configuration, the string **abbbaa** is accepted by the PDA.

To reject the string :

The sequence of moves made by the PDA for the string aabbb is shown below.

Initial ID

$(q_0, aabbb, Z_0) \vdash (q_0, abbb, aZ_0)$

$\vdash (q_0, bbb, aaZ_0)$

$\vdash (q_0, bb, aZ_0)$

$\vdash \quad (q_0, b, Z_0)$

$\vdash \quad (q_0, \in, bZ_0)$

(Final Configuration)

Since the transition $\delta(q_0, \in b)$ is not defined, the string **aabbb** is rejected by the PDA.

Note : To accept the language by an empty stack, the final state is irrelevant, the next input symbol to be scanned should be $\in$, and stack should to be empty. Even Z_0 should not be then on the stack. So, the PDA to accept equal number of a's and b's using an empty stack, change only the last transition in example 3. The last transition

$$\delta(q_0, \in, Z_0) = (q_1, Z_0)$$

can be changed as

$$\delta(q_0, \in, Z_0) = (q_1, \in)$$

So, the PDA to accept the language $L = \{w | n_a(w) = n_b(w)\}$ by an empty stack is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \in)$$

where $Q = \{q_0, q_1\}$; $\Sigma = \{a, b\}$; $\Gamma = \{a, b, Z_0\}$

δ : is shown below $\delta(q_0, a, Z_0) = (q_0, aZ_0)$

$\delta(q_0, b, Z_0) = (q_0, bZ_0)$

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

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

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

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

$\delta(q_0, \in, Z_0) = (q_1, \in)$

$q_0 \in Q$ is the start state of machine.

$Z_0 \in \Gamma$ is the initial symbol on the stack.

$F = \{\phi\}$.

Note that PDA is accepted by an empty stack.

The sequence of moves made by the PDA for the string **aabbab** by an empty stack is shown below.

Initial ID

$(q_0, aabbab, Z_0) \vdash (q_0, abbab, aZ_0)$

$\vdash (q_0, bbab, aaZ_0)$

$\vdash (q_0, bab, aZ_0)$

$\vdash (q_0, ab, Z_0)$

$\vdash (q_0, b, aZ_0)$

$\vdash (q_0, \in, Z_0)$

$\vdash (q_1, \in, \in)$

(Final Configuration)

Since the next input symbol to be scanned is $\in$ and the stack is empty, the string **aabbab** is accepted by an empty stack.

Note : q_1 is not the final state. Stack is empty .

Example 4 : Obtain a PDA to accept a string of balanced parentheses. The parentheses to be considered are (,) , [,] .

Solution :

Note1 : Some of the valid strings are : [( ) ( ) ([ ])] , $\in$, [ ] [ ] ( ) and invalid string are : [ ) ( ) [ ] ,) (, [ )

Note 2 : $\in$ (null string) is valid

Note 3 : Left parentheses can either be '(' or '[' and right parentheses can either be ')' or ']'.

Step 1 : Let q_0 be the start state and Z_0 be the initial symbol on the stack. The state q_0 itself is the final state accepting $\in$ (an empty string).

Step 2 : In the state q_0 , if the first scanned parentheses is '(' or '[', push the scanned symbol on to the stack and change the state to q_1 . The transition defined for this can be of the form

$$\delta(q_0, (, Z_0) = (q_1, (Z_0)$$

$$\delta(q_0, [, Z_0) = (q_1, [Z_0)$$

Step 3 : If at least one parentheses either '(' or '[' is present on the stack and if the scanned symbol is left parentheses, then push the left parentheses on to the stack. The transitions defined for this can be of the form

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

$$\delta(q_1, (, [) = (q_1, ([)$$

$$\delta(q_1, [, () = (q_1, [()$$

$$\delta(q_1, [, [) = (q_1, [[)$$

Step 4 : If the scanned symbol is ')' and if the top of the stack is '(' pop an element from the stack. Similarly, if the scanned symbol is ']' and if the top of the stack is '[' pop an element from the stack. The transitions defined for this can be of the form

$$\delta(q_1,) , ( ) = (q_1, \in)$$

$$\delta(q_1,] , [ ) = (q_1, \in)$$

Step 5 : When top of the stack is Z_0, it indicates that so far all the parentheses have been matched. At this point, on $\in$ - transition, the PDA enters into state q_0 and all the steps from step1 are repeated . The transition for this can be of the form

$$\delta(q_1, \in, Z_0) = (q_0, Z_0)$$

So, the PDA to accept the language consisting of balanced parentheses is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where $Q = \{ q_0, q_1 \}$; $\Sigma = \{ (,), [,] \}$; $\Gamma = \{ (, [, Z_0 \}$

δ : is shown below .

$$\delta(q_0, (, Z_0) = (q_1, (Z_0)$$

$$\delta(q_0, [, Z_0) = (q_1, [Z_0)$$

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

$$\delta(q_1, (, [) = (q_1, ([)$$

$$\delta(q_1, [, () = (q_1, [()$$

$$\delta(q_1, [, [) = (q_1, [[)$$

$$\delta(q_1,), () = (q_1, \in)$$

$$\delta(q_1,], [) = (q_1, \in)$$

$$\delta(q_1, \in, Z_0) = (q_0, Z_0)$$

$q_0 \in Q$ is the start state of machine.

$Z_0 \in \Gamma$ is the initial symbol on the stack.

$F = \{q_0\}$.

Note that even $\in$ is accepted by PDA and is valid.

To accept the string :

The sequence of moves made by the PDA for the string [( ) ( ) ([ ])] is shown below.

Initial ID

$$(q_0, [( ) ( ) ([ ])], Z_0) \vdash (q_1, ( ) ( ) ([ ])], [Z_0)$$

$$\vdash (q_1,) ( ) ([ ])], ([Z_0)$$

$$\vdash (q_1, ( ) ([ ])], [Z_0)$$

$\vdash (q_1, \quad)([\,])],([Z_0)$
$\vdash (q_1, \quad ([\,])],[Z_0)$
$\vdash (q_1, \quad [\,])],([Z_0)$
$\vdash (q_1, \quad \,])],[([Z_0)$
$\vdash (q_1, \quad)],([Z_0)$
$\vdash (q_1, \quad \,],[Z_0)$
$\vdash (q_1, \in, Z_0)$
$\vdash (q_0, \in, Z_0)$
(Final Configuration)

Since the next input symbol to be scanned is $\in$ and the stack is empty, the string [()()([])] is accepted by the PDA.

Example 5 : Obtain a PDA to accept the language $L = \{ w \mid w \in (a, b)^* \text{ and } n_a(w) > n_b(w) \}$.

solution :

Note : The solution is similar to that of the problem discussed in example 3 in which we are accepting strings of a's and b's of equal numbers. When we encounter end of the input i. e., $\in$ and top of the stack is Z_0, it has equal number of a's and b's. But, what we want is a machine to accept more number of a's than b's. For this, only change to be made is that when we encounter $\in$ (i. e., end of the input), if top of the stack contains at least one a, then change the final state to q_1 and do not alter the contents of the stack. The transition defined remains same as problem shown in example 3, except the last transition. The last transition is of the form

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

So, the PDA to accept the language $L = \{w \mid n_a(w) > n_b(w)\}$
is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
where $Q = \{q_0, q_1\}$; $\Sigma = \{a, b\}$; $\Gamma = \{a, b, Z_0\}$
δ : is shown below.

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, b) = (q_0, bb)$

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

$q_0 \in Q$ is the start state of machine ; $Z_0 \in \Gamma$ is the initial symbol on the stack.

$F = \{q_1\}$ is the final state.

Note : On similar lines we can find a PDA to accept the language

$$L=\{w \mid w\in(a,b)^* \text{ and } n_a(w)< n_b(w)\}$$

i. e., strings of a's and b's where number of b's are more than number of a's . To achieve this only change to be made in the above machine is change the final transition.

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

to

$$\delta(q_0, \in, b) = (q_1, b)$$

So, the PDA to accept the language $L = \{w|w\in(a,b)^* \text{and } n_a(w)< n_b(w)\}$ is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

where $Q = \{q_0, q_1\}$; $\Sigma = \{a, b\}$; $\Gamma = \{a, b, Z_0\}$;

δ : is shown below.

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, b) = (q_0, bb)$$
$$\delta(q_0, a, b) = (q_0, \in)$$
$$\delta(q_0, b, a) = (q_0, \in)$$
$$\delta(q_0, \in, b) = (q_1, b)$$

$q_0 \in Q$ is the start state of machine ; $Z_0 \in \Gamma$ is the initial symbol on the stack

$F = \{q_1\}$ is the final state.

Example 6 : Obtain a PDA to accept the language $L = \{a^n b^{2n} \mid n \geq 1\}$.

solution :

The machine should accept n number of a's followed by 2n number of b's.

General Procedure :

Since n number of a's should be followed by 2n number of b's, for each a in the input, push two a's on to the stack. Once we encounter b's, we should see that for each b in the input, there should be corresponding a on the stack. When the input pointer reaches the end of the string, the stack should be empty. If stack is empty, it indicates that the string scanned has n number of a's followed by 2n number of b's .

Step 1 : Let q_0 be the start state and Z_0 be the initial symbol on the stack. For each scanned input symbol a, push two a's on to the stack. The transitions defined for this can be of the form

$$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$$

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

Step 2 : In state q_0, if the next input symbol to be scanned is b and if the top of the stack is a, change the state to q_1 and delete one b from the stack. The transition for this can be of the form

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

Step 3 : Once the machine is in state q_1, the rest of the symbols to be scanned will be only b's and for each b there should be corresponding symbol a on the stack. So, as the scanned input symbol is b and if there is a matching a on the stack, remain in q_1 and delete the corresponding a from the stack. The transitions defined for this can be of the form

$$\delta(q_1, b, a) = (q_1, \in)$$

Step 4 : In state q_1, if the next input symbol to be scanned is $\in$ and if the top of the stack is Z_0, (it means that for each b in the input there exists corresponding a on the stack) change the state to q_2 which is an accepting state. The transition defined for this can be of the form

$$\delta(q_1, \in, Z_0) = (q_2, \in)$$

So, the PDA to accept the language $L = \{ a^n b^{2n} \mid n \geq 1 \}$

is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

where $Q = \{ q_0, q_1, q_2 \}$; $\Sigma = \{a, b\}$; $\Gamma = \{ a, Z_0 \}$

δ : is shown below.

$$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$$

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

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

$$\delta(q_1, b, a) = (q_1, \in)$$

$$\delta(q_1, \in, Z_0) = (q_2, \in)$$

$q_0 \in Q$ is the start state of machine ; $Z_0 \in \Gamma$ is the initial symbol on the stack.

$F = \{q_2\}$ is the final state.

To accept the string :

The sequence of moves made by the PDA for the string **aabbbb** is shown below.

Initial ID

$(q_0,\ aabbbb,\ Z_0)$ $\vdash$ $(q_0,\ abbbb,\ aaZ_0)$

$\vdash$ $(q_0,\ bbbb,\ aaaaZ_0)$

$\vdash$ $(q_0,\ bbb,\ aaaZ_0)$

$\vdash$ $(q_1,\ bb,\ aaZ_0)$

$\vdash$ $(q_1,\ b,\ aZ_0)$

$\vdash$ $(q_1,\ \in,\ Z_0)$

$\vdash$ $(q_2,\ \in,\ Z_0)$

(Final Configuration)

Since q_2 is the final state and input string is $\in$ in the final configuration, the string **aabbbb** is accepted by the PDA.

To reject the string :

The sequence of moves made by the PDA for the string **aabbb** is shown below.

Initial ID

$(q_0,\ aabbb,\ Z_0)$ $\vdash$ $(q_0,\ abbb,\ aaZ_0)$

$\vdash$ $(q_0,\ bbb,\ aaaaZ_0)$

$\vdash$ $(q_0,\ bb,\ aaaZ_0)$

$\vdash$ $(q_0,\ b,\ aaZ_0)$

$\vdash$ $(q_0,\ \in,\ aZ_0)$

(Final Configuration)

Since the transition $\delta(q_0,\ \in,\ a)$ is not defined, the string **aabbb** is rejected by the PDA.

Example 7 : Obtain a PDA to accept the language $L = \{ ww^R \mid w \in (a+b)^* \}$.

solution :

It is clear from the language $L(M) = \{ ww^R \}$ that if $w = abb$

then reverse of w denoted by w^R will be $w^R = bba$

and the language L will be ww^R i. e., abbbba which is a string of palindrome.

So, we have to construct a PDA which accepts a palindrome consisting of a's and b's. This problem is similar to the problem discussed in example 1. Only difference is that in example 1, an extra symbol C acts as a pointer to the middle string. But , here there is no way to find the mid point for the string.

General Procedure :

To check for the palindrome, let us push all scanned symbols onto the stack till we encounter the mid point (Remember that there is no way to find the midpoint). Once we pass the middle string, to be a palindrome, for each scanned input symbol , there should be a corresponding symbol (same as input symbol) on the stack. Finally, if there is no input and stack is empty, we say that the given string is a palindrome.

Step 1 : Let q_0 be the initial state and Z_0 be the initial symbol on the stack. In state q_0 and when top of the stack is Z_0, whether the input symbol is a or b push it on to the stack, and remain in q_0. The transitions defined for this can be of the form

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

Once the first scanned input symbol is pushed on to the stack, the stack may contain either a or b. Now, in state q_0, the input symbol can be either a or b. Note that irrespective of what is the input or what is there on the stack, we have to keep pushing all the symbols on to the stack, till we encounter midpoint (But, there is no way to find mid point. We continue this process till we encounter mid point through our common sense).

So, the transitions defined for this can be of the form

$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_0, ba)$$
$$\delta(q_0, a, b) = (q_0, ab)$$
$$\delta(q_0, b, b) = (q_0, bb)$$

Step 2 : Now, once we reach the midpoint, the top of the stack may be a or b. To be a palindrome, for each input symbol there should be a corresponding symbol (same as input symbol) on the stack. So, whenever the input symbol is same as symbol on the stack, change the state to q_1 and delete that symbol from the stack. The transitions defined for this can be of the form

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

Step 3 : Now, once we are in state q_1, it means that we have passed the mid point. Now, the top of the stack may be a or b. To be a palindrome, for each input symbol there should be a corresponding symbol (same as input symbol) on the stack. So, whenever the input symbol is same as symbol on the stack, remain in state q_1 and delete that symbol from the stack. The transitions defined for this can be of the form

$$\delta(q_1, a, a) = (q_1, \in)$$
$$\delta(q_1, b, b) = (q_1, \in)$$

Step 4 : Finally, in state q_1, if the string is a palindrome, there is no input symbol to be scanned and the stack should be empty i. e., the stack should contain Z_0. Now, change the state to q_2 and do not alter the contents of the stack. The transition for this can be of the form

$$\delta(q_1, \in, Z_0) = (q_2, Z_0)$$

So, the PDA M to accept the language $L(M) = \{ ww^R \mid w \in (a, b)^* \}$

is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

where $Q = \{ q_0, q_1, q_2 \}$; $\Sigma = \{ a, b \}$; $\Gamma = \{ a, b, Z_0 \}$

δ : is shown below.

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_0, ba)$$
$$\delta(q_0, a, b) = (q_0, ab)$$
$$\delta(q_0, b, b) = (q_0, bb)$$
$$\delta(q_0, a, a) = (q_1, \in)$$
$$\delta(q_0, b, b) = (q_1, \in)$$
$$\delta(q_1, a, a) = (q_1, \in)$$
$$\delta(q_1, b, b) = (q_1, \in)$$
$$\delta(q_1, \in, Z_0) = (q_2, Z_0)$$

$q_0 \in Q$ is the start state of machine ; $Z_0 \in \Gamma$ is the initial symbol on the stack.

$F = \{q_2\}$ is the final state.

Note that the transitions numbered 3 and 7, 6 and 8 can be combined and the transitions can be written as shown below also .

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$
$$\delta(q_0, a, a) = \{ (q_0, aa), (q_1, \in) \}$$
$$\delta(q_0, b, a) = (q_0, ba)$$
$$\delta(q_0, a, b) = (q_0, ab)$$
$$\delta(q_0, b, b) = \{ (q_0, bb), (q_1, \in) \}$$
$$\delta(q_1, a, a) = (q_1, \in)$$
$$\delta(q_1, b, b) = (q_1, \in)$$
$$\delta(q_1, \in, Z_0) = (q_2, Z_0)$$

Note that once the following transitions are applied

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

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

if the input symbol is same as the symbol on top of the stack, the machine may push the current symbol on to the stack, or it may pop an element from the stack. At this point, the machine makes appropriate decision so that if the string is a palindrome, it has to accept. This machine is clearly a non - deterministic PDA (in short we call NPDA).

To accept the string :

The sequence of moves made by the PDA for the string **aabbaa** is shown below.

Initial ID

$(q_0, aabbaa, Z_0) \vdash (q_0, abbaa, aZ_0)$

$\vdash (q_0, bbaa, aaZ_0)$

PDA now pops an element instead of pushing $aa, aaZ_0)$

$a, aZ_0)$

$\vdash (q_1, \in, Z_0)$

$\vdash (q_2, \in, Z_0)$

(Final Configuration)

Since q_2 is the final state and input string is $\in$ in the final configuration, the string **aabbaa** is accepted by the PDA.

Example 8 : Construct a PDA which accepts the set of strings over $\{a, b\}$ with equal number of a's and b's such that all a's and b's are consecutive.

Solution :

We construct PDA M, which accepts given language by

(a) Empty store, and

(b) Final state

(a) By empty Store :

Let $M = (\{q_0\}, \{a,b\}, \{a,b,Z_0\}, \delta, q_0, Z_0, \phi)$. We know that the given language contains all the words over $\{a, b\}$ that have equal number of consecutive a's and b's. So, the given language

$L = \{a^n b^n : n \geq 0\} \cup \{b^n a^n : n \geq 0\}$.

We use stack either to hold a's to match with b's or b's to match with a's.

Transition function δ is defined as follows :

$\delta(q_0,a,Z_0) = \{(q_0,aZ_0)\}$, (To store first a on the stack)

$\delta(q_0,a,a) = \{(q_0,aa)\}$, (To store remaining a's on the stack)

$\delta(q_0,b,Z_0) = \{(q_0,bZ_0)\}$, (To store first b on the stack)

$\delta(q_0,b,b) = \{(q_0,bb)\}$, (To store remaining b's on the stack)

$\delta(q_0,b,a) = \{(q_0,\in)\}$, (To match input b with a on the stack in case of $L = \{a^n b^n : n \geq 0\}$

$\delta(q_0,a,b) = \{(q_0,\in)\}$, (To match input a with b on the stack in case of $L = \{b^n a^n : n \geq 0\}$)

$\delta(q_0,\in,Z_0) = \{(q_0,\in)\}$ (To make the stack empty)

(b) By final State : Let $M = (\{q_0,q_f\},\{a,b\},\{a,b,Z_0\},\delta,q_0,Z_0,\{q_f\})$

$\delta(q_0,a,Z_0) = \{(q_0,aZ_0)\}$, (To store first a on the stack)

$\delta(q_0,a,a) = \{(q_0,aa)\}$, (To store remaining a's on the stack)

$\delta(q_0,b,Z_0) = \{(q_0,bZ_0)\}$, (To store first b on the stack)

$\delta(q_0,b,b) = \{(q_0,bb)\}$, (To store remaining b's on the stack)

$\delta(q_0,b,a) = \{(q_0,\in)\}$, (To match input b with a on the stack)

$\delta(q_0,a,b) = \{(q_0,\in)\}$, (To match input a with b on the stack)

$\delta(q_0,\in,Z_0) = \{(q_f,Z_0)\}$ (To reach the final state)

Example 9 : Construct a PDA, which accepts $L = \{a^n c^m b^n : m, n \geq 1\}$.

Solution : Suppose PDA $M = (\{q_0\},\{a,b\},\{a,b,Z_0\},\delta,q_0,Z_0,\phi)$ accepts L.

We have restriction imposed on the number of a's and b's that it should be equal but not on the number of c's. So, PDA stores all a's on the stack and when c's encounter, then keep the stack unchanged and when b's encounters then matches with a's stored on the stack.

δ is defined as follows

$\delta(q_0,a,Z_0) = \{(q_0,aZ_0)\}$, (To store first a on the stack)

$\delta(q_0,a,a) = \{(q_0,aa)\}$, (To store remaining a's on the stack)

$\delta(q_0,c,a) = \{(q_0,a)\}$, (To read all c's on the input tape and keep stack contents unchanged),

$\delta(q_0,b,a) = \{(q_0,\in)\}$, (To match input b with a on the stack)

$\delta(q_0,\in,Z_0) = \{(q_0,\in)\}$ (To empty the stack)

Example 10 : Construct a PDA M, which accepts $L = \{a^n b^m c^m d^n : m, n \geq 1\}$

Solution :

We construct M by using the grammar of given language L. But, if we see the format of language L, then it is clear that each word of language L contains number of a's in the starting which is equal to the number of d's at the end. In the middle the number of b's is followed by an equal number of c's. So, all the starting a's and following b's are loaded on to the stack. Now, stack will contain b's at the top and a's at the bottom (LIFO) and on the input tape $c^m d^n$ remains. After this PDA M matches the number of c's with b's and d's with a's. This is shown in the below figure.

Let PDA $M = (\{q_0, q_f\}, \{a,b\}, \{a,b,Z_0\}, \delta, q_0, Z_0, \{q_f\})$, and δ is defined as follows

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$ (To store first a on the stack),

$\delta(q_0, a, a) = \{(q_0, aa)\}$ (To store remaining a's on the stack),

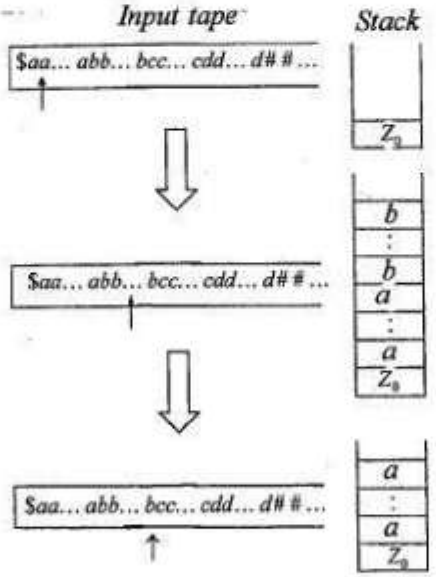
$\delta(q_0, b, a) = \{(q_0, ba)\}$ (To store first b on the stack and keeping stack contents unchanged),

$\delta(q_0, b, b) = \{(q_0, bb)\}$ (To store remaining b's on the stack),

$\delta(q_0, c, b) = \{(q_0, \in)\}$ (To match c's on the tape with b's on the stack),

$\delta(q_0, d, a) = \{(q_0, \in)\}$ (To match d's on the tape with a's on the stack), and

$\delta(q_0, \in, Z_0) = \{(q_f, Z_0)\}$ (To reach the final state)



Example 11 : Let $M = (\{p,q\},\{0,1\},\{Z_0,X\}, \delta, p, Z_0, \phi)$ be a PDA where δ is given by following transitions.

$\delta(p,1,Z_0) = \{(p,XZ_0)\}$,

$\delta(p,\in,Z_0) = \{(p,\in)\}$.

$\delta(p,1,X) = \{(p,XX)\}$.

$\delta(q,1,X) = \{(q,\in)\}$.

$\delta(p,0,X) = \{(q,X)\}$, *and*

$\delta(q,0,Z_0) = \{(p,Z_0)\}$

(a) What is the language accepted by this PDA by empty store ?

(b) Describe informally the working of the PDA.

Solution :

Let $R_1 : \delta(p,1,Z_0) = \{(p,XZ_0)\}$,

$R_2 : \delta(p,\in,Z_0) = \{(p,\in)\}$,

$R_3 : \delta(p,1,X) = \{(p,XX)\}$,

$R_4 : \delta(q,1,X) = \{(q,\in)\}$,

$R_5 : \delta(p,0,X) = \{(q,X)\}$, and

$R_6 : \delta(q,0,Z_0) = \{(p,Z_0)\}$

From the given transitions, we analyze the following things.

1. Using R_1 terminal 1 is stored as X on the stack in the state p .
2. Using R_2 , with no input PDA makes the stack empty in the state p .
3. Using R_3 , remaining 1's are stored as X's on the stack in the state p .
4. Using R_4 , input 1's are matched with X's stored on the stack in state q.
5. Using R_5 , PDA reads 0 and moves to the state q while maintaining X on the stack.
6. Using R_6 , PDA reads 0 on the tape and moves to the state p while maintaining Z_0 on the stack.

So, PDA reads 1's on the tape and loads these on to stack as X's (R_1 *and* R_3) . When 0 is read in state p then moves to the state $q(R_5)$. In the state q , 1's on the tape are matched with X's on the stack (R_4) and when 0 is read on the tape then PDA changes its state to $p(R_6)$. In the state p if no input is remaining on the tape and Z_0 is on the stack then PDA makes the stack empty (R_2) .

So, the accepted language $L = \{1^n 0 1^n 0 : n \geq 0\}$.

Example 12 : Let $Q = (\{q_0, q_1\},\{a,b\},\{a,b,Z\}, \delta, q_0, Z, \phi)$ be a PDA accepting by empty stack for the language which is the set of all nonempty even palindromes over the set $\{a, b\}$. Below is an incomplete specification of the transition δ. Complete the specification. The top of the stack is assumed to be at the right end of the string representing stack contents.

1. $\delta(q_0,a,Z) = \{(q_0,Za)\}$
2. $\delta(q_0,b,Z) = \{(q_0,Zb)\}$
3. $\delta(q_0,a,a) =$ ..... ,
4. $\delta(q_0,b,b) =$ ..... ,
5. $\delta(q_1,a,a)=\{(q_1,\in)\}$
6. $\delta(q_1,b,b) = \{(q_1,\in)\}$ and
7. $\delta(q_1,\in,Z) = \{(q_1,\in)\}$

Solution :

Let the set of even palindromes over $\{a, b\}$ is L, then

$L = \{\in, aa, bb, abba, baba, aaaa, bbbb, \ldots\}$

If we find the mid of a palindrome then left is the mirror image of right and right is the mirror image of the left. So, deciding the mid point is the problem here. We design a DPDA for given set.

In the given example either terminal symbol is stored on the stack to be matched later on after the mid point or if the corresponding match is there then PDA popped the stack symbol.

In the given transitions using (1) and (2) PDA loads a or b symbol on the stack, using (5) and (6) PDA matches the input with stack symbol and using (7) PDA makes the stack empty. So, if more a or b symbols are there then PDA will use (3) and (4) and in these either symbol will be loaded on the stack and remain in the state q_1 or matched with the same symbol and moved to state q_2.

So, (3) and (4) solve the non-determinism problem to select the mid point into the palindromes.

So, transitions are given below.

1. $\delta(q_0,a,Z) = \{(q_0,Za)\}$,
2. $\delta(q_0,b,Z) = \{(q_0,Zb)\}$
3. $\delta(q_0,a,a) = \{(q_0,aa),(q_1,\in)\}$
4. $\delta(q_0,b,b) = \{(q_0,bb),(q_1,\in)\}$
5. $\delta(q_1,a,a) = \{(q_1,\in)\}$
6. $\delta(q_1,b,b) = \{(q_1,\in)\}$ and
7. $\delta(q_1,\in,Z) = \{(q_1,\in)\}$

6.3 DETERMINISTIC AND NONDETERMINISTIC PUSHDOWN AUTOMATA

In this section, we will discuss about the deterministic and nondeterministic behavior of pushdown automata.

6.3.1 Nondeterministic PDA (NPDA)

Like NFA, nondeterministic PDA (NPDA) has finite number of choices for its inputs. As we have discussed in the mathematical description that transition function δ which maps from $Q \times (\Sigma \cup \{\in\}) \times \Gamma$ to (**finite subset** of) $Q \times \Gamma^*$. A nondeterministic PDA accepts an input if a sequence of choices leads to some final state or causes PDA to empty its stack. Since, sometimes it has more than one choice to move further on a particular input ; it means, PDA guesses the right choice always, otherwise it will fail and will be in hang state.

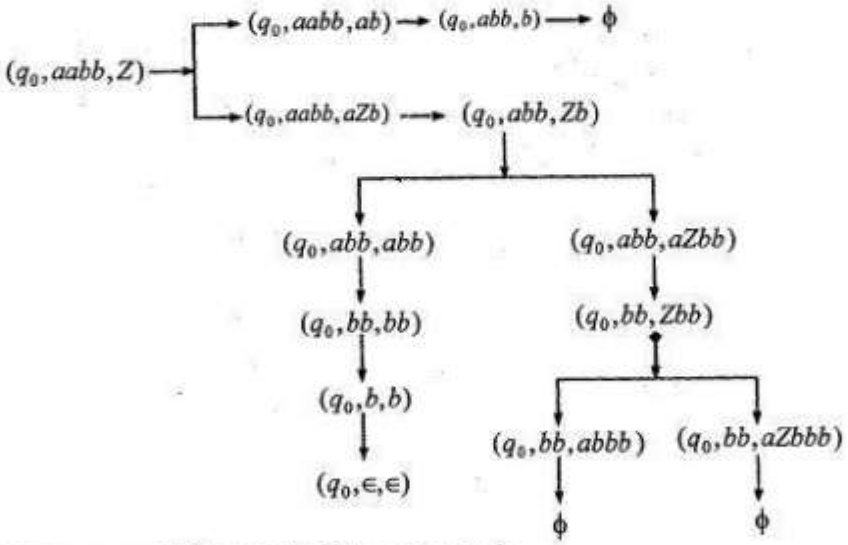
Example : consider a nondeterministic PDA $M = (\{q_0\},\{a,b\},\{a,b,Z\},\delta,q_0,Z,\phi)$, for the language $L = \{a^n b^n : n \geq 1\}$, where δ is defined as follows :

$\delta(q_0,\in,Z) = \{(q_0,ab),(q_0,aZb)\}$ (Two possible moves for input $\in$ on the tape and Z on the stack),

$\delta(q_0,a,a) = \{(q_0,\in)\}$, and $\delta(q_0,b,b) = \{(q_0,\in)\}$

Check whether string $w = aabb$ is accepted or not ?

Solution : Initial configuration is $(q_0,aabb,Z)$. Following moves are possible :



Hence, $w = aabb$ is accepted by empty stack.

One thing is noticeable here that only one move sequence leads to empty store and other don't. In other words, we say that some move sequence(s) leads to accepting configuration and other lead to hang state.

6.3.2 Deterministic PDA (DPDA)

Deterministic PDA (DPDA) is just like DFA, which has *at most one choice* to move for certain input. A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is deterministic if it satisfies both the conditions given as follows :

1. For any $q \in Q$, $a \in (\Sigma \cup \{\in\})$, and $Z \in \Gamma$, $\delta(q, a, Z)$ has at most one choice of move.
2. For any $q \in Q$, and $Z \in \Gamma$, if $\delta(q, \in, Z)$ is defined i.e. $\delta(q, \in, Z) \neq \phi$, then $\delta(q, a, Z) = \phi$ for all $a \in \Sigma$

Example : Consider a DPDA $M = (\{q_0, q_1\}, \{a, c\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$ accepting the language $\{a^n c a^n : n \geq 1\}$, where δ is defined as follows :

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$

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

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

$\delta(q_1, a, a) = \{(q_1, \in)\}$, and $\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$

Check whether the string $w = aacaa$ is accepted by empty stack or not ?

Solution :

We see that in each transition DPDA has at most one move. Initial configuration is $(q_0, aacaa, Z_0)$. Following are the possible moves.

$$(q_0, aacaa, Z_0) \rightarrow (q_0, acaa, aZ_0) \rightarrow (q_0, caa, aaZ_0) \rightarrow (q_1, aa, aaZ_0)$$

$$\downarrow$$

$$(q_1, \in, \in) \leftarrow (q_1, \in, Z_0) \leftarrow (q_1, a, aZ_0)$$

Hence, the string $w = aacaa$ is accepted by empty stack.

As we have discussed in earlier chapters that DFA and NFA are equivalent with respect to the language acceptance, but the same is not true for the PDA.

For example, language $L = \{ww^R : w \in (a \cup b)^*\}$ is accepted by nondeterministic PDA, can not by any deterministic PDA. A nondeterministic PDA can not be converted into equivalent deterministic PDA, but all DCFLs which are accepted by DPDA, are also accepted by NPDA. So, we say that deterministic PDA is a proper subset of nondeterministic PDA. Hence, the power of nondeterministic PDA is more as compared to deterministic PDA.

Procedure to find whether PDA is deterministic or not

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. The PDA is deterministic if

1. $\delta(q, a, Z)$ has only one element.
2. If $\delta(q, \in, Z)$ is not empty, then $\delta(q, a, Z)$ should be empty.

Both the conditions should be satisfied for the PDA to be deterministic. If one of the conditions fails, the PDA is non - deterministic.

Example 1 : Is the PDA to accept the language $L(M) = \{w\, Cw^R \mid w \in (a+b)^*\}$ is deterministic or not ?

Solution :

The transitions defined for this machine are

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_0, ba)$$
$$\delta(q_0, a, b) = (q_0, ab)$$
$$\delta(q_0, b, b) = (q_0, bb)$$
$$\delta(q_0, C, Z_0) = (q_1, Z_0)$$
$$\delta(q_0, C, a) = (q_1, a)$$
$$\delta(q_0, C, b) = (q_1, b)$$
$$\delta(q_1, a, a) = (q_1, \in)$$
$$\delta(q_1, b, b) = (q_1, \in)$$
$$\delta(q_1, \in, Z_0) = (q_2, Z_0)$$

The PDA should satisfy the two conditions shown in the procedure to be deterministic.

1. $\delta(q, a, Z)$ has only one element : Note that in the transitions, for each $q \in Q$, $a \in \Sigma$ *and* $Z \in \Gamma$, there is only one component defined and the first condition is satisfied.
2. The second condition states that if $\delta(q, \in, Z)$ is not empty, then $\delta(q, a, Z)$ should be empty i.e., if there is an $\in$ – transition, (in this case it is $\delta(q_1, \in, Z_0)$), then there should not be any transition from the state q_1 when top of the stack is Z_0 which is true.

Since, the PDA satisfies both the conditions, the PDA is deterministic.

Example 2 : Is the PDA corresponding to the language $L = \{ a^n b^n | n \geq 1 \}$ by a final state is deterministic or not ?

Solution :

The transitions defined for this machine are

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_1, \in)$$
$$\delta(q_1, b, a) = (q_1, \in)$$
$$\delta(q_1, \in, Z_0) = (q_2, \in)$$

The first condition to be deterministic is $\delta(q, a, Z)$ should have only one component. In this case, for each $q \in Q$, $a \in \Sigma$ and $Z \in \Gamma$, there exists only one definition. So, the first condition is satisfied.

To satisfy the second condition, consider the transition

$$\delta(q_1, \in, Z_0) = (q_2, \in)$$

Since the transition is defined, the transition $\delta(q_1, a, Z_0)$ where $a \in \Sigma$ should not be defined which is true. Since both the conditions are satisfied, the given PDA is deterministic.

Example 3 : Is the PDA to accept the language $L(M) = \{w | w \in (a+b)^* \text{ and } n_a(w) = n_b(w)$ is deterministic or not ?

Solution : The transitions defined for this machine are

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, b) = (q_0, bb)$$
$$\delta(q_0, a, b) = (q_0, \in)$$
$$\delta(q_0, b, a) = (q_0, \in)$$
$$\delta(q_0, \in, Z_0) = (q_1, Z_0)$$

The first condition to be deterministic is $\delta(q, a, Z)$ should have only one component. In this case, for each $q \in Q$, $a \in \Sigma$ and $Z \in \Gamma$, there exists only one component. So, the first condition is satisfied.

To satisfy the second condition, consider the transition

$$\delta(q_0, \in, Z_0) = (q_1, Z_0)$$

Since this transition is defined, the transition $\delta(q_0, a, Z_0)$ where $a \in \Sigma$ should not be defined. But, there are two transitions

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

defined from q_0 when top of the stack is Z_0. Since the second condition is not satisfied, the given PDA is non - deterministic PDA.

Example 4 : Give a deterministic PDA for the language $L = \{a^n cb^{2n} : n \geq 1\}$ over the alphabet $\Sigma = \{a, b, c\}$. Specify the acceptance state.

Solution :

Let DPDA $M = (\{q_a, q_b, q_f\}, \{a, b, c\}, \{a, Z_0\}, \delta, q_a, Z_0, \{q_f\})$ accepts the given language L. We analyze that in the given L, in each word, double of number of a's at the starting is equal to the number of b's at the end and no restriction apposed on the c in relation with the number of a's or b's. So, one a is read on the tape and stored as aa on the stack and when b's encounted then matched with a's on the stack.

In state q_a all a's are read and stored on the stack and when c is read then DPDA moves in the state q_b and in the state q_b, b's are read on the tape and matched with a's on the stack. When no symbol is on the tape then DPDA moves to the final state q_f.

The transition function δ is defined as follows:

$\delta(q_a, a, Z_0) = \{(q_a, aaZ_0)\}$ (To store the first a as aa on the stack),

$\delta(q_a, a, a) = \{(q_a, aaa)\}$ (To store the remaining a's as double on the stack),

$\delta(q_a, c, a) = \{(q_b, a)\}$ (To read c and move to state q_b),

$\delta(q_b, b, a) = \{(q_b, \in)\}$ (To match the b's and a's stored on the stack), and

$\delta(q_b, \in, Z_0) = \{(q_f, Z_0)\}$ (To reach the final state)

The acceptance state is q_f and DPDA M is shown in below figure.

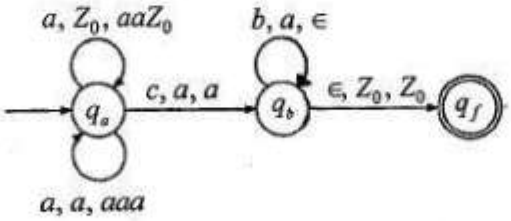


FIGURE : PDA accepting $\{a^n cb^{2n} : n \geq 1\}$

6.4 ACCEPTANCE OF LANGUAGE BY PDA

The language can be accepted by a Push Down Automata using two approaches.

1. **Acceptance by Final State :** The PDA accepts its input by consuming it and then it enters in the final state.

2. **Acceptance by empty stack :** On reading the input string from initial configuration for some PDA, the stack of PDA gets empty.

6.4.1 Equivalence of Empty Store and Final state acceptance

Theorem:

If $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, p_1, Z_1, \phi)$ is a PDA accepting CFL L by empty store then there exists PDA $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, p_2, Z_2, \{q_f\})$ which accepts L by final state.

Proof :

First we construct PDA M_2 based on PDA M_1 and then we prove that both accept L.

Step 1 : Construction of PDA M_2 based on given PDA M_1

Σ is same for both PDAs. We add a new initial state and a new final state with given PDA M_1.

So, $Q_2 = Q_1 \cup \{p_2 \cup q_f\}$

The stack alphabet Γ_2 of PDA M_2 contains one additional symbol Z_2 with Γ_1.

So, $\Gamma_2 = \Gamma_1 \cup \{Z_2\}$

The transition function δ_2 contains all the transitions of given PDA M_1 and two additional transitions (R_1 *and* R_3) as defined as follows :

$R_1 : \delta_2(p_2, \in, Z_2) = \{(p_1, Z_1 Z_2)\}$,

$R_2 : \delta_2(q, a, Z) = \delta_1(q, a, Z)$ for all (q, a, Z) in $Q_1 \times (\Sigma \cup \{\in\}) \times \Gamma_1$
(the original transitions of M_1), and

$R_3 : \delta_2(q, \in, Z_2) = \{(q_f, \in)\}$ for all $q \in Q_1$

By the R_1, M_2 moves from its initial ID $(p_2, \in, Z_2)$ to the initial ID of M_1. By R_2, M_2 uses all the transitions of M_1 after reaching the initial ID of M_1 and by using R_3 M_2 reaches the final state q_f.

The block diagram is shown in below figure.

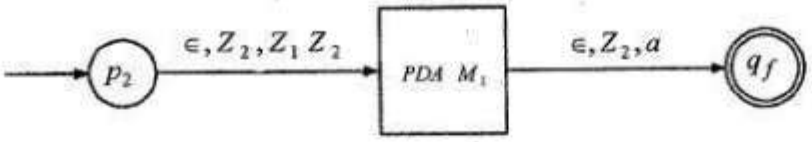


FIGURE : Block diagram of PDA M_2

Step 2 : The language accepted by PDA M_1 and PDA M_2

The behaviors of M_1 and M_2 are same except the two by $\in$ - moves defined by R_1 *and* R_3.

Let string $w \in L$ and accepted by M_1, then

$(p_1, w, Z_1) \vdash_{M_1}^{*} (q, \in, \in)$ where $q \in Q_1$ **(Result 1)**

For M_2, the initial ID is (p_2, w, Z_2) and it can be written as $(p_2, \in w \in, Z_2)$. So,

$(p_2, \in w \in, Z_2) \vdash_{M_2} (p_1, w, Z_1 Z_2)$ (This initial ID of M_1)

$\vdash_{M_2}^{*} (q, \in, Z_2)$ (by R_2 and Result 1)

$\vdash_{M_2}^{*} (q_f, \in, \alpha)$ $\alpha \in \Gamma_2^{*}$ (By R_3)

Thus, if M_1 accepts w, then M_2 also accepts it.

It means $L(M_2) \subseteq L(M_1)$ **(Result 2)**

Let string $w \in L$ and accepted by PDA M_2, then

$(p_2, \in w \in, Z_2) \vdash_{M_2} (p_1, w, Z_1 Z_2)$ (By R_1) **(Result 3)**

$\vdash_{M_2}^{*} (q, \in, Z_2)$ (By R_2) **(Result 4)**

$\vdash_{M_2} (q_f, \in, \alpha)$ $\alpha \in \Gamma_2^{*}$ (By R_3)

Note : The Result 3 is the initial ID of M_1. The Result 4 shows the empty store for M_1 if symbol Z_2 is not there.

For M_1, the initial ID is (p_1, w, Z_1)

So, $(p_1, w, Z_1) \vdash_{M_2}^{*} (q, \in, \in)$, where $q \in Q_1$ (By Result 3 and Result 4) Thus, if M_2 accepts w, then M_1 also accepts it.

It means, $L(M_1) \subseteq L(M_2)$ **(Result 5)**

Therefore, $L = L(M_2) = L(M_1)$ (From Result 2 and Result 5)

Hence, the statement of theorem is proved.

Example: Consider a nondeterministic PDA $M_1 = (\{q_0\}, \{a,b\}, \{a,b,S\}, \delta, q_0, S, \phi)$ which accepts the language $L = \{a^n b^n : n \geq 1\}$ by empty store, where δ is defined as follows :

$\delta(q_0, \in, S) = \{(q_0, ab), (q_0, aSb)\}$ (Two possible moves),

$\delta(q_0, a, a) = \{(q_0, \in)\}$, and $\delta(q_0, b, b) = \{(q_0, \in)\}$

Construct an equivalent PDA M_2 which accepts L in final state and check whether string $w = aabb$ is accepted or not ?

Solution : Following moves are carried out by PDA M_1 in order to accept $w = aabb$:

$$(q_0, aabb, S) \vdash (q_0, aabb, aSb)$$
$$\vdash (q_0, abb, Sb)$$
$$\vdash (q_0, abb, abb)$$
$$\vdash (q_0, bb, bb)$$
$$\vdash (q_0, b, b)$$
$$\vdash (q_0, \in, \in)$$

Hence, $(q_0, aabb, S) \vdash_{M_1}^{*} (q_0, \in, \in)$

Therefore, $w = aabb$ is accepted by M_1.

Construction of PDA M_2 based on given PDA M_1

Let PDA $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, p_2, Z_2, \{q_f\})$, where

$Q_2 = \{q, q_f, p_2\}$; $\Sigma = \{a, b\}$

$\Gamma_2 = \{a, b, S, Z_2\}$, and transition function δ_2 is defined as follows :

$\delta_2(p_2, \in, Z_2) = \{(q, SZ_2)\}$ (Using R_1)

$\delta_2(q, \in, S) = \{(q, ab), (q, aSb)\}$ (Using R_2)

$\delta_2(q, a, a) = \{(q, \in)\}$ (Using R_2)

$\delta_2(q, b, b) = \{(q, \in)\}$ (Using R_2)

$\delta_2(q, \in, Z_2) = \{(q_f, \in)\}$ (Using R_3)

Following moves are carried out by PDA M_2 in order to accept $w = aabb$:

$$(p_2, aabb, Z_2) \vdash (q, aabb, SZ_2)$$

$$\vdash (q, aabb, aSbZ_2)$$

$$\vdash (q, abb, SbZ_2)$$

$$\vdash (q, abb, abbZ_2)$$

$$\vdash (q, bb, bbZ_2)$$

$$\vdash (q, b, bZ_2)$$

$$\vdash (q, \in, Z_2)$$

$$\vdash (q_f, \in, Z_2)$$

Hence, $(p_2, aabb, Z_2) \overset{*}{\underset{M_2}{\vdash}} (q_f, \in, Z_2)$

The PDA M_2 is shown in below figure.

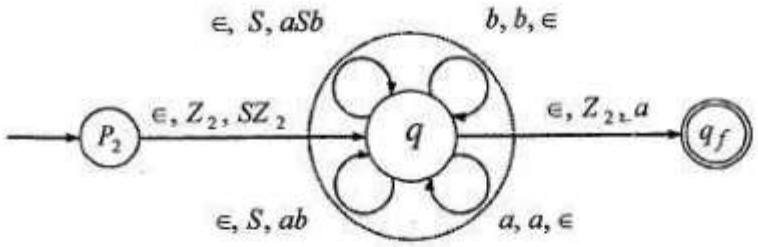


FIGURE : PDA M_2 accepting $\{a^n b^n : n \geq 1\}$

6.4.2 Equivalence of Final state and Empty Store acceptance

Theorem :

If $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_0, Z_0, F)$ is a PDA accepting CFL L by final state then there exists PDA $M_2 = (Q_2, \Sigma, \Gamma, \delta_2, q_0, Z_0, \phi)$ which accepts L by empty store.

Proof :

First we construct PDA M_2 based on PDA M_1 and then we prove that both accept L.

Step 1 : Construction of PDA M_2 based on given PDA M_1

Σ, Γ, initial state q_0, and initial symbol on the stack Z_0 are same for both PDAs. We add a new state with given PDA M_1. All final states of PDA M_1 are converted into non-final states.

So, $Q_2 = Q_1 \cup \{E\}$ (where E is new added state)

The transition function δ_2 contains all the transitions of given PDA M_1 and additional transitions (R_2 *and* R_3) defined as follows :

$R_1 : \delta_2(q, a, Z) = \delta_1(q, a, Z)$ for all (q, a, Z) in $Q_1 \times (\Sigma \cup \{\in\}) \times \Gamma_1$ (the original transitions of M_1),

$R_2 : \delta_2(p, \in, \alpha) = \{(E, \alpha)\}$ for all $p \in F$ and $\alpha \in \Gamma^*$, and

$R_3 : \delta_2(E, \in, \alpha) = \{(E, \in)\}$ for all $\alpha \in \Gamma^*$ and $E \in Q_2$

By R_1, PDA M_2 uses all the transitions of M_1 and reaches the final state if acceptability is there. By R_2, M_2 reaches state E and after reaching state E, by R_3, erases all the stack symbols. R_3 provides a loop incase of stack is not empty. The block diagram is shown in below figure.

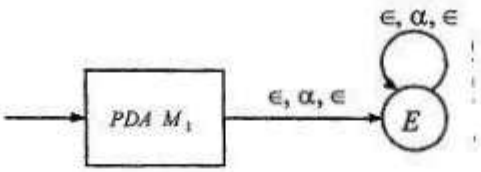


FIGURE : Block diagram of PDA M_2

Step 2 : The language accepted by PDA M_1 and PDA M_2

The behaviors of PDA M_1 and PDA M_2 are the same except the two $\in$ - moves defined in R_2 and R_3.

Let string $w \in L$ and accepted by PDA M_1, then

$(q_0, w, Z_0) \vdash_{M_1}^{*} (q_f, \in, \alpha)$, where $q_f \in F$ and $\alpha \in \Gamma^*$ **(Result 1)**

For PDA M_2, the initial ID is (q_0, w, Z_0).

So, $(q_0, w, Z_0) \vdash_{M_1}^{*} (q_f, \in, \alpha)$, where $q_f \in Q_2$ and $\alpha \in \Gamma^*$ (By R_1 and Result 1)

$\vdash_{M_1}^{*} (E, \in, \alpha)$ (By R_2)

$\vdash_{M_1}^{*} (E, \in, \in)$ (By R_3)

Thus, if M_1 accepts w, then M_2 also accepts it.

It means, $L(M_2) \subseteq L(M_1)$ **(Result 2)**

Let string $w \in L$ and accepted by M_2, then

For PDA M_2, the initial ID is (q_0, w, Z_0). So

$(q_0, w, Z_0) \vdash_{M_1}^{*} (q_f, \in, \alpha)$ for some $q_f \in Q_2$ and $\alpha \in \Gamma^*$ **(Result 3)**

$\vdash_{M_1}^{*} (E, \in, \alpha)$ (By R_2)

$\vdash_{M_1}^{*} (E, \in, \in)$ (By R_3)

For M_1, the initial ID is (q_0, w, Z_0).

So, $(q_0, w, Z_0) \vdash_{M_1}^{*} (q_f, \in, \alpha)$, where $q_f \in Q_2$ and $\alpha \in \Gamma^*$ **(By Result 3)**

Thus, if M_2 accepts w, then M_1 also accepts it.

It means $L(M_1) \subseteq L(M_2)$ **(Result 4)**

Therefore, $L = L(M_2) = L(M_1)$ (From Result 2 and Result 4)

Hence, the statement of theorem is proved.

Example :

Consider a PDA $M_1 = (\{q_0, q_1, q_2\}, \{a,c\}, \{a, Z_0\}, \delta_1, q_0, Z_0, \{q_2\})$, convert it into PDA M_2 whose acceptance is by empty store. Check the acceptability of string $w = aacaa$.

The transition function δ_1 is defined as follows :

$\delta_1(q_0, a, Z_0) = \{(q_0, aZ_0)\}$,

$\delta_1(q_0, a, a) = \{(q_0, aa)\}$,

$\delta_1(q_0, c, a) = \{(q_1, a)\}$

$\delta_1(q_1, a, a) = \{(q_1, \in)\}$ and

$\delta_1(q_1, \in, Z_0) = \{(q_2, Z_0)\}$

Solution :

Following moves are carried out by M_1 in order to check acceptability of string $w = aacaa$

$$(q_0, aacaa, Z_0) \vdash (q_0, acaa, aZ_0)$$

$$\vdash (q_0, caa, aaZ_0)$$

$$\vdash (q_1, aa, aaZ_0)$$

$$\vdash (q_1, a, aZ_0)$$

$$\vdash (q_1, \in, Z_0)$$

$$\vdash (q_2, \in, Z_0)$$

Hence, $(q_0, aacaa, Z_0) \overset{*}{\underset{M_1}{\vdash}} (q_2, \in, Z_0)$

Therefore, PDA M_1 accepts the string $aacaa$.

The transition function δ_2 for M_2 is defined as follows :

$\delta_2(q_0, a, Z_0) = \{(q_0, aZ_0)\}$ (Using R_1),

$\delta_2(q_0, a, a) = \{(q_0, aa)\}$ (Using R_1),

$\delta_2(q_0, c, a) = \{(q_1, a)\}$ (Using R_1),

$\delta_2(q_1, a, a) = \{(q_1, \in)\}$ (Using R_1)

$\delta_2(q_1, \in, Z_0) = \{(q_2, Z_0)\}$ (Using R_1)

$\delta_2(q_2, \in, a) = \{(E, a)\}$ (Using R_2)

$\delta_2(q_2, \in, c) = \{(E, c)\}$ (Using R_2)

$\delta_2(q_2, \in, Z_0) = \{(E, Z_0)\}$ (Using R_2)

$\delta_2(E, \in, a) = \{(E, \in)\}$ (Using R_3)

$\delta_2(E, \in, c) = \{(E, \in)\}$ (Using R_3), and

$\delta_2(E, \in, Z_0) = \{(E, \in)\}$ (Using R_3)

Following moves are carried out by M_2 in order to check the acceptability of the string $w = aacaa$

$$(q_0, aacaa, Z_0) \vdash (q_0, acaa, aZ_0)$$
$$\vdash (q_0, caa, aaZ_0)$$
$$\vdash (q_1, aa, aaZ_0)$$
$$\vdash (q_1, a, aZ_0)$$
$$\vdash (q_1, \in, aZ_0)$$
$$\vdash (q_2, \in, Z_0)$$
$$\vdash (E, \in, Z_0)$$
$$\vdash (E, \in, \in)$$

Hence, $(q_0, aacaa, Z_0) \overset{*}{\underset{M_2}{\vdash}} (E, \in, \in)$

6.5 PUSHDOWN AUTOMATA AND CFL

6.5.1 PDA FROM CFG

It is quite easy to get a PDA from the context free grammar. This is possible only from a CFG which is in GNF. So, given any grammar, first obtain the grammar in GNF and then obtain PDA. The steps to be followed to convert a grammar to its equivalent PDA are shown below.

1. Convert the grammar into GNF
2. Let q_0 be the start state and Z_0 is the initial symbol on the stack. Without consuming any input, push the start symbol S onto the stack and change the state to q_1. The transition for this can be

$$\delta(q_0, \in, Z_0) = (q_1, SZ_0)$$

3. For each production of the form

$$A \rightarrow a\alpha$$

introduce the transition $\delta(q_1, a, A) = (q_1, \alpha)$

4. Finally, in state q_1, without consuming any input, change the state to q_f which is an accepting state. The transition for this can be of the form

$$\delta(q_1, \in, Z_0) = (q_f, Z_0)$$

Example 1 : For the grammar

S → aABC
A → aB | a
B → bA|b
C → a Obtain the corresponding PDA

Solution :

Let q_0 be the start state and Z_0 the initial symbol on the stack.

Step 1 : Push the start symbol S on to the stack and change the state to q_1. The transition for this can be of the form

$$\delta(q_0, \in, Z_0) = (q_1, SZ_0)$$

Step 2 : For each production $A \rightarrow a\alpha$ introduce the transition

$$\delta(q_1, a, A) = (q_1, \alpha)$$

This can be done as shown below.

Production			Transition
S	→	aABC	$\delta(q_1, a, S) = (q_1, ABC)$
A	→	aB	$\delta(q_1, a, A) = (q_1, B)$
A	→	a	$\delta(q_1, a, A) = (q_1, \in)$
B	→	bA	$\delta(q_1, b, B) = (q_1, A)$
B	→	b	$\delta(q_1, b, B) = (q_1, \in)$
C	→	a	$\delta(q_1, a, C) = (q_1, \in)$

Step 3 : Finally in state q_1, without consuming any input change the state to q_f which is an accepting state

$$\text{i.e., } \delta(q_1, \in, Z_0) = (q_f, Z_0)$$

So, the PDA M is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

where $Q = \{q_0, q_1, q_f\}$; $\Sigma = \{a, b\}$; $\Gamma = \{S, A, B, C, Z_0\}$

δ : is shown below.

$$\delta(q_0, \in, Z_0) = (q_1, SZ_0)$$
$$\delta(q_1, a, S) = (q_1, ABC)$$
$$\delta(q_1, a, A) = (q_1, B)$$
$$\delta(q_1, a, A) = (q_1, \in)$$
$$\delta(q_1, b, B) = (q_1, A)$$
$$\delta(q_1, b, B) = (q_1, \in)$$
$$\delta(q_1, a, C) = (q_1, \in)$$
$$\delta(q_1, \in, Z_0) = (q_f, Z_0)$$

$q_0 \in Q$ is the start state of machine ; $Z_0 \in \Gamma$ is the initial symbol on the stack.

$F = \{q_f\}$ is the final state

Note that the terminals grammar G will be input symbols in PDA and the non - terminals will be the stack symbols in PDA.

The derivation from the grammar is shown below

$$\begin{aligned} S &\Rightarrow aABC \\ &\Rightarrow aaBBC \\ &\Rightarrow aabBC \\ &\Rightarrow aabbC \\ &\Rightarrow aabba \end{aligned}$$

The string aabba is derived from the start symbol S. The same string should be accepted by PDA also. The moves made by the PDA are shown below.

Initial ID			
$(q_0, aabba, Z_0)$	$\vdash$	$(q_1, aabba, SZ_0)$	By Rule 1
	$\vdash$	$(q_1, abba, ABCZ_0)$	By Rule 2
	$\vdash$	$(q_1, bba, BBCZ_0)$	By Rule 3

$\vdash$ (q_1, ba, BCZ_0) By Rule 6

$\vdash$ (q_1, a, CZ_0) By Rule 6

$\vdash$ $(q_1, \in, Z_0)$ By Rule 7

$\vdash$ $(q_f, \in, Z_0)$ By Rule 8

(Final Configuration)

Since q_f is the final state and input string is $\in$ in the final configuration, the string **aabba** is accepted by the PDA.

Example 2 : Construct PDA M equivalent to the CFG $S \rightarrow 0BB, B \rightarrow 1S \mid 0S \mid 0$ and check whether 010000 is in $N(M)$ or not ?

Solution : Let PDA $M = (\{q_0\},\{a,b\},\{S,B,0,1\},\delta,q_0,S,\phi)$, δ be defined as follows :

$\delta(q_0,\in,S) = \{(q_0,0BB)\}$ (For the production $S \rightarrow 0BB$),

$\delta(q_0,\in,B)=\{(q_0,1S)\}$ (For the production $B \rightarrow 1S$),

$\delta(q_0,\in,B) =\{(q_0,0S)\}$ (For the production $B \rightarrow 0S$),

$\delta(q_0,\in,B) = \{(q_0,0)\}$ (For the production $B \rightarrow 0$),

$\delta(q_0,0,0) = \{(q_0,\in)\}$ (For terminal 0),

$\delta(q_0,1,1) = \{(q_0,\in)\}$ (For terminal 1)

For string $w = 010000, M$ has following moves :

$(q_0,010000\ ,S) \vdash (q_0,\underline{0}10000\ ,\underline{0}BB)$

$\vdash (q_0,10000\ ,\underline{B}B)$

$\vdash (q_0,\underline{1}0000\ ,\underline{1}SB)$

$\vdash (q_0,0000\ ,\underline{S}B)$

$\vdash (q_0,\underline{0}000\ ,\underline{0}BBB)$

$\vdash (q_0,000\ ,\underline{B}BB)$

$\vdash (q_0,\underline{0}00\ ,\underline{0}BB)$

$\vdash (q_0,00\ ,\underline{B}B)$

$$\vdash (q_0, \underline{0}\,0, \underline{0}\,B)$$

$$\vdash (q_0, 0, \underline{B})$$

$$\vdash (q_0, \underline{0}, \underline{0})$$

$$\vdash (q_0, \in, \in)$$

Hence, $010000 \in N(M)$.

6.5.2 Construction of CFG from Given PDA

As per our discussion, the CFG and PDA has a strong relationship. As we have seen in the previous section that we can construct a PDA from given CFG. Similarly we can obtain a CFG from given PDA.

Theorem : If $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \phi)$ is a PDA which accepts the language L, then there is a CFG $G = (V, T, P, S)$ such that $L(G) \subseteq N(M) = L$.

Proof : $V = \{S\} \cup \{[p, Z, q] : p, q \in Q \text{ and } Z \in \Gamma\}$, T is same for both P includes the following productions :

$P_1 : S \to [q_0, Z_0, q]$ is in P for every $q \in Q$

$P_2 : [p, Z, q] \to a$ is in P for every $p, q \in Q, a \in T \cup \{\in\}$, and $Z \in \Gamma$ such that $\delta(p, a, Z) = \{(q, \in)\}$, and

$P_3 : [p, Z, q_{m+1}] \to a\,[q_1, B_1, q_2]\,[q_2, B_2, q_3] \ldots [q_m, B_m, q_{m+1}]$ is in P for every $p, q_i \in Q, a \in (T \cup \{\in\})$, and $Z, B_i \in \Gamma$, where $1 \le i \le m$ such that $\delta(p, a, Z) = \{(q_1, B_1 B_2 \ldots B_m)\}$.

If $m = 0$, then $[p, Z, q_1] \to a$

Example 1 : Consider the PDA $M = (\{q_0, q_1\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$ accepting the language $L = \{a^n b^m a^n : m, n \ge 1\}$, which has the following transition function.

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$,

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

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

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

$\delta(q_1, a, a) = \{(q_1, \in)\}$, *and*

$\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$

Construct a CFG G which generates the same language.

Solution : Let CFG $G = (V, T, P, S)$, where

$V = \{[q_0, Z_0, q_0], [q_0, Z_0, q_1], [q_1, Z_0, q_0], [q_1, Z_0, q_1], [q_0, a, q_0], [q_0, a, q_1], [q_1, a, q_0], [q_1, a, q_1], S\}$

(**NOTE :** The number of states are two and stack symbols are two, then number of combination of these in triple form is $2 \times 2 \times 2 = 2^3 = 8$).

$\Sigma = \{a, b\}$, S is the start symbol, and P consists of following production rules :

Using construction rule P_1 :

$S \to [q_0, Z_0, q_0]$ and $S \to [q_0, Z_o, q_1]$

For transition $\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$ **:**

$[q_0, Z_o, q_0] \to a[q_0, a, q_0][q_0, Z_o, q_0]$ (Using construction rule P_3)

$[q_0, Z_o, q_0] \to a[q_0, a, q_1][q_1, Z_0, q_0]$ (Using construction rule P_3)

$[q_0, Z_0, q_1] \to a[q_0, a, q_0][q_0, Z_0, q_1]$ (Using construction rule P_3)

$[q_0, Z_0, q_1] \to a[q_0, a, q_1][q_1, Z_0, q_1]$ (Using construction rule P_3)

For transition $\delta(q_0, a, a) = \{(q_0, aa)\}$ **:**

$[q_0, a, q_0] \to a[q_0, a, q_0][q_0, a, q_0]$ (Using construction rule P_3)

$[q_0, a, q_0] \to a[q_0, a, q_1][q_1, a, q_0]$ (Using construction rule P_3)

$[q_0, a, q_1] \to a[q_0, a, q_0][q_0, a, q_1]$ (Using construction rule P_3)

$[q_0, a, q_1] \to a[q_0, a, q_1][q_1, a, q_1]$ (Using construction rule P_3)

For transition $\delta(q_0, b, a) = \{(q_1, a)\}$ **:**

$[q_0, a, q_0] \to b[q_1, a, q_0]$ (Using construction rule P_3)

$[q_0, a, q_1] \to b[q_1, a, q_1]$ (Using construction rule P_3)

For transition $\delta(q_1, b, a) = \{(q_1, a)\}$ **:**

$[q_1, a, q_0] \to b[q_1, a, q_0]$ (Using construction rule P_3)

$[q_1, a, q_1] \to b[q_1, a, q_1]$ (Using construction rule P_3)

For transition $\delta(q_1, a, a) = \{(q_1, \in)\}$ **:**

$[q_1, a, q_1] \to a$ (Using construction rule P_2)

For transition $\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$ **:**

$[q_1, Z_0, q_1] \to \in$ (Using construction rule P_2)

Example 2 : Construct CFG G for the PDA given as follows :

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$

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

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

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

$\delta(q_0, \in, Z_0) = \{(q_0, \in)\}$

Solution :

Let $G = (V, T, P, S)$, $V = \{[q_0, Z_0, q_0], [q_0, a, q_0]\}$, $\Sigma = \{a, b, c\}$, P consists of following production rules :

$S \rightarrow [q_0, Z_0, q_0]$,

For transition $\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$:

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

For the transition $\delta(q_0, a, a) = \{(q_0, aa)\}$:

$[q_0, a, q_0] \rightarrow a[q_0, a, q_0]\,[q_0, a, q_0]$

For the transition $\delta(q_0, c, a) = \{(q_0, a)\}$:

$[q_0, a, q_0] \rightarrow c[q_0, a, q_0]$

For the transition $\delta(q_0, b, a) = \{(q_0, \in)\}$:

$[q_0, a, q_0] \rightarrow b$

For the transition $\delta(q_0, \in, Z_0) = \{(q_0, \in)\}$:

$[q_0, Z_0, q_0] \rightarrow \in$

Example 3 : Let $M = (\{q_0, q_1\}, \{a, b\}, \{c, Z_0\}, \delta, q_0, Z_0, \phi)$ is a PDA and δ is defined as follows:

$\delta(q_0, a, Z_0) = \{(q_0, cZ_0)\}$,

$\delta(q_0, a, c) = \{(q_0, cc)\}$,

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

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

$\delta(q_1, \in, c) = \{(q_1, \in)\}$,

$\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$

Construct CFG G generating $N(M)$.

Solution :

Let $G = (V, T, P, S)$, where

1. V contains elements from the set

   $\{S, [q_0,c,q_0], [q_0,c,q_1], [q_0,Z_0,q_0], [q_0,Z_0,q_1], [q_1,c,q_1], [q_1,c,q_0], [q_1,Z_0,q_1], [q_1,Z_0,q_0]\}$,
2. $\Sigma = \{a, b\}$,
3. S is the starting symbol, and
4. P includes the following production rules.

   For variable S

   $P_1: S \rightarrow [q_0, Z_0, q_0]$, and

   $P_2: S \rightarrow [q_0, Z_0, q_1]$

   For variable $[q_0, Z_0, q_0]$ using transition $\delta(q_0, a, Z_0) = \{(q_0, cZ_0)\}$

   $P_3: [q_0, Z_0, q_0] \rightarrow a[q_0, c, q_0][q_0, Z_0, q_0]$ and

   $P_4: [q_0, Z_0, q_0] \rightarrow a[q_0, c, q_1][q_1, Z_0, q_0]$

   For variable $[q_0, Z_0, q_1]$ $P_5: [q_0, Z_0, q_1] \rightarrow a[q_0, c, q_0][q_0, Z_0, q_1]$ and

   $P_6: [q_0, Z_0, q_1] \rightarrow a[q_0, c, q_1][q_1, Z_0, q_1]$

   For variable $[q_0, c, q_0]$ using transition $\delta(q_0, a, c) = \{(q_0, cc)\}$:

   $P_7: [q_0, c, q_0] \rightarrow a[q_0, c, q_0][q_0, c, q_0]$, and

   $P_8: [q_0, c, q_0] \rightarrow a[q_0, c, q_1][q_1, c, q_0]$

   For variable $[q_0, c, q_1]$:

   $P_9: [q_0, c, q_1] \rightarrow a[q_0, c, q_0][q_0, c, q_1]$ and

   $P_{10}: [q_0, c, q_1] \rightarrow a[q_0, c, q_1][q_1, c, q_1]$

   For variable $[q_0, c, q_1]$ using transition $\delta(q_0, b, c) = \{(q_1, \epsilon)\}$:

   $P_{11}: [q_0, c, q_1] \rightarrow b$

   For variable $[q_1, Z_0, q_1]$ using transition $\delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$:

   $P_{12}: [q_1, Z_0, q_1] \rightarrow \epsilon$

   For variable $[q_1, c, q_1]$ using transition $\delta(q_1, \epsilon, c) = \{(q_1, \epsilon)\}$:

   $P_{13}: [q_1, c, q_1] \rightarrow \epsilon$

   For variable $[q_1, c, q_1]$ using transition $\delta(q_1, b, c) = \{(q_1, \epsilon)\}$:

   $P_{14}: [q_1, c, q_1] \rightarrow b$

We have no production for the variable $[q_1, Z_0, q_0]$ and $[q_1, c, q_0]$. So, these are discarded with their associated productions.

So, now we have following productions.

$P_1: S \rightarrow [q_0, Z_0, q_0]$,

$P_2: S \rightarrow [q_0, Z_0, q_1]$,

$P_3: [q_0, Z_0, q_0] \rightarrow a[q_0, c, q_0][q_0, Z_0, q_0]$,

$P_5: [q_0, Z_0, q_1] \rightarrow a[q_0, c, q_0][q_0, Z_0, q_1]$,

$P_6: [q_0, Z_0, q_1] \rightarrow a[q_0, c, q_1][q_1, Z_0, q_1]$,

$P_7: [q_0, c, q_0] \rightarrow a[q_0, c, q_0][q_0, c, q_0]$,

$P_9: [q_0, c, q_1] \rightarrow a[q_0, c, q_0][q_0, c, q_1]$,

$P_{10}: [q_0, c, q_1] \rightarrow a[q_0, c, q_1][q_1, c, q_1]$,

$P_{11}: [q_0, c, q_1] \rightarrow b$,

$P_{12}: [q_1, Z_0, q_1] \rightarrow \in$,

$P_{13}: [q_1, c, q_1] \rightarrow \in$,

$P_{14}: [q_1, c, q_1] \rightarrow b$.

In Production P_3, and P_7, variables $[q_0, Z_0, q_0]$ and $[q_0, c, q_0]$ have right and left recursion respectively but have no terminating production i.e. no terminal from these variables. So, all the productions that include these variables including P_3 and P_7 are discarded.

Now, we have following productions included in P:

$P_2: S \rightarrow [q_0, Z_0, q_1]$,

$P_6: [q_0, Z_0, q_1] \rightarrow a[q_0, c, q_1][q_1, Z_0, q_1]$,

$P_{10}: [q_0, c, q_1] \rightarrow a[q_0, c, q_1][q_1, c, q_1]$,

$P_{11}: [q_0, c, q_1] \rightarrow b$,

$P_{12}: [q_1, Z_0, q_1] \rightarrow \in$,

$P_{13}: [q_1, c, q_1] \rightarrow \in$,

$P_{14}: [q_1, c, q_1] \rightarrow b$.

Theorem : Let L_1 be a context - free language and L_2 be a regular language. Then show that $L_1 \cap L_2$ is context - free.

Proof :

Let $M_1 = (Q, \Sigma, \Gamma, \delta_1, q_0, Z, F_1)$ be an NPDA which accepts L_1 and $M_2 = (P, \Sigma, \Gamma, \delta_2, q_0, F_2)$ be a DFA that accepts L_2. We construct a pushdown automaton $M = (\hat{Q}, \Sigma, \Gamma, \hat{\delta}, \hat{q}_0, Z, \hat{F})$ which simulates the parallel action of M_1 and M_2, whenever a symbol is read from the input string, $\hat{M}$ simultaneously executes the move M_1 and M_2.

Let $\hat{Q} = Q \times P$, $\hat{q}_0 = (q_0, p_0)$, $\hat{F} = F_1 \times F_2$

and define $\hat{\delta}$ such that, $((q_k, p_2), x) \in \hat{\delta}((q_i, p_j), a, b)$, if and only if,

$(q_k, x) \in \delta_1(q_i, a, b)$ and $\delta_2(p_j, a) = p_l$

In this, we also require that if $a = \in$, then $p_j = p_l$. In other words, the states of $\hat{M}$ are labeled with pairs (q_i, p_j), representing the respective states in which M_1 and M_2 can be after reading a certain input string. It is a straightforward induction argument to show that,

$((q_0, p_0), w, Z) \vdash^* ((q_r, p_s), x)$,

with $q_r \in F_1$ and $p_s \in F_2$

if and only if, $(q_0, w, z) \; M_1^{(q_r, x)}$ and $\hat{\delta}(p_0, w) = p_s$.

Therefore, a string is accepted by M' if and only if it is accepted by M_1 and M_2 that is, if it is in $L(M_1) \cap L(M_2) = L_1 \cap L_2$.

REVIEW QUESTIONS

Q1. What is PDA ? Explain.

Answer :

For Answer refer to Topic : 6.1 , Page No : 6.1.

Q2. Obtain a PDA to accept the language $L(M) = \{ wCw^R \mid w \in (a+b)^* \}$ where W^R is reverse of W .

Answer :

For Answer refer to example - 1 , Page No : 6.6.

Q3. Obtain a PDA to accept the language $L = \{ a^n b^n \mid n \geq 1 \}$ by a final state.

Answer :

For Answer refer to example - 2 , Page No : 6.9.

Q4. Obtain a PDA to accept the language $L(M) = \{ w \mid w \in (a+b)^* \text{ and } n_a(w) = n_b(w) \}$.

Answer :

For Answer refer to example - 3 , Page No : 6.12.

Q5. Obtain a PDA to accept a string of balanced parentheses. The parentheses to be considered are (,) , [,].

Answer :

For Answer refer to example - 4 , Page No : 6.15.

Q6. Obtain a PDA to accept the language $L = \{ w \mid w \in (a, b)^* \text{ and } n_a(w) > n_b(w) \}$.

Answer :

For Answer refer to example - 5 , Page No : 6.17.

Q7. Obtain a PDA to accept the language $L = \{ a^n b^{2n} \mid n \geq 1 \}$.

Answer :

For Answer refer to example - 6 , Page No : 6.18.

Q8. Obtain a PDA to accept the language $L = \{ ww^R \mid w \in (a+b)^* \}$.

Answer :

For Answer refer to example - 7 , Page No : 6.20.

Q9. Construct a PDA which accepts the set of strings over $\{a, b\}$ with equal number of *a*'s and *b*'s such that all *a*'s and *b*'s are consecutive.

Answer :

For Answer refer to example - 8 , Page No : 6.23.

Q10. Construct a PDA, which accepts $L = \{a^n c^m b^n : m, n \geq 1\}$.

Answer :

For Answer refer to example - 9 , Page No : 6.24.

Q11. Construct a PDA *M*, which accepts $L = \{a^n b^m c^m d^n : m, n \geq 1\}$

Answer :

For Answer refer to example - 10 , Page No : 6.25.

Q12. Let $M = (\{p, q\}, \{0,1\}, \{Z_0, X\}, \delta, p, Z_0, \phi)$ be a PDA where δ is given by following transitions.

$\delta(p, 1, Z_0) = \{(p, XZ_0)\}$,

$\delta(p, \in, Z_0) = \{(p, \in)\}$,

$\delta(p, 1, X) = \{(p, XX)\}$,

$\delta(q, 1, X) = \{(q, \in)\}$,

$\delta(p, 0, X) = \{(q, X)\}$, *and*

$\delta(q, 0, Z_0) = \{(p, Z_0)\}$

(a) What is the language accepted by this PDA by empty store ?

(b) Describe informally the working of the PDA.

Answer :

For Answer refer to example - 11 , Page No : 6.26.

Q13. Let $Q = (\{q_0, q_1\}, \{a, b\}, \{a, b, Z\}, \delta, q_0, Z, \phi)$ be a PDA accepting by empty stack for the language which is the set of all nonempty even palindromes over the set $\{a, b\}$. Below is an incomplete specification of the transition δ. Complete the specification. The top of the stack is assumed to be at the right end of the string representing stack contents.

1. $\delta(q_0, a, Z) = \{(q_0, Za)\}$
2. $\delta(q_0, b, Z) = \{(q_0, Zb)\}$
3. $\delta(q_0, a, a) = \ldots\ldots\ldots$,
4. $\delta(q_0, b, b) = \ldots\ldots\ldots$,
5. $\delta(q_1, a, a) = \{(q_1, \in)\}$
6. $\delta(q_1, b, b) = \{(q_1, \in)\}$ and
7. $\delta(q_1, \in, Z) = \{(q_1, \in)\}$

Answer :

For Answer refer to example - 12 , Page No : 6.27.

Q14. Distinguish between NPDA and DPDA.

Answer :

For Answer refer to Topic : 6.3 , Page No : 6.28.

Q15. Is the PDA to accept the language $L(M) = \{w\, Cw^R \mid w \in (a+b)^*\}$ is deterministic or not ?

Answer :

For Answer refer to example - 1 , Page No : 6.30.

Q16. Is the PDA corresponding to the language $L = \{a^n b^n \mid n \geq 1\}$ by a final state is deterministic or not ?

Answer :

For Answer refer to example - 2 , Page No : 6.31.

Q17. Is the PDA to accept the language $L(M) = \{w \mid w \in (a+b)^*$ *and* $n_a(w) = n_b(w)$ is deterministic or not ?

Answer :

For Answer refer to example - 3 , Page No : 6.31.

Q18. Give a deterministic PDA for the language $L = \{a^n cb^{2n} : n \geq 1\}$ over the alphabet $\Sigma = \{a, b, c\}$. Specify the acceptance state.

Answer :

For Answer refer to example - 4 , Page No : 6.32

Q19. If $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, p_1, Z_1, \phi)$ is a PDA accepting CFL L by empty store then there exists PDA $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, p_2, Z_2, \{q_f\})$ which accepts L by final state.

Answer :

For Answer refer to Topic : 6.4.1 , Page No : 6.33.

Q20. Consider a nondeterministic PDA $M_1 = (\{q_0\}, \{a,b\}, \{a,b,S\}, \delta, q_0, S, \phi)$ which accepts the language $L = \{a^n b^n : n \geq 1\}$ by empty store, where δ is defined as follows :

$\delta(q_0, \in, S) = \{(q_0, ab), (q_0, aSb)\}$ (Two possible moves),

$\delta(q_0, a, a) = \{(q_0, \in)\}$, and $\delta(q_0, b, b) = \{(q_0, \in)\}$

Construct an equivalent PDA M_2 which accepts L in final state and check whether string $w = aabb$ is accepted or not ?

Answer :

For Answer refer to example , Page No : 6.35.

Q21. If $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_0, Z_0, F)$ is a PDA accepting CFL L by final state then there exists PDA $M_2 = (Q_2, \Sigma, \Gamma, \delta_2, q_0, Z_0, \phi)$ which accepts L by empty store.

Answer :

For Answer refer to Topic : 6.4.2 , Page No : 6.37.

Q22. Consider a PDA $M_1 = (\{q_0, q_1, q_2\}, \{a,c\}, \{a, Z_0\}, \delta_1, q_0, Z_0, \{q_2\})$, convert it into PDA M_2 whose acceptance is by empty store. Check the acceptability of string $w = aacaa$.

The transition function δ_1 is defined as follows :

$\delta_1(q_0, a, Z_0) = \{(q_0, aZ_0)\}$,

$\delta_1(q_0, a, a) = \{(q_0, aa)\}$,

$\delta_1(q_0, c, a) = \{(q_1, a)\}$

$\delta_1(q_1, a, a) = \{(q_1, \in)\}$ and

$\delta_1(q_1, \in, Z_0) = \{(q_2, Z_0)\}$

Answer :

For Answer refer to example, Page No : 6.39.

Q23. Explain procedure to construct PDA from CFG.

Answer :

For Answer refer to Topic : 6.5.1, Page No : 6.40.

Q24. For the grammar

S → aABC
A → aB | a
B → bA|b
C → a

Obtain the corresponding PDA

Answer :

For Answer refer to example - 1 , Page No : 6.41.

Q25. Construct PDA M equivalent to the CFG $S \rightarrow 0BB$, $B \rightarrow 1S \mid 0S \mid 0$ and check whether 010000 is in $N(M)$ or not ?

Answer :

For Answer refer to example - 2 , Page No : 6.43.

Q26. Explain procedure to construct CFG from PDA.

Answer :

For Answer refer to Topic : 6.5.2, Page No : 6.44.

Q27. Consider the PDA $M = (\{q_0, q_1\}, \{a,b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$ accepting the language $L = \{a^n b^m a^n : m, n \geq 1\}$, which has the following transition function.

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$,
$\delta(q_0, a, a) = \{(q_0, aa)\}$,
$\delta(q_0, b, a) = \{(q_1, a)\}$,
$\delta(q_1, b, a) = \{(q_1, a)\}$,
$\delta(q_1, a, a) = \{(q_1, \in)\}$, *and*
$\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$

Construct a CFG G which generates the same language.

Answer :

For Answer refer to example - 1 , Page No : 6.44.

Q28. Construct CFG G for the PDA given as follows :

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$
$\delta(q_0, a, a) = \{(q_0, aa)\}$
$\delta(q_0, c, a) = \{(q_0, a)\}$

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

$\delta(q_0, \in, Z_0) = \{(q_0, \in)\}$

Answer :

For Answer refer to example - 2 , Page No : 6.46.

Q29. Let $M = (\{q_0, q_1\}, \{a, b\}, \{c, Z_0\}, \delta, q_0, Z_0, \phi)$ is a PDA and δ is defined as follows:

$\delta(q_0, a, Z_0) = \{(q_0, cZ_0)\}$,

$\delta(q_0, a, c) = \{(q_0, cc)\}$,

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

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

$\delta(q_1, \in, c) = \{(q_1, \in)\}$,

$\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$

Construct CFG G generating $N(M)$.

Answer :

For Answer refer to example - 3 , Page No : 6.46.

Q30. Let L_1 be a context - free language and L_2 be a regular language. Then show that $L_1 \cap L_2$ is context - free.

Answer :

For Answer refer to Theorem , Page No : 6.49.

OBJECTIVE TYPE QUESTIONS

1. Choose the correct statements :
   (a) the power of NPDA and DPDA are same
   (b) the power of DFA and NDFA are different
   (c) the power of DFA and NDFA are almost same
   (d) None of the above

2. A PDA behaves like an FA when the number of auxiliary memory it has is
   (a) 2 or more (b) 0 (c) 1 or more (d) none

3. $A=(\{q_0,q_1,q_f\}),\{a,b,c\},\{a,b,Z_0\},\delta,q_0\ Z_0,\{q_f\})$ is a PDA, where ϕ is defined as

   $\delta(q_0,a,Z_0)=\{(q_0,aZ_0)\}$, $\delta(q_0,b,Z_0)=\{(q_0,bZ_0)\}$

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

   $\delta(q_0,a,b)=\{(q_0,ab)\}$, $\delta(q_0,b,b)=\{(q_0,bb)\}$

   $\delta(q_0,c,a)=\{(q_1,a)\}$, $\delta(q_0,c,b)=\{(q_1,b)\}$, $\delta(q_0,c,Z_0)=\{(q_1,Z_0)\}$

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

   $\delta(q_1,\in,Z_0)=\{(q_f,Z_0)\}$

   Match the ID that the PDA is in after the strings listed on the LHS are processed by the PDA.

(i) abcba	(A)	$(q_0,\in,babaZ_0)$
(ii) abcb	(B)	$(q_1,\in,aZ_0)$
(iii) acba	(C)	(q_1,ba,aZ_0)
(iv) abab	(D)	$(q_f,\in,Z_0)$

   (a) D C A B (b) D C B A
   (c) D B C A (d) A B C D

4. A pda is said to be deterministic if -
   (a) $\delta(q,a,Z)$ is either empty or singleton or $\delta(q,\in,Z)\neq\phi$ implies $\delta(q,a,Z)=\phi$
   (b) $\delta(q,a,Z)$ is either empty or singleton & $\delta(q,\in,Z)=\phi$ implies $\delta(q,a,Z)=\phi$
   (c) $\delta(q,a,Z)$ is either empty or singleton & $\delta(q,\in,Z)\neq\phi$ implies $\delta(q,a,Z)=\phi$
   (d) $\delta(q,a,Z)$ is either empty or singleton or $\delta(q,\in,Z)=\phi$ implies $\delta(q,a,Z)\neq\phi$

5. Let Ld be the set of all languages accepted by a PDA by final states and Le the set of all languages accepted by empty stack. Which of the following is true?
   (a) $L_d\supset L_e$ (b) $L_d=L_e$
   (c) Both (d) None of the above.

6. Pushdown automata can recognize -----

(a) all regular languges, some nonregular languages, all context-free languages, and all non-context -free languages.

(b) all regular languages, some nonregular languages, all context-free languages, and some non-context - free languages.

(c) all regular languages, all nonregular languages, all context-free languages.

(d) all regular languages, some nonregular languages, all context-free languages.

7. Given two PDAs M & M' :

where $M = \langle S, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ and $M' = \langle S', \Sigma, \Gamma', \delta', q'_0, Z_0', F' \rangle$

Which of the following conditions hold if $L(M) = L(M')$.

(i) There is a one-to-one correspondance between accepting computations of M and M'

(ii) If M has no Λ – moves, then M' has no Λ – moves ; If M is unambiguous, then M' is unambiguous.

(iii) For all $p \in S'$, all $a \in \Sigma U\{\varepsilon\}$, all $Z \in \Gamma'$, if $(q, y) \in d\delta'(p, a, Z)$ then $q <> q_0'$ and $|y| =< 2$

(a) Only (ii). (b)Only (i) & (ii)

(c) Only (i) (d) All of the above.

8. Running time of a finite automata (like NPDA) for an input string of length n is

(a) Can be anything depending on the automata (b) Exponential in n

(c) Polynomial in n (d) Linear in n

9. Which of the following is false :

(a) Every CFL corresponds to a NPDA

(b) Ever NPDA corresponds to a CFL

(c) Both (a) and (b)

(d) None

10. Which of the following languages not is accepted by a NPDA

(a) $a^n b^{2n}$ (b) wcw^r (c) ww (d) ww^r

11. Let L_1 be the set of the languages accepted by a NPDA and L_2 be the set of context free languages. Then :

(a) $L_2 \subseteq L_1$ (b) $L_1 = L_2$ (c) $L_1 \subseteq L_2$ (d) none

12. If L is N(M) for some PDA M, the L is a

(a) Regular grammar (b) Context sensitive language

(c) Context free language (d) none.

13. If L is N(M) for some PDA M, then L is a :

(a) RE (b) DCFL (c) CFL (d) none

14. Every regular set accepted by FA with n states is accepted by a DPDA with_____ states and n pushdown symbols
(a) n (b) 1 (c) 2 (d) none

15. A regular set accepted by DFA with n states is accepted to final state by a DPDA with n states and at least_____ pushdown symbols
(a) 3 (b) 1 (c) 2 (d) none

16. Match the following :

(a) Type 0 grammar	(a) A grammar $G=(V,T,S,P)$ in which all productions in P are of the form $A \to V$ where $A \in V$ and $X \in (V \cup T)^*$
(b) Type 1 grammar	(b) A grammar $G=(V,T,S,P)$ in which all productions in P are of the form $x \in y$ where $x,y \in (V \cup T)^+$ and $y \in (V \cup T)^*$ and $\|x\| \leq \|y\|$
(c) Type 2 grammar	(c) A grammar $G=(V,T,S,P)$ in which all productions in P are of the form $x \in y$ where $x \in (V \cup T)^+$ and $y((V(T)^*$
(d) Type 3 grammar	(d) A grammar $G=(V,T,S,P)$ in which all productions in P are of the form $A \in a$ and $A \in aB$ where $a \in T^*$ and $A,B \in V$

17. Which one of the following is true about PDA's
(a) $\delta: S \times (\Sigma \cup \{\in\}) \times \Gamma \to S \times \Gamma^*$
(b) PDA has an auxillary memory in the form of a stack.
(c) PDA is represented by 7-tuple $(S, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
(d) all of the above

18. If for a language L we can find a RE then definitely (choose most appropriate) :
(a) we can draw DPDA for L (b) we can draw NPDA for L
(c) we can't draw NPDA for L (d) none.

19. Which of the following doesn't hold (in ID of pushdown automata) ?
(a) If $(q,x,WU) \to (t, \in, U)$ then $(q,x,W) \to (t,y,Z)$
(b) If $(q,x,W) \to (t, \in, Z)$ then $(q,x,WU) \to (t,y,U)$
(c) If $(q,xy,W) \to (t,y,Z)$ then $(q,x,W) \to (t, \in, Z)$
(d) If $(q,x,W) \to (t, \in, Z)$ then $(q,xy,W) \to (t,y,Z)$

20. For Pushdown Automatas, which of the following is true.

(a) If $(q,x,a)|^{*}-(q',\in,\gamma)$, the for every $\beta\in\Gamma^{*}$, $(q,x,\alpha\beta)|^{*}-(q',\in,\gamma\beta)$

(b) If $(q_1,x,a)|^{*}-(q_2,\in,\gamma\beta)$, then for every $y\in\Sigma^{*}$, $(q_1,xy,a)|^{*}-(q_2,y,\beta)$

(c) Both a and b.

(d) Only one out of a or b

21. For Pushdown automatas, which of the following is true.

(a) If $(q,x,a)|^{*}-(q',\in,\gamma)$, the for every $\beta\in\Gamma^{*}$,, $(q,x,\alpha\beta)|^{*}-(q',\in,\gamma\beta)$

(b) If $(q_1,x,\alpha)^{*}|-(q_2,\in,\beta)$, then for every $y\in\Sigma^{*}$, $(q_1,xy,a)|^{*}-(q_2,y,\beta)$

(c) Both a and b.

(d) Only one out of a or b

22. If S is the number of states in NDFA then equivalent DFA can have maximum of

(a) $2^{S}-1$ states (b) $_2S$ states (c) S-1 states (d) S states

23. The language $\{ww^{r}\}$ is

(a) Accepted by a NPDA not by a DPDA

(b) Accepted by a DPDA not by a NPDA

(c) Cannot say.

(d) None

24. Which of the following can not be accepted by Deterministic PDA

(a) $a^{n}b^{n}n\geq 1\cup a^{m}b^{2m}m\geq 1$ (b) $_{wcw^{I}}$

(c) $a^{n}b^{n}n\geq 1$ (d) none

25. A PDA A is deterministic if :

(i) $\delta(q,\in,z)\neq\phi$ implies $\delta(q,a,z)=\phi\forall a\in Z$ (ii) $\delta(q,a,z)$ is

(a) singleton (b) empty

(c) either(a) or (b) (d) none

26. Which of the following statements is false ?

(a) The class of sets accepted by pushdown automata properly includes the regular sets.

(b) Inherently ambiguous languages can be modeled LR grammars.

(c) A language is CFL iff it can be accepted by a Non deterministic PDA.

(d) A language is LR iff it can be accepted by a Deterministic PDA.

27. A pda is said to be deterministic if :
(a) $\delta(q,a,Z)$ is either empty or singleton or & $(q,\in,Z) \neq \phi$ implies $\delta(q,a,Z) \neq \phi$
(b) $\delta(q,a,Z)$ is either empty or singleton & $(q,\in,Z) = \phi$ implies $\delta(q,a,Z) \neq \phi$
(c) $\delta(q,a,Z)$ is either empty or singleton & $\delta(q,\in,Z) \neq \phi$ implies $\delta(q,a,Z) = \phi$
(d) $\delta(q,a,Z)$ is either empty or singleton or $(q,\in,Z) = \phi$ implies $\delta(q,a,Z) \neq \phi$

28. Which of the following is accepted by an NPDA and not DPDA
(a) String ending with a particular alphabet
(b) All strings in which a given symbol is present atleast twice
(c) Even palindromes (i.e. palindromes made up if even no of symbols)
(d) None

29. Given two PDAs M & M' :
where $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ and
$M' = \langle Q', \Sigma, \Gamma', d', q_0', Z_0', F' \rangle$
Which of the following conditions hold if $L(M) = L(M')$.
(i) There is a one-to-one correspondance between accepting computations of M and M'
(ii) If M has no $\in$–moves, then M' has no $\in$–moves; If M is unambiguous, then M' is unambiguous.
(iii) For all $p \in Q'$, all $a\Sigma \cup \{\in\}$, all $Z \in \Gamma'$, if $(q,y) \in \delta'(p,a,Z)$, then $q <> q'0$ and $|y| =< 2$
(a) Only (ii) (b) Only (i) & (ii)
(c) Only (i) (d) All of the above.

30. For Deterministic Context Free languages, which of the following hold?
(i) The complement of a DCFL is a DCFL.
(ii) Let L be a DCFL and R is a regular set. Then L/R is a DCFL.
(iii) DCFL's are closed under intersection with a regular set.
(a) (i), (ii) and (iii) (b) (i) and (ii)
(c) (i) and (iii) (d) None of the above.

31. Complement L_1 is :
(a) RL (b) deterministic (c) CFL (d) None

32. $L_1 L_2$ is
(a) RL (b) deterministic (c) CFL (d) None

33. If L_1 & L_2 are two deterministic languages $L_1 \cup L_2$ is
(a) RL (b) deterministic (c) CFL (d) None

34. Let $A=(\{q_0,q_1,q_f\},\{a,b,c\},\ \{a,b,Z_0\},\delta,q_0,\ Z_0,\{q_f\})$ is a PDA, where ϕ is defined as

$\delta(q_0,a,Z_0)=\{(q_0,aZ_0)\}$, $\delta(q_0,b,Z_0)=\{(q_0,bZ_0)\}$

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

$\delta(q_0,a,b)=\{(q_0,ab)\}$, $\delta(q_0,b,b)=\{(q_0,bb)\}$

$\delta(q_0,c,a)=\{(q_1,a)\}$, $\delta(q_0,c,b)=\{(q_1,b)\}$, $\delta(q_0,c,Z_0)=\{(q_1,Z_0)\}$

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

$\delta(q_1,\in,Z_0)=\{(q_f,Z_0)\}$

Then

(a) A is NPDA but not DPDA (b) A is a NPDA

(c) A is a DPDA (d) none

ANSWER KEY

1.(c)	2.(b)	3.(c)	4.(c)	5.(a)	6.(d)	7.(b)	8.(d)	9.(c)	10.(c)
11.(b)	12.(c)	13.(c)	14.(b)	15.(b)	16.a-c, b-b, c-a, d-d			17.(d)	18.(a)
19.(a)	20.(c)	21.(c)	22.(b)	23.(a)	24.(a)	25.(c)	26.(c)	27.(c)	28.(c)
29.(c)	30.(c)	31.(d)	32.(b)	33.(c)	34.(c)				

Formal Languages And Automata Theory

UNIT 4

COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF AERONAUTICAL ENGINEERING

DUNDIGAL, HYDERABAD - 500 043

7

TURING MACHINES

After going through this chapter, you should be able to understand :

- Turing Machine
- Design of TM
- Computable functions
- Recursively Enumerable languages
- Church's Hypothesis & Counter machine
- Types of Turing Machines

7.1 INTRODUCTION

The Turing machine is a generalized machine which can recognize all types of languages viz, regular languages (generated from regular grammar), context free languages (generated from context free grammar) and context sensitive languages (generated from context sensitive grammar). Apart from these languages, the Turing machine also accepts the language generated from unrestricted grammar. Thus, Turing machine can accept any generalized language. This chapter mainly concentrates on building the Turing machines for any language.

7.2 TURING MACHINE MODEL

The Turing machine model is shown in below figure . It is a finite automaton connected to read - write head with the following components :

- Tape
- Read - write head
- Control unit

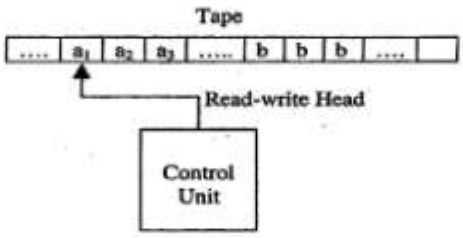


FIGURE : Turing machine model

Tape : It is a temporary storage and is divided into cells. Each cell can store the information of only one symbol. The string to be scanned will be stored from the left most position on the tape. The string to be scanned should end with infinite number of blanks.

Read - write head : The read - write head can read a symbol from where it is pointing to and it can write into the tape to where the read - write head points to.

Control Unit : The reading / writing from / to the tape is determined by the control unit. The different moves performed by the machine depends on the current scanned symbol and the current state. The read - write head can move either towards left or right i.e., movement can be on both the directions. The various moves performed by the machine are :

1. Change of state from one state to another state
2. The symbol pointing to by the read - write head can be replaced by another symbol.
3. The read - write head may move either towards left or towards right.

The Turing machine can be represented using various notations such as

- Transition table
- Instantaneous description
- Transition diagram

7.2.1 Transition Table

The table below shows the transition table for some Turing machine. Later sections describe how to obtain the transition table.

δ	Tape Symbols (Γ)				
States	a	b	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	(q_1, a, R)	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	(q_2, a, L)	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

Note that for each state q, there can be a corresponding entry for the symbol in Γ. In this table the symbols a and b are input symbols and can be denoted by the symbol Σ. Thus $\Sigma \subseteq \Gamma$ excluding the symbol B. The symbol B indicates a blank character and usually the string ends with infinite number of B's i. e., blank characters. The undefined entries indicate that there are no - transitions defined or there can be a transition to dead state. When there is a transition to the dead state, the machine halts and the input string is rejected by the machine. It is clear from the table that

$$\delta : Q \times \Gamma \text{ to } (Q \times \Gamma \times \{L, R\})$$

where $Q = \{q_0, q_1, q_2, q_3, q_4\}$; $\Sigma = \{a, b\}$

$\Gamma = \{a, b, X, Y, B\}$

q_0 is the initial state; B is a special symbol indicating blank character

$F = \{q_4\}$ which is the final state.

Thus, a Turing Machine M can be defined as follows.

Definition : The Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where

Q is set of finite states

Σ is set of input alphabets

Γ is set of tape symbols

δ is transition function $Q \times \Gamma$ to $(Q \times \Gamma \times \{L, R\})$

q_0 is the initial state

B is a special symbol indicating blank character

$F \subseteq Q$ is set of final states.

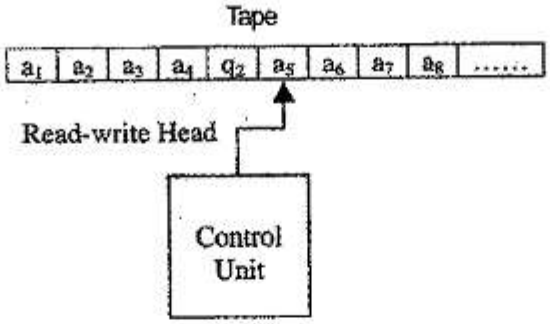
7.2.2 Instantaneous description (ID)

Unlike the ID described in PDA, in Turing machine (TM), the ID is defined on the whole string (not on the string to be scanned) and the current state of the machine.

Definition :

An ID of TM is a string in $\alpha q \beta$, where q is the current state, $\alpha \beta$ is the string made from tape symbols denoted by Γ i. e., α and $\beta \in \Gamma^*$. The read - write head points to the first character of the substring β. The initial ID is denoted by $q\alpha\beta$ where q is the start state and the read - write head points to the first symbol of α from left. The final ID is denoted by $\alpha\beta qB$ where $q \in F$ is the final state and the read - write head points to the blank character denoted by B.

Example : Consider the snapshot of a Turing machine



In this machine, each $a_i \in \Gamma$ (i. e., each a_i belongs to the tape symbol). In this snapshot, the symbol a_5 is under read - write head and the symbol towards left of a_5 i. e., q_2 is the current state. Note that, in the Turing machine, the symbol immediately towards left of the read - write head will be the current state of the machine and the symbol immediately towards right of the state will be the next symbol to be scanned. So, in this case an ID is denoted by

$$a_1a_2a_3a_4q_2a_5a_6a_7a_8.....$$

where the substring $a_1a_2a_3a_4$ towards left of the state q_2 is the left sequence, the substring $a_5a_6a_7a_8$..... towards right of the state q_2 is the right sequence and q_2 is the current state of the machine. The symbol a_5 is the next symbol to be scanned.

Assume that the current ID of the Turing machine is $a_1a_2a_3a_4q_2a_5a_6a_7a_8$..... as shown in snapshot of example.

Suppose, there is a transition $\delta(q_2, a_5) = (q_3, b_1, R)$

It means that if the machine is in state q_2 and the next symbol to be scanned is a_5, then the machine enters into state q_3 replacing the symbol a_5 by b_1 and R indicates that the read - write head is moved one symbol towards right. The new configuration obtained is

$$a_1a_2a_3a_4b_1q_3a_6a_7a_8.....$$

This can be represented by a move as $a_1a_2a_3a_4q_2a_5a_6a_7a_8.... \vdash a_1a_2a_3a_4b_1q_3a_6a_7a_8.....$

Similarly if the current ID of the Turing machine is $a_1a_2a_3a_4q_2a_5a_6a_7a_8.....$

and there is a transition

$$\delta(q_2, a_5)=(q_1, c_1, L)$$

means that if the machine is in state q_2 and the next symbol to be scanned is a_5, then the machine enters into state q_1 replacing the symbol a_5 by c_1 and L indicates that the read - write head is moved one symbol towards left. The new configuration obtained is

$$a_1a_2a_3q_1a_4c_1a_6a_7a_8.....$$

This can be represented by a move as $a_1a_2a_3a_4q_2a_5a_6a_7a_8\ldots. \vdash a_1a_2a_3q_1a_4a_5a_6a_7a_8\ldots.$

This configuration indicates that the new state is q_1, the next input symbol to be scanned is a_4. The actions performed by TM depends on

1. The current state.
2. The whole string to be scanned
3. The current position of the read - write head

The action performed by the machine consists of

1. Changing the states from one state to another
2. Replacing the symbol pointed to by the read - write head
3. Movement of the read - write head towards left or right.

7.2.3 The move of Turing Machine M can be defined as follows

Definition : Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a TM. Let the ID of M be $a_1a_2a_3\ldots\ldots a_{k-1}qa_ka_{k+1}\ldots\ldots a_n$ where $a_j \in \Gamma$ for $1 \le j \le n-1$, $q \in Q$ is the current state and a_k as the next symbol to scanned. If there is a transition $\delta(q, a_k) = (p, b, R)$ then the move of machine M will be $a_1a_2a_3\ldots\ldots a_{k-1}qa_ka_{k+1}\ldots\ldots a_n \vdash a_1a_2a_3\ldots\ldots a_{k-1}bpa_{k+1}\ldots\ldots a_n$

If there is a transition $\delta(q, a_k) = (p, b, L)$ then the move of machine M will be

$$a_1a_2a_3\ldots\ldots a_{k-1}qa_ka_{k+1}\ldots\ldots a_n \vdash a_1a_2a_3\ldots\ldots a_{k-2}pa_{k-1}ba_{k+1}\ldots\ldots a_n$$

7.2.4 Acceptance of a language by TM

The language accepted by TM is defined as follows.

Definition :

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a TM. The language L(M) accepted by M is defined as

$$L(M) = \{w \mid q_0w \vdash^* \alpha_1\ p\ \alpha_2 \text{ where } w \in \Sigma^*, p \in F \text{ and } \alpha_1, \alpha_2 \in \Gamma^*\}$$

i.e., set of all those words w in Σ^* which causes M to move from start state q_0 to the final state p. The language accepted by TM is called recursively enumerable language.

The string w which is the string to be scanned, should end with infinite number of blanks. Initially, the machine will be in the start state q_0 with read - write head pointing to the first symbol of w from left. After some sequence of moves, if the Turing machine enters into the final state and halts, then we say that the string w is accepted by Turing machine.

7.2.5 Differences between TM and PDA

Push Down Automa :

1. A PDA is a nondeterministic finite automaton coupled with a stack that can be used to store a string of arbitrary length.
2. The stack can be read and modified only at its top.
3. A PDA chooses its next move based on its current state, the next input symbol and the symbol at the top of the stack.
4. There are two ways in which the PDA may be allowed to signal acceptance. One is by entering an accepting state, the other by emptying its stack.
5. ID consisting of the state, remaining input and stack contents to describe the "current condition" of a PDA.
6. The languages accepted by PDA's either by final state or by empty stack, are exactly the context - free languages.
7. A PDA languages lie strictly between regular languages and CSL's.

Turing Machines :

1. The TM is an abstract computing machine with the power of both real computers and of other mathematical definitions of what can be computed.
2. TM consists of a finite - state control and an infinite tape divided into cells.
3. TM makes moves based on its current state and the tape symbol at the cell scanned by the tape head.
4. The blank is one of tape symbols but not input symbol.
5. TM accepts its input if it ever enters an accepting state.
6. The languages accepted by TM's are called Recursively Enumerable (RE) languages.
7. Instantaneous description of TM describes current configuration of a TM by finite - length string.
8. Storage in the finite control helps to design a TM for a particular language.
9. A TM can simulate the storage and control of a real computer by using one tape to store all the locations and their contents.

7.3 CONSTRUCTION OF TURING MACHINE (TM)

In this section, we shall see how TMs can be constructed.

Example 1 : Obtain a Turing machine to accept the language $L = \{ 0^n 1^n \mid n \geq 1 \}$.

Solution : Note that n number of 0's should be followed by n number of 1's. For this let us take an example of the string $w = 00001111$. The string w should be accepted as it has four zeroes followed by equal number of 1's.

General Procedure :

Let q_0 be the start state and let the read - write head points to the first symbol of the string to be scanned. The general procedure to design TM for this case is shown below :

1. Replace the left most 0 by X and change the state to q_1 and then move the read - write head towards right. This is because, after a zero is replaced, we have to replace the corresponding 1 so that number of zeroes matches with number of 1's.
2. Search for the leftmost 1 and replace it by the symbol Y and move towards left (so as to obtain the leftmost 0 again). Steps 1 and 2 can be repeated.

Consider the situation

XX00YY11

↑

q_0

where first two 0's are replaced by Xs and first two 1's are replaced by Ys. In this situation, the read - write head points to the left most zero and the machine is in state q_0. With this as the configuration, now let us design the TM.

Step 1 : In state q_0, replace 0 by X, change the state to q_1 and move the pointer towards right. The transition for this can be of the form

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

The resulting configuration is shown below.

XXX0YY11

↑

q_1

Step 2 : In state q_1, we have to obtain the left - most 1 and replace it by Y. For this, let us move the pointer to point to leftmost one. When the pointer is moved towards 1, the symbols encountered may be 0 and Y. Irrespective what symbol is encountered, replace 0 by 0, Y by Y, remain in state q_1 and move the pointer towards right. The transitions for this can be of the form

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

When these transitions are repeatedly applied, the following configuration is obtained.

XXX0YY11

↑

q_1

Step 3 : In state q_1, if the input symbol to be scanned is a 1, then replace 1 by Y, change the state to q_2 and move the pointer towards left. The transition for this can be of the form

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

and the following configuration is obtained.

XXX0YYY1

↑

q_2

Note that the pointer is moved towards left. This is because, a zero is replaced by X and the corresponding 1 is replaced by Y. Now, we have to scan for the left most 0 again and so, the pointer was move towards left.

Step 4 : Note that to obtain leftmost zero, we need to obtain right most X first. So, we scan for the right most X. During this process we may encounter Y's and 0's . Replace Y by Y, 0 by 0, remain in state q_2 only and move the pointer towards left. The transitions for this can be of the form

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

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

The following configuration is obtained

XXX0YYY1

↑

q_2

Step 5 : Now, we have obtained the right most X. To get leftmost 0, replace X by X, change the state to q_0 and move the pointer towards right. The transition for this can be of the form

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

and the following configuration is obtained

XXX0YYY1

↑

q_0

Now, repeating the steps 1 through 5, we get the configuration shown below :

XXXXYYYY

↑

q_0

Step 6 : In state q_0, if the scanned symbol is Y, it means that there are no more 0's. If there are no zeroes we should see that there are no 1's. For this we change the state to q_3, replace Y by Y and move the pointer towards right. The transition for this can be of the form

$$\delta(q_0, Y) = (q_3, Y, R)$$

and the following configuration is obtained

XXXXYYYY

↑

q_3

In state q_3, we should see that there are only Ys and no more 1's. So, as we can replace Y by Y and remain in q_3 only. The transition for this can be of the form

$$\delta(q_3, Y) = (q_3, Y, R)$$

Repeatedly applying this transition, the following configuration is obtained .

XXXXYYYYB

↑

q_3

Note that the string ends with infinite number of blanks and so, in state q_3 if we encounter the symbol B, means that end of string is encountered and there exists n number of 0's ending with n number of 1's. So, in state q_3, on input symbol B, change the state to q_4, replace B by B and move the pointer towards right and the string is accepted. The transition for this can be of the form $\delta(q_3, B) = (q_4, B, R)$

The following configuration is obtained

XXXXYYYYBB

↑

q_4

So, the Turing machine to accept the language $L = \{a^n\ b^n \mid n \geq 1\}$

is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

where

$Q = \{q_0, q_1, q_2, q_3\}$; $\Sigma = \{0, 1\}$; $\Gamma = \{0, 1, X, Y, B\}$

$q_0 \in Q$ is the start state of machine; $B \in \Gamma$ is the blank symbol.

$F = \{q_4\}$ is the final state.

δ is shown below.

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

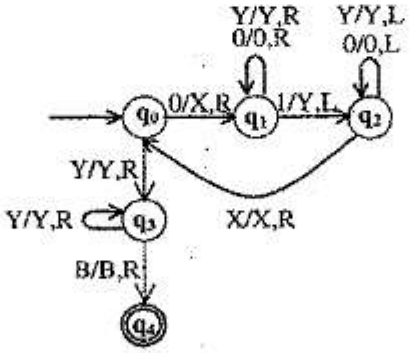
$$\delta(q_1, 0) = (q_1, 0, R)$$

$$\delta(q_1, Y) = (q_1, Y, R)$$
$$\delta(q_1, 1) = (q_2, Y, L)$$
$$\delta(q_2, Y) = (q_2, Y, L)$$
$$\delta(q_2, 0) = (q_2, 0, L)$$
$$\delta(q_2, X) = (q_0, X, R)$$
$$\delta(q_0, Y) = (q_3, Y, R)$$
$$\delta(q_3, Y) = (q_3, Y, R)$$
$$\delta(q_3, B) = (q_4, B, R)$$

The transitions can also be represented using tabular form as shown below.

δ	Tape Symbols (Γ)				
States	0	1	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	$(q_1, 0, R)$	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	$(q_2, 0, L)$	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

The transition table shown above can be represented as transition diagram as shown below :



To accept the string :

The sequence of moves or computations (IDs) for the string 0011 made by the Turing machine are shown below :

Initial ID		
$q_0 0011$	$\vdash X q_1 011$	$\vdash X\,0\,q_1 11$
	$\vdash X q_2 0Y1$	$\vdash q_2 X 0Y1$
	$\vdash X q_0 0Y1$	$\vdash XX q_1 Y1$
	$\vdash XXY q_1 1$	$\vdash XX q_2 YY$
	$\vdash X q_2 XYY$	$\vdash XX q_0 YY$
	$\vdash XXY q_3 Y$	$\vdash XXYY q_3$
	$\vdash XXYYB q_4$	
	(Final ID)	

Example 2 : Obtain a Turing machine to accept the language $L(M) = \{ 0^n 1^n 2^n \mid n \geq 1 \}$

Solution : Note that n number of 0's are followed by n number of 1's which in turn are followed by n number of 2's. In simple terms, the solution to this problem can be stated as follows :

Replace first n number of 0's by X's, next n number of 1's by Y's and next n number of 2's by Z's. Consider the situation where in first two 0's are replaced by X's , next immediate two 1's are replaced by Y's and next two 2's are replaced by Z's as shown in figure 1(a).

XX00YY11ZZ22	XXX0YY11ZZ22	XXX0YY11ZZ22
↑	↑	↑
q_0	q_1	q_1
(a)	(b)	(c)

FIGURE 1 : Various Configurations

Now, with figure 1(a). a as the current configuration, let us design the Turing machine. In state q_0, if the next scanned symbol is 0 replace it by X, change the state to q_1 and move the pointer towards right and the situation shown in figure 1(b) is obtained . The transition for this can be of the form

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

In state q_1, we have to search for the leftmost 1. It is clear from figure 1(b) that, when we are searching for the symbol 1, we may encounter the symbols 0 or Y. So, replace 0 by 0 , Y by Y and move the pointer towards right and remain in state q_1 only. The transitions for this can be of the form

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

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

The configuration shown in figure 1(c) is obtained. In state q_1, on encountering 1 change the state to q_2, replace 1 by Y and move the pointer towards right. The transition for this can be of the form

$$\delta(q_1,1)=(q_2,Y,R)$$

and the configuration shown in figure 2(a) is obtained

XXX0YYY1ZZ22	XXX0YYY1ZZ22	XXX0YYY1ZZZ2
↑	↑	↑
q_2	q_2	q_3
(a)	(b)	(c)

FIGURE 2 : Various Configurations

In state q_2, we have to search for the leftmost 2. It is clear from figure 2(a) that, when we are searching for the symbol 2, we may encounter the symbols 1 or Z. So, replace 1 by 1, Z by Z and move the pointer towards right and remain in state q_2 only and the configuration shown in figure 2(b) is obtained. The transitions for this can be of the form

$$\delta(q_2,1)=(q_2,1,R)$$
$$\delta(q_2,Z)=(q_2,Z,R)$$

In state q_2, on encountering 2, change the state to q_3, replace 2 by Z and move the pointer towards left. The transition for this can be of the form

$$\delta(q_2,2)=(q_3,Z,L)$$

and the configuration shown in figure 2(c) is obtained. Once the TM is in state q_3, it means that equal number of 0's, 1's and 2's are replaced by equal number of X's, Y's and Z's respectively. At this point, next we have to search for the rightmost X to get leftmost 0. During this process, it is clear from figure 2(c) that the symbols such as Z's, 1,s, Y's, 0's and X are scanned respectively one after the other. So, replace Z by Z, 1 by 1, Y by Y, 0 by 0, move the pointer towards left and stay in state q_3 only. The transitions for this can be of the form

$$\delta(q_3,Z)=(q_3,Z,L)$$
$$\delta(q_3,1)=(q_3,1,L)$$
$$\delta(q_3,Y)=(q_3,Y,L)$$
$$\delta(q_3,0)=(q_3,0,L)$$

Only on encountering X, replace X by X, change the state to q_0 and move the pointer towards right to get leftmost 0. The transition for this can be of the form

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

All the steps shown above are repeated till the following configuration is obtained.

XXXXYYYYZZZZ

↑

q_0

In state q_0, if the input symbol is Y, it means that there are no 0's . If there are no 0's we should see that there are no 1's also. For this to happen change the state to q_4, replace Y by Y and move the pointer towards right. The transition for this can be of the form

$$\delta(q_0, Y) = (q_4, Y, R)$$

In state q_4 search for only Y's, replace Y by Y, remain in state q_4 only and move the pointer towards right. The transition for this can be of the form

$$\delta(q_4, Y) = (q_4, Y, R)$$

In state q_4, if we encounter Z, it means that there are no 1's and so we should see that there are no 2's and only Z's should be present. So, on scanning the first Z, change the state to q_5, replace Z by Z and move the pointer towards right. The transition for this can be of the form

$$\delta(q_4, Z) = (q_5, Z, R)$$

But, in state q_5 only Z's should be there and no more 2's. So, as long as the scanned symbol is Z, remain in state q_5, replace Z by Z and move the pointer towards right. But, once blank symbol B is encountered change the state to q_6, replace B by B and move the pointer towards right and say that the input string is accepted by the machine. The transitions for this can be of the form

$$\delta(q_5, Z) = (q_5, Z, R)$$

$$\delta(q_5, B) = (q_6, B, R)$$

where q_6 is the final state.

So, the TM to recognize the language $L = \{ 0^n 1^n 2^n \mid n \geq 1 \}$ is given by

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

where

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$; $\Sigma = \{0, 1, 2\}$

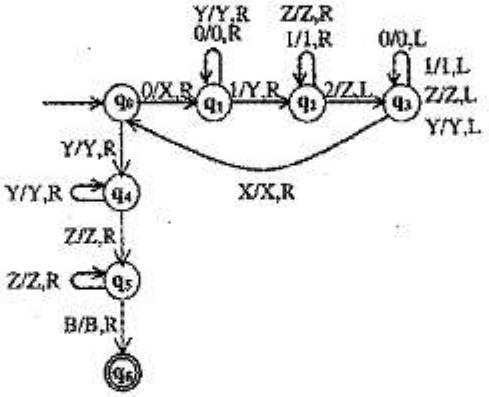
$\Gamma = \{0, 1, 2, X, Y, Z, B\}$; q_0 is the initial state

B is blank character; $F = \{ q_6 \}$ is the final state

δ is shown below using the transition table.

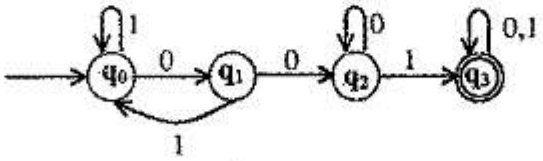
	Γ						
States	0	1	2	Z	Y	X	B
q_0	q_1, X, R				q_4, Y, R		
q_1	q_1, 0, R	q_2, Y,R			q_1, Y,R		
q_2		q_2, 1,R	q_3, Z,L	q_2, Z,R			
q_3	q_3, 0,L	q_3, 1,L		q_3, Z,L	q_3, Y,L	q_0, X,R	
q_4				q_5, Z,R	q_4, Y,R		
q_5				q_5, Z,R			(q_6, B, R)
q_6							

The transition diagram for this can be of the form



Example 3 : Obtain a TM to accept the language $L = \{w \mid w \in (0+1)^*\}$ containing the substring 001.

Solution : The DFA which accepts the language consisting of strings of 0's and 1's having a sub string 001 is shown below :



The transition table for the DFA is shown below :

	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_3
q_3	q_3	q_3

We have seen that any language which is accepted by a DFA is regular. As the DFA processes the input string from left to right in only one direction, TM also processes the input string in only one direction (unlike the previous examples, where the read - write header was moving in both the directions). For each scanned input symbol (either 0 or 1), in whichever state the DFA was in, TM also enters into the same states on same input symbols, replacing 0 by 0 and 1 by 1 and the read - write head moves towards right. So, the transition table for DFA and TM remains same (the format may be different. It is evident in both the transition tables). So, the transition table for TM to recognize the language consisting of 0's and 1's with a substring 001 is shown below :

	0	1	B
q_0	q_1,0, R	q_0, 1, R	-
q_1	q_2, 0, R	q_0, 1, R	-
q_2	q_2, 0, R	q_3, 1, R	-
q_3	q_3, 0, R	q_3, 1, R	q_4, B, R
q_4			

The TM is given by

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

where

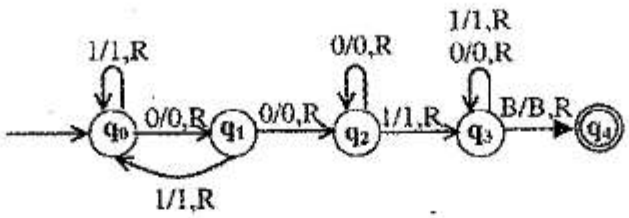
$Q = \{q_0, q_1, q_2, q_3, q_4\}$; $\Sigma = \{0, 1\}$

$\Gamma = \{0, 1\}$; δ – is defined already

q_0 is the initial state ; B blank character

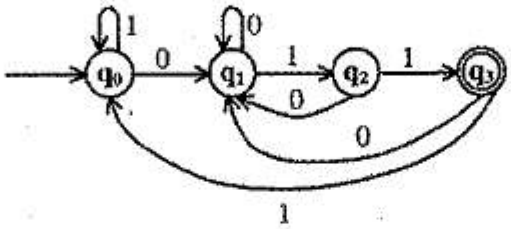
$F = \{q_4\}$ is the final state

The transition diagram for this is shown below.



Example 4 : Obtain a Turing machine to accept the language containing strings of 0's and 1's ending with 011.

Solution : The DFA which accepts the language consisting of strings of 0's and 1's ending with the string 001 is shown below :



The transition table for the DFA is shown below :

δ	0	1
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_1	q_0

We have seen that any language which is accepted by a DFA is regular. As the DFA processes the input string from left to right in only one direction, TM also processes the input string in only one direction. For each scanned input symbol (either 0 or 1), in whichever state the DFA was in, TM also enters into the same states on same input symbols, replacing 0 by 0 and 1 by 1 and the read - write head moves towards right. So, the transition table for DFA and TM remains same (the format may be different. It is evident in both the transition tables). So, the transition table for TM to recognize the language consisting of 0's and 1's ending with a substring 001 is shown below :

δ	0	1	B
q_0	q_1,0, R	q_0, 1, R	-
q_1	q_1, 0, R	q_2, 1, R	-
q_2	q_1, 0, R	q_3, 1, R	-
q_3	q_1, 0, R	q_0, 1, R	q_4, B, R
q_4	-	-	-

The TM is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
where

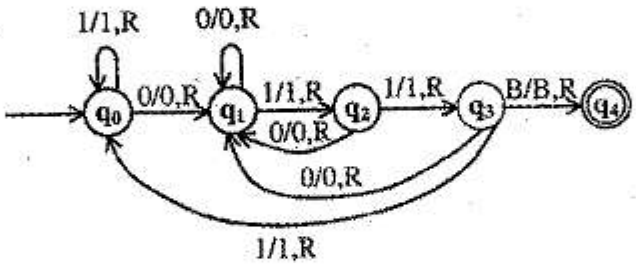
$Q = \{ q_0, q_1, q_2, q_3 \}$; $\Sigma = \{0, 1\}$; $\Gamma = \{0, 1\}$

δ – is defined already

q_0 is the initial state ; B does not appear

F = { q_4 } is the final state

The transition diagram for this is shown below :

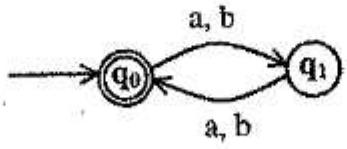


Example 5 : Obtain a Turing machine to accept the language

$L = \{ w | w \text{ is even and } \Sigma = \{ a, b \} \}$

Solution :

The DFA to accept the language consisting of even number of characters is shown below.



The transition table for the DFA is shown below :

	a	b
q_0	q_1	q_1
q_1	q_0	q_0

We have seen that any language which is accepted by a DFA is regular. As the DFA processes the input string from left to right in only one direction, TM also processes the input string in only one direction. For each scanned input symbol (either a or b), in whichever state the DFA was in, TM also enters into the same states on same input symbols, replacing a by a and b by b and the read - write head moves towards right. So, the transition table for DFA and TM remains same (the format may be different). So, the transition table for TM to recognize the language consisting of a's and b's having even number of symbols is shown below :

δ	a	b	B
q_0	q_1, a, R	q_1, b, R	q_2, B, R
q_1	q_0, a, R	q_0, b, R	-
q_2	-	-	-

The TM is given by

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

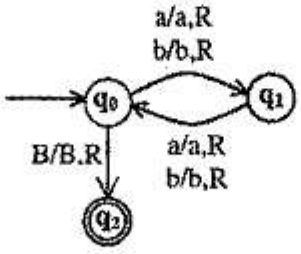
where

$Q = \{ q_0, q_1 \}$; $\Sigma = \{a, b\}$; $\Gamma = \{a, b\}$

δ – is defined already ; q_0 is the initial state

B does not appear ; $F = \{ q_2 \}$ is the final state

The transition diagram of TM is given by



Example 6 : Obtain a Turing machine to accept a palindrome consisting of a's and b's of any length.

Solution : Let us assume that the first symbol on the tape is blank character B and is followed by the string which in turn ends with blank character B. Now, we have to design a Turing machine which accepts the string, provided the string is a palindrome. For the string to be a palindrome, the first and the last character should be same. The second character and last but one character in the string should be same and so on. The procedure to accept only string of palindromes is shown below. Let q0 be the start state of Turing machine.

Step 1 : Move the read - write head to point to the first character of the string. The transition for this can be of the form $\delta(q_0, B) = (q_1, B, R)$

Step 2 : In state q_1, if the first character is the symbol a, replace it by B and change the state to q_2 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_1, a) = (q_2, B, R)$$

Now, we move the read - write head to point to the last symbol of the string and the last symbol should be a. The symbols scanned during this process are a's, b's and B. Replace a by a, b by b and move the pointer towards right. The transitions defined for this can be of the form

$$\delta(q_2, a) = (q_2, a, R)$$

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

But, once the symbol B is encountered, change the state to q_3, replace B by B and move the pointer towards left. The transition defined for this can be of the form

$$\delta(q_2, B) = (q_3, B, L)$$

In state q_3, the read - write head points to the last character of the string. If the last character is a, then change the state to q_4, replace a by B and move the pointer towards left. The transitions defined for this can be of the form

$$\delta(q_3, a) = (q_4, B, L)$$

At this point, we know that the first character is a and last character is also a. Now, reset the read - write head to point to the first non blank character as shown in step5.

In state q_3, if the last character is B (blank character), it means that the given string is an odd palindrome. So, replace B by B change the state to q_7 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_3, B) = (q_7, B, R)$$

Step 3 : If the first character is the symbol b, replace it by B and change the state from q_1 to q_5 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_1, b) = (q_5, B, R)$$

Now, we move the read - write head to point to the last symbol of the string and the last symbol should be b. The symbols scanned during this process are a's, b's and B. Replace a by a, b by b and move the pointer towards right. The transitions defined for this can of the form

$$\delta(q_5, a) = (q_5, a, R)$$
$$\delta(q_5, b) = (q_5, b, R)$$

But, once the symbol B is encountered, change the state to q_6, replace B by B and move the pointer towards left. The transition defined for this can be of the form

$$\delta(q_5, B) = (q_6, B, L)$$

In state q_6, the read - write head points to the last character of the string. If the last character is b, then change the state to q_6, replace b by B and move the pointer towards left. The transitions defined for this can be of the form

$$\delta(q_6, b) = (q_4, B, L)$$

At this point, we know that the first character is b and last character is also b. Now, reset the read - write head to point to the first non blank character as shown in step 5.

In state q_6, If the last character is B (blank character), it means that the given string is an odd palindrome. So, replace B by B, change the state to q_7 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_6, B) = (q_7, B, R)$$

Step 4 : In state q_1, if the first symbol is blank character (B), the given string is even palindrome and so change the state to q_7, replace B by B and move the read - write head towards right. The transition for this can be of the form

$$\delta(q_1, B) = (q_7, B, R)$$

Step 5 : Reset the read - write head to point to the first non blank character. This can be done as shown below.

If the first symbol of the string is a, step 2 is performed and if the first symbol of the string is b, step 3 is performed. After completion of step 2 or step 3, it is clear that the first symbol and the last symbol match and the machine is currently in state q_4. Now, we have to reset the read - write head to point to the first nonblank character in the string by repeatedly moving the head towards left and remain in state q_4. During this process, the symbols encountered may be a or b or B (blank character). Replace a by a, b by b and move the pointer towards left. The transitions defined for this can be of the form $\delta(q_4, a) = (q_4, a, L)$

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

But, if the symbol B is encountered , change the state to q_1, replace B by B and move the pointer towards right. the transition defined for this can be of the form

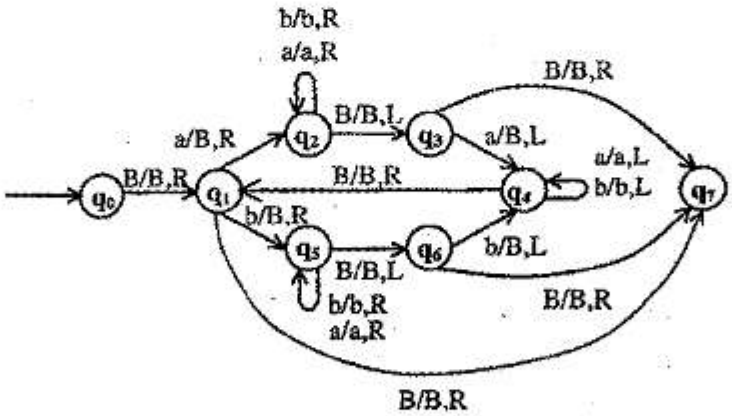
$$\delta(q_4, B) = (q_1, B, R)$$

After resetting the read - write head to the first non - blank character, repeat through step 1.

So, the TM to accept strings of palindromes over { a, b } is given by $M = (Q, \Sigma, \delta, q_0, B, F)$ where $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$; $\Sigma = \{a, b\}$; $\Gamma = \{a, b, B\}$; q_0 is the initial state B is the blank character ; $F = \{q_7\}$; δ is shown below using the transition table

	Γ		
δ	a	b	B
q_0	-	-	q_1, B, R
q_1	q_2, B, R	q_3, B, R	q_7, B, R
q_2	q_2, a, R	q_2, b, R	q_3, B, L
q_3	q_4, B, L	-	q_7, B, R
q_4	q_4, a, L	q_4, b, L	q_1, B, R
q_5	q_5, a, R	q_5, b, R	q_6, B, L
q_6	-	q_4, B, L	q_7, B, R
q_7	-	-	-

The transition diagram to accept palindromes over { a, b }is given by

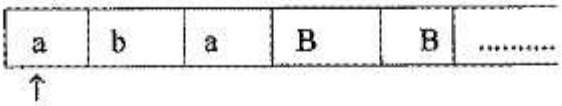


The reader can trace the moves made by the machine for the strings abba, aba and aaba and is left as an exercise.

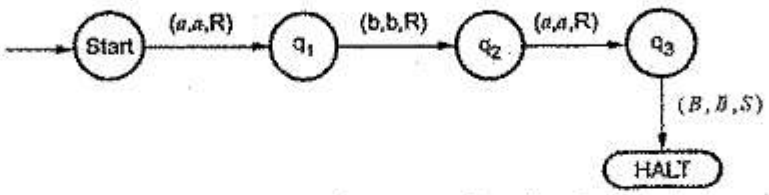
Example 7 : Construct a Turing machine which accepts the language of aba over $\Sigma = \{a,b\}$.

Solution : This TM is only for L = { aba }

We will assume that on the input tape the string 'aba' is placed like this

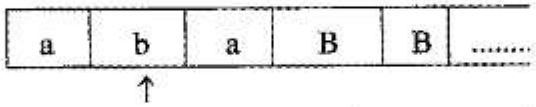


The tape head will read out the sequence upto the B character if 'aba' is readout the TM will halt after reading B.

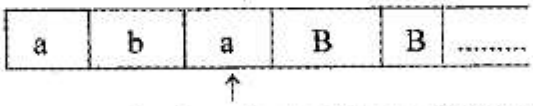


The triplet along the edge written is (input read, output to be printed, direction)

Let us take the transition between start state and q_1 is (a, a, R) that is the current symbol read from the tape is a then as a output a only has to be printed on the tape and then move the tape head to the right. The tape will look like this



Again the transition between q_1 and q_2 is (b, b, R). That means read b, print b and move right. Note that as tape head is moving ahead the states are getting changed.



The TM will accept the language when it reaches to halt state. Halt state is always a accept state for any TM. Hence the transition between q_3 and halt is (B, B, S). This means read B, print B and stay there or there is no move left or right. Eventhough we write (B, B, L) or (B, B, R) it is equally correct. Because after all the complete input is already recognized and now we simply want to enter into a accept state or final state. Note that for invalid inputs such as abb or ab or bab ..... there is either no path reaching to final state and for such inputs the TM gets stucked in between. This indicates that these all invalid inputs can not be recognized by our TM.

The same TM can be represented by another method of transition table

	a	b	B
Start	(q_1, a, R)	-	-
q_1	-	(q_2, b, R)	-
q_2	(q_1, a, R)	-	-
q_3	-	-	(HALT, B, S)
HALT	-	-	-

In the given transition table, we write the triplet in each row as :

(Next state, output to be printed, direction)

Thus TM can be represented by any of these methods.

Example 8 : Design a TM that recognizes the set $L = \{0^{2n}1^n \mid n \geq 0\}$.

Solution : Here the TM checks for each one whether two 0's are present in the left side. If it match then only it halts and accept the string.

The transition graph of the TM is ,

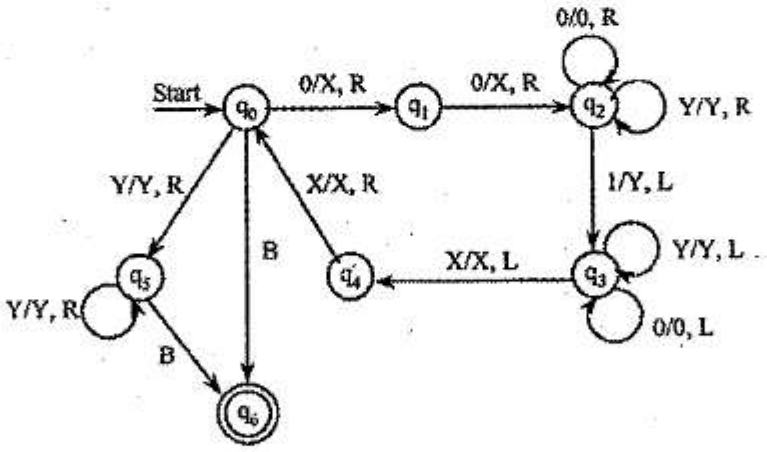


FIGURE : Turing Machine for the given language $L = \{0^{2n}1^n \mid n \geq 0\}$

Example 9 : Design Turing machine to recognize the palindromes of digits { 0, 1 }. Give its state transition diagram also.

Solution : The construction is made by defining moves in the following manner.

i. The machine scans the first input symbol (either 0 or 1), erases (but remembers) it, writes a blank symbol in place and changes state (q_1 or q_2).

ii. It scans the remaining part without changing the tape symbol until it encounters b. It then moves the read / write head a step left. If the rightmost symbol tallies with the leftmost symbol, the rightmost symbol is erased. Otherwise T. M. halts. The read/write head moves to the left until b is encountered.

iii. The above steps are repeated after changing the states suitably.

The transition table is shown below.

Present State	Tape Symbols 0	1	b
→ q_0	bRq_1	bRq_2	bRq_7
q_1	$0Rq_1$	$1Rq_1$	bLq_3
q_2	$0Rq_2$	$1Rq_2$	bLq_4
q_3	bLq_5	-	bRq_6
q_4	-	bLq_5	bRq_6
q_5	$0Lq_5$	$1Lq_5$	bRq_6
(q_6)	-	-	-

The transition diagram is shown in below figure.

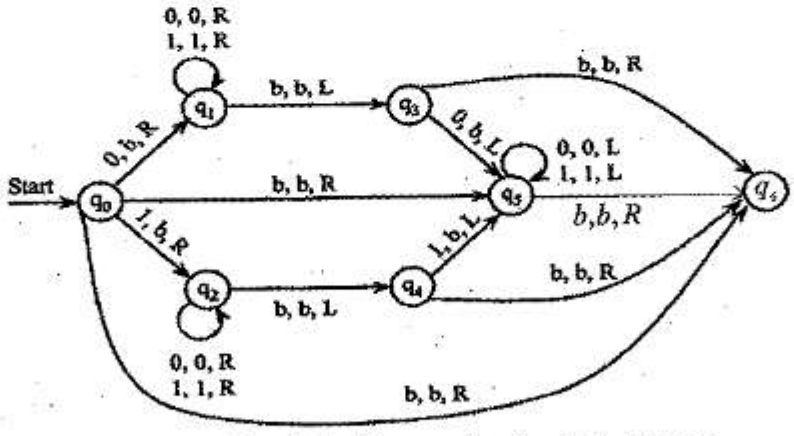


FIGURE : Transition State Diagram for the Palindromes

Example 10 : Design a Turing machine that accepts $L = \{a^n b^n | n \geq 0\}$.

Solution : The logic that we use for the Turing machine to be constructed is,

The Turing machine will remember leftmost a, by replacing it with B, then it moves the tape head right keeping the symbols it scans as it is, until it gets rightmost b, it remembers rightmost b, by replacing it with B, and moves the tape head left keeping the symbols it scans as it is till it reaches the B, on getting B, it moves the tape head one position right and repeats the above cycle if it gets a. If it gets B instead of a , then it is an indication of the fact the string is of the form $a^n b^n$, hence the Turing machine enters into the final state. Therefore, the moves of the Turing machine are given in below table .

	a	b	B
q_0	(q_1, B, R)		(q_4, B, R)
q_1	(q_1, a, R)	(q_1, b, R)	(q_2, B, L)
q_2		(q_3, B, L)	
q_3	(q_3, a, L)	(q_3, b, L)	(q_0, B, R)
q_4			

TABLE : Moves of the Turing Machine for the given language

Therefore, the Turing machine $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{a, b, B\}, \delta, q_0, B, \{q_4\})$, where is given above.

The transition diagram corresponding to the above Table is shown in below figure.

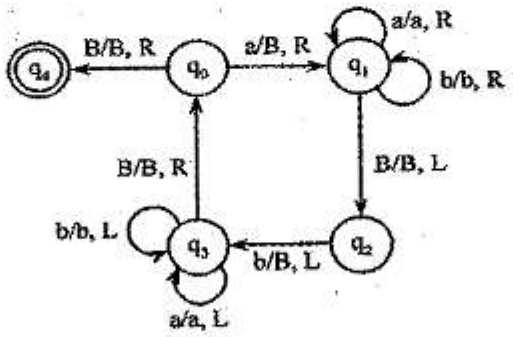


FIGURE : Transition Diagram for the above Table

Example 11 : What does the Turing Machine described by the 5 - tuples, $(q_0,0,q_0,1,R),(q_0,1,q_1,0,r),(q_0,B,q_2,B,R)$, $(q_1,0,q_1,0,R)$, $(q_1,1,q_0,1,R)$ *and* (q_1,B,q_2,B,R) .Do when given a bit string as input ?

Solution : The transition diagram of the TM is ,

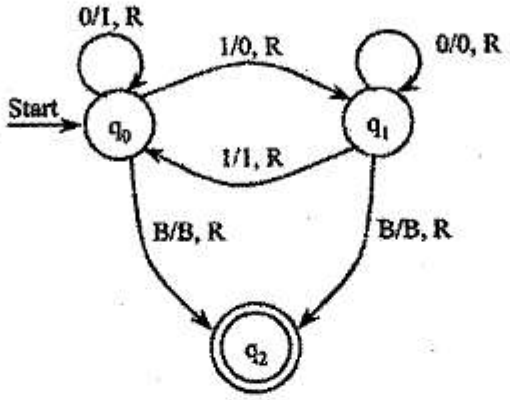


FIGURE : Transition Diagram for the given TM

The TM here reads an input and starts inverting 0's to 1's and 1's to 0's till the first 1. After it has inverted the first 1, it read the input symbol and keeps it as it is till the next 1. After encountering the 1 it starts repeating the cycle by inverting the symbol till next 1. It halts when it encounters a blank symbol.

7.4 COMPUTABLE FUNCTIONS

A Turing machine is a language acceptor which checks whether a string x is accepted by a language L. In addition to that it may be viewed as computer which performs computations of functions from integers to integers. In traditional approach an integer is represented in unary, an integer $i \geq 0$ is represented by the string 0^i .

Example 1 : 2 is represented as 0^2 . If a function has k arguments, $i_1, i_2, \ldots\ldots i_k$, then these integers are initially placed on the tape separated by 1's, as $0^{i_1}10^{i_2}1 \ldots\ldots 10^{i_k}$.

If the TM halts (whether in or not in an accepting state) with a tape consisting of 0's for some m, then we say that $f(i_1, i_2, \ldots\ldots i_k) = m$, where f is the function of k arguments computed by this Turing machine.

Example 2 :

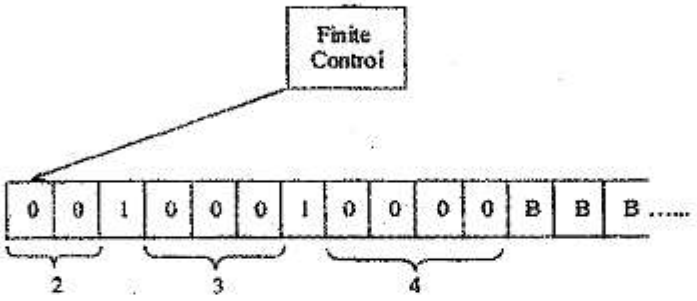
Consider a function in C.

```
int sum ( int x, int y, int z )
{       int s ;
        s = x + y + z ;
        return s;
}
```

Suppose this function is invoked using statement,

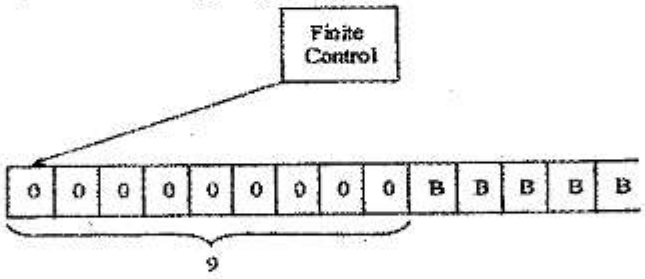
```
c = sum ( 2, 3, 4 ) ;
```

After invoking sum ( ), c will have the value 9. The same computation can be performed by Turing machine also. Initially, the Turing machine will have the arguments of sum( ) i.e., 2, 3, 4 on its tape as shown in figure (a).



(a) Before Computation

This Turing machine performs the sum of these arguments. After some moves it halts with the tape containing value 9, as shown in figure (b).



(b) After Computation

FIGURE : Elements on Tape to Compute Sum

Note that a Turing machine may compute a function of one argument, a function of two arguments and so on. The Turing machine given in figure can perform sum of two arguments or three arguments or in general sum of any finite number of arguments.

If TM M computes function f of k arguments i then f need not have a value for all different k - tuples of integers $i_1, i_2, \ldots\ldots i_k$ .if $f(i_1, i_2, \ldots\ldots i_k)$ is defined for all, $i_1 \ldots i_k$, then we say f is a total recursive function, otherwise we say f is partial recursive function. Total recursive functions are analogues to recursive language because they are computed by TM that always halts. Partial recursive function are analogues to recursively enumerable languages. Because they are computed by TM that may or may not halt. Examples of total recursive functions, all common arithmetic functions on integers, such as multiplication etc, are total recursive functions.

Example 3 : Construct Turing machine to find proper subtraction m - n is defined to be m - n for $m \geq n$ and zero for m < n .

Solution : The TM $M = (\{q_0, q_1, \ldots q_6\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \phi)$ defined below, started with $0^m 1 0^n$ on its tape, halts with 0^{m-n} on its tape. M repeatedly replaces its leading 0 by blank, then searches right for a 1 followed by a 0 and changes the 0 to 1. Next, M moves left until it encounters a blank and then repeats the cycle. The repetition ends if

i. Searching right for a 0, M encounters a blank. Then, the n 0's in $0^m 1 0^n$ have all changed to 1's and n + 1 of the m 0's have been changed to B. M replaces the n + 1 1's by a 0 and n B's leaving m - n 0's on its tape.

ii. Beginning the cycle, M cannot find a 0 to change to a blank, because the first m 0 is already have been changed. Then $n \geq m$. So m - n = 0. M replaces all remaining 1's and 0's by B.

The function δ is described below.

1. $\delta(q_0, 0) = (q, B, R)$
   Begin the cycle, Replace the leading 0 by B.
2. $\delta(q_1, 0) = (q_1, 0, R)$
   $\delta(q_1, 1) = (q_2, 1, R)$
   Search right, looking for the first 1.
3. $\delta(q_2, 1) = (q_2, 1, R)$
   $\delta(q_2, 0) = (q_3, 1, L)$
   Search right past 1's until encountering a 0, change that to 1.
4. $\delta(q_3, 0) = (q_3, 0, L)$
   $\delta(q_3, 1) = (q_3, 1, L)$
   $\delta(q_3, B) = (q_0, B, R)$
   Move left to a blank. Enter state q_0 to repeat the cycle.
5. $\delta(q_2, B) = (q_4, B, L)$

$$\delta(q_4,1) = (q_4, B, L)$$
$$\delta(q_4,0) = (q_4, 0, L)$$
$$\delta(q_4,0) = (q_6, 0, R)$$

If in state q_2 a B is encountered before a 0, we have situation (i) described above. Enter state q_4 and move left, changing all 1's to B 's until encountering a 'B'. This B is changed back to a 0, state q_6 is entered, and M halts.

6. $$\delta(q_0,1) = (q_5, B, R)$$
$$\delta(q_5,0) = (q_5, B, R)$$
$$\delta(q_5,1) = (q_5, B, R)$$
$$\delta(q_5,B) = (q_6, B, R)$$

If in state q_0 a 1 is encountered instead of a 0, the first block of 0's has been exhausted, as in situation (ii) above. M enters state q_5 to erase the rest of the tape, then enters q_6 and halts.

Example 4 : Design a TM which computes the addition of two positive integers.

Solution : Let TM $M = (Q, \{0, 1, \#\}, \delta, s)$ computes the addition of two positive integers m and n. It means, the computed function f(m, n) defined as follows :

$$f(m,n) = \begin{cases} m+n (If\ m, n \geq 1) \\ 0 \qquad (m = n = 0) \end{cases}$$

1 on the tape separates both the numbers m and n. Following values are possible for m and n.

1. $m = n = 0$ (# 1 # ..... is the input),
2. m = 0 and $n \neq 0$ (#10^n# ..... is the input),
3. $m \neq 0$ and n = 0 (#0^{m}1# ... is the input), and
4. $m \neq 0$ and $n \neq 0$ (#0^{m}10^n# ..... is the input)

Several techniques are possible for designing of M, some are as follows :

(a) M appends (writes) m after n and erases the m from the left end.

(b) M writes 0 in place of 1 and erases one zero from the right or left end . This is possible in case of $n \neq 0$ or $m \neq 0$ only. If m =0 or n = 0 then 1 is replaced by #.

We use techniques (b) given above. M is shown in below figure.

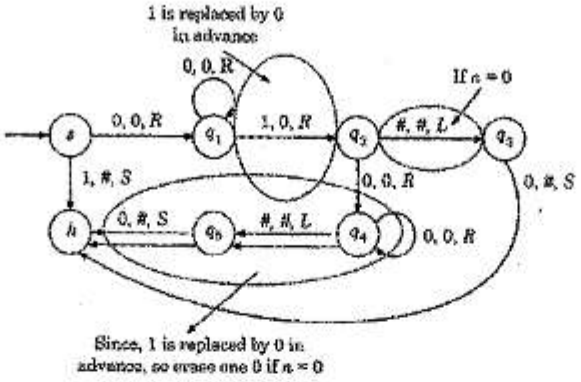


FIGURE : TM for addition of two positive integers

7.5 RECURSIVELY ENUMERABLE LANGUAGES

A language L over the alphabet Σ is called recursively enumerable if there is a TM M that accept every word in L and either rejects (crashes) or loops for every word in language L' the complement of L.

Accept (M) = L

Reject (M) + Loop (M) = L'

When TM M is still running on some input (of recursively enumerable languages) we can never tell whether M will eventually accept if we let it run for long time or M will run forever (in loop).

Example : Consider a language (a + b) * bb (a + b) *.

TM for this language is ,

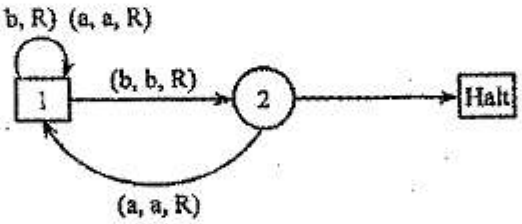


FIGURE : Turing Machine for (a + b) * bb (a + b) *

Here the inputs are of three types.

1. All words with bb = accepts (M) as soon as TM sees two consecutive b's it halts.
2. All strings without bb but ending in b = rejects (M). When TM sees a single b, it enters state2. If the string is ending with b, TM will halt at state 2 which is not accepting state. Hence it is rejected.
3. All strings without bb ending in 'a' or blank 'B' = loop (M) here when the TM sees last a it enters state 1. In this state on blank symbol it loops forever.

Recursive Language

A language L over the alphabet Σ is called recursive if there is a TM M that accepts every word in L and rejects every word in L' i. e.,

accept (M) = L
reject (M) = L'
loop (M) = ϕ.

Example :Consider a language b (a + b) * . It is represented by TM as :

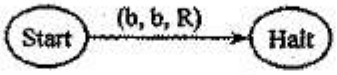


FIGURE : Turing Machine for b (a + b) *

This TM accepts all words beginning with 'b' because it enters halt state and it rejects all words beginning with a because it remains in start state which is not accepting state.

A language accepted by a TM is said to be recursively enumerable languages. The subclass of recursively enumberable sets (r. e) are those languages of this class are said to be recursive sets or recursive language.

7.6 CHURCH'S HYPOTHESIS

According to church's hypothesis, all the functions which can be defined by human beings can be computed by Turing machine. The Turing machine is believed to be ultimate computing machine.

The church's original statement was slightly different because he gave his thesis before machines were actually developed. He said that any machine that can do certain list of operations will be able to perform all algorithms. TM can perform what church asked, so they are possibly the machines which church described.

Church tied both recursive functions and computable functions together. Every partial recursive function is computable on TM. Computer models such as RAM also give rise to partial recursive functions. So they can be simulated on TM which confirms the validity of churches hypothesis.

Important of church's hypothesis is as follows .

1. First we will prove certain problems which cannot be solved using TM.

2. If churches thesis is true this implies that problems cannot be solved by any computer or any programming languages we might every develop.

3. Thus in studying the capabilities and limitations of Turing machines we are indeed studying the fundamental capabilities and limitations of any computational device we might even construct.

It provides a general principle for algorithmic computation and, while not provable, gives strong evidence that no more powerful models can be found.

7.7 COUNTER MACHINE

Counter machine has the same structure as the multistack machine, but in place of each stack is a counter. Counters hold any non negative integer, but we can only distinguish between zero and non zero counters.

Counter machines are off - line Turing machines whose storage tapes are semi - infinite, and whose tape alphabets contain only two symbols, Z and B (blank). Furthermore the symbol Z, which serves as a bottom of stack marker, appears initially on the cell scanned by the tape head and may never appear on any other cell. An integer i can be stored by moving the tape head i cells to the right of Z. A stored number can be incremented or decremented by moving the tape head right or left. We can test whether a number is zero by checking whether Z is scanned by the head, but we cannot directly test whether two numbers are equal.

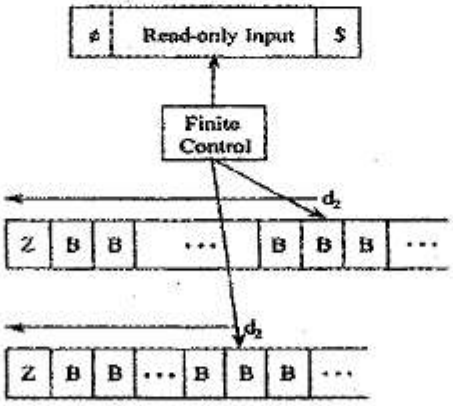


FIGURE : Counter Machine

¢ and $ are customarily used for end markers on the input. Here Z is the non blank symbol on each tape. An instantaneous description of a counter machine can be described by the state, the input tape contents, the position of the input head, and the distance of the storage heads from the symbol Z (shown here as d_1 and d_2). We call these distances the counts on the tapes. The counter machine can only store a count an each tape and tell if that count is zero.

Power of Counter Machines

- Every language accepted by a counter Machine is recursively enumerable.
- Every language accepted by a one - counter machine is a CFL so a one - counter machine is a special case of one - stack machine i. e., a PDA

7.8 TYPES OF TURING MACHINES

Various types of Turing Machines are :

i. With multiple tapes.
ii. With one tape but multiple heads.
iii. With two dimensional tapes.
iv. Non deterministic Turing machines.

It is observed that computationally all these Turing Machines are equally powerful. That means one type can compute the same that other can. However, the efficiency of computation may vary.

1. Turing machine with Two - Way Infinite Tape :

This is a TM that have one finite control and one tape which extends infinitely in both directions.

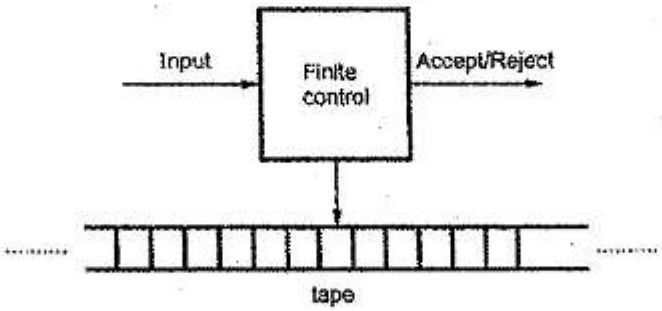


FIGURE : TM with infinite Tape

It turns out that this type of Turing machines are as powerful as one tape Turing machines whose tape has a left end.

2. Multiple Turing Machines :

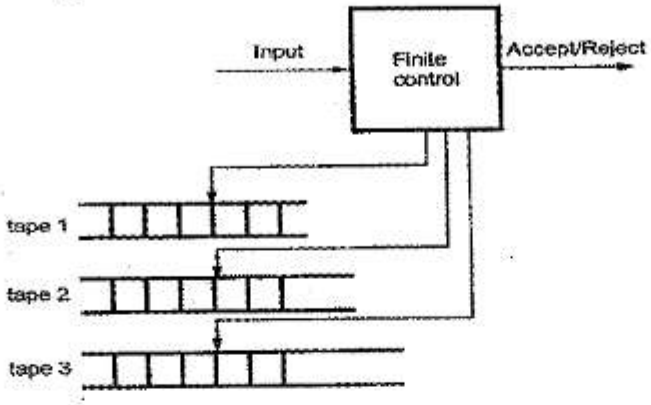


FIGURE : Multiple Turing Machines

A multiple Turing machine consists of a finite control with k tape heads and k tapes, each tape is infinite in both directions. On a single move depending on the state of the finite control and the symbol scanned by each of the tape heads, the machine can

1. Change state.
2. Print a new symbol on each of the cells scanned by its tape heads.
3. Move each of its tape heads, independently, one cell to the left or right or keep it stationary.

Initially, the input appears on the first tape and the other tapes are blank.

3. Nondeterministic Turing Machines :

A nondeterministic Turing machine is a device with a finite control and a single, one way infinite tape. For a given state and tape symbol scanned by the tape head, the machine has a finite number of choices for the next move. Each choice consists of a new state, a tape symbol to print, and a direction of head motion. Note that the non deterministic TM is not permitted to make a move in which the next state is selected from one choice, and the symbol printed and / or direction of head motion are selected from other choices. The non deterministic TM accepts its input if any sequence of choices of moves leads to an accepting state.

As with the finite automaton, the addition of nondeterminism to the Turing machine does not allow the device to accept new languages.

4. Multidimensional Turing Machines :

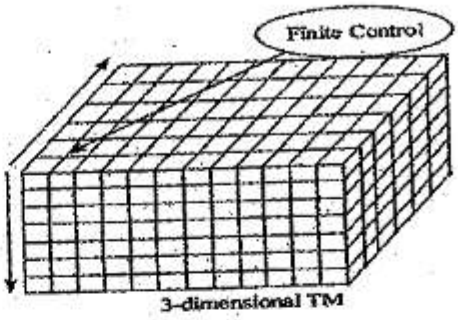


FIGURE : Multidimensional Turing Machine

The multidimensional Turing machine has the usual finite control, but the tape consists of a k - dimensional array of cells infinite in all 2k directions, for some fixed k. Depending on the state and symbol scanned, the device changes state, prints a new symbol, and moves its tape head in one of 2 k directions, either positively or negatively, along one of the k axes. Initially, the input is along one axis, and the head is at the left end of the input.At any time, only a finite number of rows in any dimension contains nonblank symbols, and these rows each have only a finite number of nonblank symbols

5. Multihead Turing Machines :

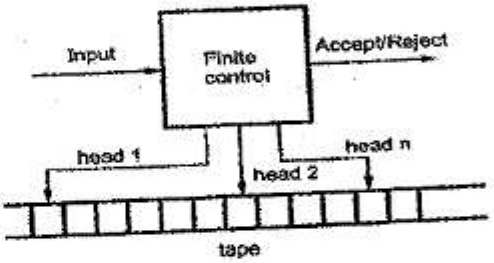


FIGURE : Multihead Turing Machine

A k - head Turing machine has some fixed number, k, of heads. The heads are numbered 1 through k, and a move of the TM depends on the state and on the symbol scanned by each head. In one move, the heads may each move independently left, right or remain stationary.

6. Off - Line Turing Machines :

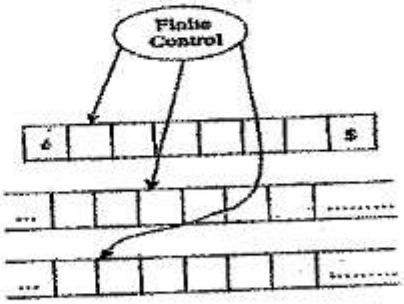


FIGURE : Off - line Turing Machine

An off - line Turing machine is a multitape TM whose input tape is read - only. Usually we surround the input by end markers, ¢ on the left and $ on the right. The Turing machine is not allowed to move the input tape head off the region between ¢ and $.

Off - line TM is just a special case of the multitape TM, and is no more powerful than any of the models we have considered. Conversely, an off - line TM can simulate any TM M by using one more tape than M. The first thing the off - line TM does is copy its own input onto the extra tape, and it then simulates M as if the extra tape were M's input.

7. Multistack Machines :

A deterministic two - stack machine is a deterministic Turing machine with a read only input and two storage tapes. If a head moves left on either tape, a blank is printed on that tape.

Multistack machine and counter machines are restricted Turing machines equivalent to the basic model.

7.9 COMPARISON OF FM, PDA AND TM

Basically have discussed three models viz. finite automata or finite machines (FM), Pushdown automata (PDA) and Turing machine (TM). We will now discuss the comparison between these models,

1. The finite machine is of two types - deterministic finite state machine and non deterministic finite state machine. Both of these DFA and NFA accept regular language only. Hence both the machines have equal power i. e. DFA = NFA.
2. We have then learn push down automata again, pushdown automata consists of two types of models deterministic PDA and Non deterministic PDA. The advantage of PDA over FA is that PDA has a memory and hence PDA accepts large class of languages than FA. Hence PDA has more power than FA. The non deterministic PDA accepts the language of context free grammar power of DPDA is less than NPDA as NPDA accepts a larger class of CFL.
3. The class of two stack or n - stack PDA has more power than one stack DPDA or NPDA. Hence two - stack / n - stack PDAS are more powerful.
4. Turing machines can be programmed. Hence TM accepts very very large class of languages. TM, therefore is the most powerful computational model.

TM > PDA > FM

TM accepts regular and non - regular languages ; context free and context sensitive languages as well.

REVIEW QUESTIONS

Q1. Explain Turing machine .

Answer :

For Answer refer to Topic : 7.2, Page No : 7.1.

Q2. Differentiate between TM and PDA.

Answer :

For Answer refer to Topic : 7.2.5, Page No : 7.6.

Q3. Obtain a Turing machine to accept the language $L = \{ 0^n 1^n \mid n \geq 1 \}$.

Answer :

For Answer refer to example - 1 , Page No : 7.6.

Q4. Obtain a Turing machine to accept the language $L(M) = \{ 0^n 1^n 2^n \mid n \geq 1 \}$

Answer :

For Answer refer to example - 2 , Page No : 7.11.

Q5. Obtain a TM to accept the language $L = \{ w \mid w \in (0+1)^* \}$ containing the substring 001.

Answer :

For Answer refer to example - 3 , Page No : 7.14.

Q6. Obtain a Turing machine to accept the language containing strings of 0's and 1's ending with 011.

Answer :

For Answer refer to example - 4 , Page No : 7.16.

Q7. Obtain a Turing machine to accept the language $L = \{ w \mid w \text{ is even and } \Sigma = \{ a, b \} \}$

Answer :

For Answer refer to example - 5 , Page No : 7.17.

Q8. Obtain a Turing machine to accept a palindrome consisting of a's and b's of any length.

Answer :

For Answer refer to example - 6 , Page No : 7.19.

Q9. Construct a Turing machine which accepts the language of aba over $\Sigma = \{a,b\}$.

Answer :

For Answer refer to example - 7 , Page No : 7.22.

Q10. Design a TM that recognizes the set $L = \{0^{2^n}1^n \mid n \geq 0\}$.

Answer :

For Answer refer to example - 8 , Page No : 7.23.

Q11. Design Turing machine to recognize the palindromes of digits { 0, 1 }. Give its state transition diagram also.

Answer :

For Answer refer to example - 9 , Page No : 7.24.

Q12. Design a Turing machine that accepts $L = \{a^n b^n \mid n \geq 0\}$.

Answer :

For Answer refer to example - 10 , Page No : 7.25.

Q13. What does the Turing Machine described by the 5 - tuples,

$(q_0,0,q_0,1,R),(q_0,1,q_1,0,r),(q_0,B,q_2,B,R)$,

$(q_1,0,q_1,0,R)$, $(q_1,1,q_0,1,R)$ *and* (q_1,B,q_2,B,R). Do when given a bit string as input ?

Answer :

For Answer refer to example - 11 , Page No : 7.26.

Q14. Write a short notes on computable functions.

Answer :

For Answer refer to Topic : 7.4, Page No : 7.26.

Q15. Construct Turing machine to find proper subtraction m - n is defined to be m - n for $m \geq n$ and zero for m < n .

Answer :

For Answer refer to example - 3 , Page No : 7.28.

Q16. Design a TM which computes the addition of two positive integers.

Answer :

For Answer refer to example - 4 , Page No : 7.29.

Q17. Write about recursively Enumerable Languages .

Answer :

For Answer refer to Topic : 7.5, Page No : 7.30.

Q18. Explain about church's Hypothesis.

Answer :

For Answer refer to Topic : 7.6, Page No : 7.31.

Q19. Explain about counter machine with a neat diagram.

Answer :

For Answer refer to Topic : 7.7, Page No : 7.32.

Q20. List and explain various types of Turing Machines.

Answer :

For Answer refer to Topic : 7.8, Page No : 7.33.

OBJECTIVE TYPE QUESTIONS

1. The no.of symbols necessary to simulate any TM with m symbols & n states is
   (a) $4mn+m$ (b) mn (c) $8mn+4m$ (d) $m+n$
2. Find the false statement.
   (a) Turing machine is simple mathematical model of general purpose computer.
   (b) Turing machine is not capable to performing any calculation which can be performed by computer
   (c) We construct Turing machine to accept a given language
   (d) We construct Turing machine to carry out some algorithm
3. Which of the following classes of Turing machine is not equivalent to the class of standard Turing machine?
   (a) Non-deterministic Turing machines
   (b) Turing machines with stay option
   (c) Turing machines with semi-infinite tapes
   (d) All of these
4. Choose the correct statements
   (a) Every recursive language is recursively enumerable
   (b) $L\{a^n b^n c^n\}$ is recursively enumerable
   (c) Recursive languages are closed under union
   (d) All
5. A TM is more powerful than Finite state machine because
   (a) it has the capability to remember arbitrary long input symbols
   (b) tape movement is confined to one direction
   (c) it has no finite state control
   (d) none
6. An Finite state machine can be considered to be a TM
   (a) a finite tape length, rewinding capability and bi-directional tape movement.
   (b) a finite tape length, without rewinding and bi-directional movement
   (c) a finite tape length, without rewinding capability and unidirectional tape movement
   (d) a finite tape length, with rewinding and unidirectional movement

7. Turing machines can move how in memory?
   (a) It cannot move. (b) forward and backward
   (c) backward (d) forward
8. Turing machines use what as their memory?
   (a) infinite tape (b) finite tape
   (c) RAM (d) ROM
9. Turing machines can do----------
   (a) less than a real computer can do
   (b) everything that a real computer can do
   (c) more than a real computer can do.
   (d) Nothing
10. Turing machines are similar to finite automaton but have ------
   (a) unlimited and read-write memory
   (b) finite and read-write memory
   (c) unlimited and read-only memory
   (d) finite and read-only memory.
11. Comparing TM and computers we find
   (a) They cannot be compared
   (b) Both are Equivalent
   (c) TM have more computational power
   (d) Computers have more computational power
12. The class of TMs is equivalent to the class of
   (a) Type 3 Grammars (b) Type 2 Grammars
   (c) Type 1 Grammars (d) Type 0 Grammars
13. The class of unrestricted languages corresponds to
   (a) FA (b) PDA (c) LBA (d) TM
14. Any TM with m symbols & n states can be simulated by another TM with just 2 symbols & less than
   (a) mn states (b) $8mn+4$ states (c) $4mn+8$ states (d) $8mn$ states

15. Which statement is false?
(d) Turing machine is simple mathematical model of general purpose computer.
(c) Turing machine is not capable of performing any calculation which can be performed by computer
(b) We construct Turing machine to accept a given language
(a) we construct Turing machine to carry out some algorithm

16. By giving Turing machine more complex power we can increase the power of the Turing Machine
(a) Absolutely False (b) May not be True
(c) May be True (d) Absolutely True

17. The definition of Turing machines is robust because.....
(a) Turing machine has nothing to do with robustness.
(b) certain changes (such as many tapes) result in machines of equivalent power.
(c) turing machines will not crash for any input string.
(d) functional testing of turing machines finds no errors.

18. A Turing machine computes by going from one configuration to another. We say that configuration C_1 yields configuration C_2 if the Turing Machine can legally move from C_1 to C_2 in ------
(a) an infinite number of steps
(b) a single step
(c) a finite number of steps
(d) none of the above

19. In a Turing machine for a state q and two strings u and v over the tape alphabet writing 'uqv', specifies that the current state is q------
(a) the tape contents are uv, and the current head location is the first symbol of v.
(b) the tape contents are uv, and the current head location is the first symbol of u.
(c) the tape contents are uqv, and the current head location is the first symbol of v.
(d) The tape contents are uqv, and the current head location is the first symbol of u.

20. Turing machines output accept if they enter an accept state. When do Turing machine output reject?
(a) Never
(b) When they enter a reject state
(c) When they never end
(d) When they are not in an accept state and halts

21. Consider the Turing Machine M described the transition table.

Present State	Tape Symbols				
	0	1	x	y	b
q_1	xR_{q2}		-		bR_{q5}
q_2	$0bR_{q2}$	yL_{q3}	-	xR_{q2}	
q_3	$0L_{q4}$		xR_{q5}	xR_{q3}	
q_4	$0L_{q4}$		xR_{q1}		
q_5				xR_{q5}	bR_{q6}
q_6					

q_6 is the final state.

Refer to the Turing Machine whose transition diagram is given above. What is the final ID when string 011 is processed?

(a) xyq_51 (b) xyq_6yx (c) $xxyybq_6$ (d) xyq_61

22. Consider the transition table of a Turing machine :

Present State	Tape symbols		
	b	o	1
q_1	$1L_{q2}$	$0R_{q1}$	
q_2	bR_{q3}	$0L_{q2}$	$1L_{q2}$
q_3	bR_{q4}	bR_{q5}	
q_4	$0R_{q5}$	$0R_{q4}$	$1R_{q4}$
q_5	$0L_{q2}$		

q_5 is the final state.

Computation sequence of string 00 leads to?

(a) Error (b) $bbbb_{q5}0000$ (c) $bbb_{q5}000$ (d) $bb_{q5}00$

23. The grammar generated by production rules $S \rightarrow aSBc \mid abc$, $cB \rightarrow Bc$, $aB \rightarrow aa$ is

(a) $a^nb^nc^n\ n \leq 0$ (b) $a^nb^nc^n, n > 0$

(c) $a^nb^nc^n, n \geq 0$ (d) $a^nb^nc^n, n > 1$

24. The grammar generated by production rules $S \to A \mid Sc, A \to ab \mid aAb$ is

(a) $a^n b^n c^j\ n \geq 0$ and $c > 0$ (b) $a^n b^n c^j, n > 0$ and $c > 0$

(c) $a^n b^n c^j\ n \geq 0$ and $c \geq 0$ (d) $a^n b^n c^j, n > 0$ and $c \geq 0$

25. Consider a new type of turing machine where the head can move left and move right but cannot stay put. This new type of turing machine is.....

(a) not comparable.

(b) More powerful than the original Turing mahcine

(c) equivalent in power to the original Turing machine

(d) less powerful than the original Turing mahcine

26. The statement "Standard TM accepts the same languages as are accepted by a stay TM" is

(a) Always false. (b) True for all languages

(c) True only if languages is regular (d) True only if languages is a CFL

27. Find the false statement

(a)Standard TM is equivalent to linear bounded automata

(b) Standard Turing machine(TM) is equivalent to multi tape TM

(c) Standard TM is equivalent to non deterministic TM

(d) None

28. Which of the following is true: Read Write head can move

(a) to the left of right endmarker in LBA

(b) to the right of right endmarker in LBA

(c) to the right of left endmarker in LBA

(d)to the left of left endmarker in LBA

29. LBA is

(a) restricted T.M. from both sides (b) unrestricted T.M.

(c) restricted T.M. from one side (d) none

30. Which automata is associated with Context Sensitive Language? (Give the best answer)

(a) Linear Bounded Automata (b) Pushdown Automata

(c) Finite Automata (d) Turing Machine

31. Refer to the Turing Machine whose transition diagram as given above in question 21, What is the final ID when string 0011 is processed?

(a) $xyq_5 1$ (b) $xyq_6 yx$ (c) $xxyybq_6$ (d) $xyq_6 1$

32. Which of the following is not a variant of the standard Turing Mahine
(a) Universal Turing Machine (b) Linear Bounded Automata
(c) Pushdown Automata (d) None of the above.

33. Let B be a Linear bounded automata. Then grammar corresponding to L(B) is
(a) Regular grammar (b) Unrestricted grammar
(c) Context free language (d) Context sensitive language

34. The Linear bounded automata is a variant of
(a) Finite Automata (b) Turing Machine
(c) Pushdown Automata (d) None of these

35. Non-Deterministic Turing Machines are more powerful than deterministic Turing Machine
(a) Absolutely False (b) May not be True
(c) May be True (d) Absolutely True

36. Many models of general purpose computation exist. Some are very similar to the original Turing machine, others can be very different than the original.
All of these models are equivalent in power if.....
(a) there is no model if everything is equivalent to everything else!
(b) they have unrestricted access to unlimited memory, and satisfy certain reasonable requirements like performing only a finite amount of work in a single step.
(c) satisfy certain reasonable requirements like performing only a finite amount of work in a single step.
(d) they have unrestricted access to unlimited memory.

ANSWER KEY

1.(b)	2.(c)	3.(d)	4.(b)	5.(a)	6.(a)	7.(a)	8.(d)	9.(d)	10.(b)
11.(d)	12.(d)	13.(d)	14.(a)	15.(d)	16.(a)	17.(c)	18.(a)	19.(d)	20.(b)
21.(b)	22.(b)	23.(b)	24.(b)	25.(a,c)	26.(b)	27.(a)	28.(a,c)	29.(a)	30.(a)
31.(b)	32.(c)	33.(d)	34.(b)	35.(a)	36.(b)				

Formal Languages And Automata Theory

UNIT 5

COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF AERONAUTICAL ENGINEERING

DUNDIGAL, HYDERABAD - 500 043

8

COMPUTABILITY THEORY

After going through this chapter, you should be able to understand :

- Chomsky hierarchy of Languages
- Linear Bounded Automata and CSLs
- LR (0) Grammar
- Decidability of problems
- UTM and PCP
- P and NP problems

8.1 CHOMSKY HIERARCHY OF LANGUAGES

Chomsky has classified all grammars in four categories (type 0 to type 3) based on the right hand side forms of the productions.

(a) Type 0

These types of grammars are also known as phrase structured grammars, and RHS of these are free from any restriction. All grammars are type 0 grammars.

Example : productions of types $AS \rightarrow aS$, $SB \rightarrow Sb, S \rightarrow \in$ are type 0 production.

(b) Type 1

We apply some restrictions on type 0 grammars and these restricted grammars are known as type 1 or **context - sensitive grammars** (CSGs). Suppose a type 0 production $\gamma\alpha\delta \rightarrow \gamma\beta\delta$ and the production $\alpha \rightarrow \beta$ is restricted such that $|\alpha| \leq |\beta|$ and $\beta \neq \in$. Then these type of productions is known as type 1 production. If all productions of a grammar are of type 1 production, then grammar is known as type 1 grammar. The language generated by a context - sensitive grammar is called context - sensitive language (CSL).

In CSG, there is left context or right context or both. For example, consider the production $\alpha A\beta \rightarrow \alpha a\beta$. In this, α is left context and β is right context of A and A is the variable which is replaced.

The production of type $S \rightarrow \in$ is allowed in type 1 if $\in$ is in L(G), but S should not appear on right hand side of any production.

Example : productions $S \rightarrow AB, S \rightarrow \in, A \rightarrow c$ are type 1 productions, but the production of type $A \rightarrow Sc$ is not allowed . Almost every language can be thought as CSL.

Note : If left or right context is missing then we assume that $\in$ is the context.

(c) Type 2

We apply some more restrictions on RHS of type 1 productions and these productions are known as type 2 or context - free productions. A production of the form $\alpha \rightarrow \beta$, where $\alpha, \beta \in (V \cup \Sigma)^*$ is known as type 2 production. A grammar whose productions are type 2 production is known as type 2 or context - free grammar (CFG) and the languages generated by this type of grammars is called context - free languages (CFL).

Example : $S \rightarrow S + S, S \rightarrow S * S, S \rightarrow id$ are type 2 productions.

(d) Type 3

This is the most restricted type. Productions of types $A \rightarrow a$ or $A \rightarrow aB|Ba$, where $A, B \in V$, and $a \in \Sigma$ are known as type 3 or regular grammar productions. A production of type $S \rightarrow \in$ is also allowed, if $\in$ is in generated language.

Example : productions $S \rightarrow aS$, $S \rightarrow a$ are type 3 productions.

Left - linear production : A production of type $A \rightarrow Ba$ is called left - linear production.

Right - linear production : A production of type $A \rightarrow aB$ is called right - linear production. A left - linear or right - linear grammar is called regular grammar. The language generated by a regular grammar is known as regular language.

A production of type $A \to w$ or $A \to wB$ or $A \to Bw$, where $w \in \Sigma^*$ can be converted into the forms $A \to a$ or $A \to aB$ or $A \to Ba$, where $A, B \in V$ and $a \in \Sigma$.

Example : $A \to 10\,A$ can be replaced by productions $A \to 1B$, where B is a new variable and $B \to 0A$.

In general, if $A \to a_1a_2a_3 \ldots\ldots a_na_{n+1}B$, then this production can replaced by the following productions.

$$A \to a_1\ B_1,$$
$$B_1 \to a_2\ B_2,$$
$$B_2 \to a_3\ B_3,$$
$$\ldots$$
$$B_n \to a_{n+1}\ B$$

Similar result is obtained for left - linear grammars also.

8.1.1 Hierarchy of grammars

Type 0 or Phrase structured grammar

⇓ Restrictions applied

Type 1 or Context - sensitive grammar

⇓ Restrictions applied

Type 2 or Context - free grammar

⇓ Restrictions applied

Type 3 or Regular grammar

Example : Consider the following and find the type of the grammar.

(a) $S \to Aa,\ A \to c \mid Ba,\ B \to abc$

(b) $S \to aSa \mid c$

(c) $S \to aAS \mid SBb,\ AS \to aAS \mid aS,\ SB \to Sb \mid SBb$

Solution :

(a)

Production			Type
S	→	Aa	Type 3
A	→	c	Type 3
A	→	Ba	Type 3
B	→	abc	Type 3

So, given productions are of type 3 and hence grammar is regular.

(b)

S	→	aSa	Type 2
S	→	c	Type 3

So, given productions are of type 2 and hence grammar is CFG.

Note : We select the higher type and higher type between type 3 and type 2 is type 2).

(c)

S	→	aAS	Type 2
S	→	SBb	Type 2
AS	→	aAS	Type 1
AS	→	aS	Type 1
SB	→	Sb	Type 1
SB	→	SBb	Type 1

So, given productions are of type 1 and hence grammar is CSG.

8.1.2 Relation Among Grammars and Languages

Type 0 is the super set and type 1 is contained in type 0, type 2 is contained in type 1, and type 3 is contained in type 2.

$$\text{Type } 0 \subseteq \text{Type } 1 \subseteq \text{Type } 2 \subseteq \text{Type } 3$$

8.1.3 Languages and Their Related Automaton

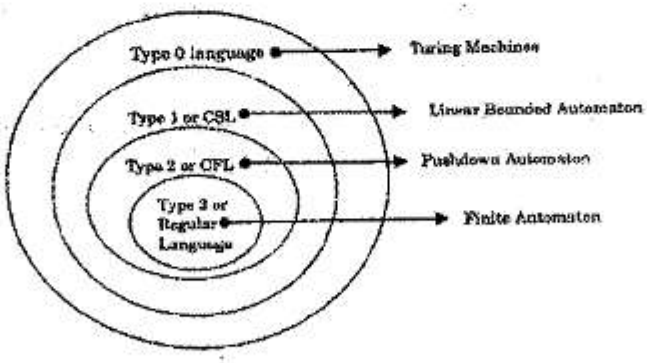


FIGURE : Languages and their related Automaton

8.2 LINEAR BOUNDED AUTOMATA

The Linear Bounded Automata (LBA) is a model which was originally developed as a model for actual computers rather than model for computational process. A linear bounded automaton is a restricted form of a non deterministic Turing machine.

A linear bounded automaton is a multitrack Turing machine which has only one tape and this tape is exactly of same length as that of input.

The linear bounded automaton (LBA) accepts the string in the similar manner as that of Turing machine does. For LBA halting means accepting. In LBA computation is restricted to an area bounded by length of the input. This is very much similar to programming environment where size of variable is bounded by its data type.

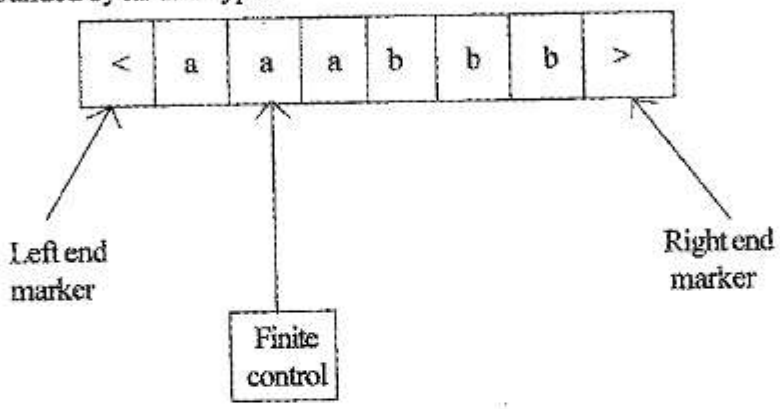


FIGURE : Linear bounded automaton

The LBA is powerful than NPDA but less powerful than Turing machine. The input is placed on the input tape with beginning and end markers. In the above figure the input is bounded by < and >.

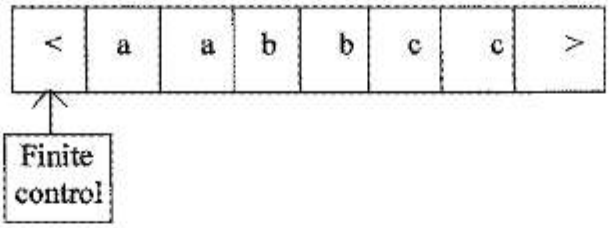
A linear bounded automata can be formally defined as :

LBA is 7 - tuple on deterministic Turing machine with

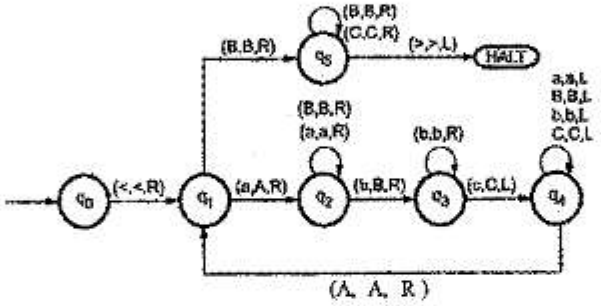
$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}) \text{ having}$$

1. Two extra symbols of left end marker and right end marker which are not elements of Γ.
2. The input lies between these end markers.
3. The TM cannot replace < or > with anything else nor move the tape head left of < or right of >.

Example : We can construct a language $L = \{a^n\ b^n\ c^n \mid n \geq 1\}$ using LBA as follows.



The input is placed on the input tape which is enclosed within left end marker and right end marker. We will apply the simple logic as : when we read 'a' convert it to A then move right by skipping all a's. On encountering first 'b' we will convert it to B. Then move right by skipping all b's. On receiving first c convert it to C. Move in left direction unless you get A. Repeat the above procedure and convert equal number of a's, b's, and c's to corresponding A's, B's and C's. Finally move completely to the rightmost symbol if it is '>' a right end marker,then HALT. The machine will be :



Simulation : Consider input aabbcc

< aabbcc >	Move right.
↑	
< aabbcc >	Convert to A, move right.
↑	
< Aabbcc >	Move right.
↑	
< Aabbcc >	Convert to B, move right.
↑	
< AaBbcc >	Move right.
↑	

<AaBbcc>	Convert to C, move left.
↑	
<AaBbCc>	Move left
↑	
<AaBbCc>	Move left.
↑	
<AaBbCc>	Move left.
↑	
<AaBbCc>	Move right.
↑	
<AaBbCc>	Convert to A, Move right.
↑	
<AABbCc>	Move right.
↑	
<AABbCc>	Convert to B, Move right.
↑	
<AABBCc>	Move right.
↑	
<AABBCc>	Convert to C, Move left.
↑	
<AABBCC>	Move left continuously by skipping B's.
↑	
<AABBCC>	Move right.
↑	
<AABBCC>	If we get B, we will move right to check whether all b's and c's are converted to B and C.
↑	
<AABBCC>	If we get right end marker '>' then we HALT by accepting the input aabbcc.
↑	

Thus in LBA the length of tape exactly equal to the input string and tape head can not move left of '<' right of '>'.

8.3 CONTEXT SENSITIVE LANGUAGES (CSLs)

The context sensitive languages are the languages which are accepted by linear bounded automata. These type of languages are defined by context sensitive grammar. In this grammar more than one terminal or non terminal symbol may appear on the left hand side of the production rule. Along with it, the context sensitive grammar follows following rules :

i. The number of symbols on the left hand side must not exceed number of symbols on the right hand side.
ii. The rule of the form $A \rightarrow \in$ is not allowed unless A is a start symbol. It does not occur on the right hand side of any rule.

The classic example of context sensitive language is $L = \{a^n\ b^n\ c^n \mid n \geq 1\}$. The context sensitive grammar can be written as :

S	→	aBC
S	→	SABC
CA	→	AC
BA	→	AB
CB	→	BC
aA	→	aa
aB	→	ab
bB	→	bb
bC	→	bc
cC	→	cc

Now to derive the string aabbcc we will start from start symbol :

S	rule S →	SABC
SABC	rule S →	aBC
aBCABC	rule CA →	AC
aBACBC	rule CB →	BC
aBABCC	rule BA →	AB
aABBCC	rule aA →	aa
aaBBCC	rule aB →	ab
aabBCC	rule bB →	bb
aabbCC	rule bC →	bc
aabbcC	rule cC →	cc
aabbcc		

Note : The language $a^n\ b^n\ c^n$ where $n \geq 1$ is represented by context sensitive grammar but it can not be represented by context free grammar.

Every context sensitive language can be represented by LBA.

8.4 LR (k) GRAMMARS

Before going to the topic of LR (k) grammar, let us discuss about some concepts which will be helpful understanding it.

In the unit of context free grammars you have seen that to check whether a particular string is accepted by a particular grammar or not we try to derive that sentence using rightmost derivation or leftmost derivation. If that string is derived we say that it is a valid string.

Example :

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow id \mid (E)$$

Suppose we want to check validity of a string id + id * id . Its rightmost derivation is

$$\begin{aligned} E &\Rightarrow E + T \\ &\Rightarrow E + T * F \\ &\Rightarrow E + T * id \\ &\Rightarrow E + F * id \\ &\Rightarrow E + id * id \\ &\Rightarrow T + id * id \\ &\Rightarrow F + id * id \\ &\Rightarrow id + id * id \end{aligned}$$

FIGURE(a) : Rightmost Derivation of id + id * id

Since this sentence is derivable using the given grammar. It is a valid string. Here we have checked the validity of string using process known as derivation.

The validity of a sentence can be checked using reverse process known as reduction. In this method for a given x, inorder to know whether it is valid sentence of a grammar or not, we start with x and replace a substring x_1 with variable A if $A \rightarrow X_1$ is a production. We repeat this process until we get starting state.

Consider the grammar,

$$E \rightarrow E + T \mid T$$
$$E \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid id$$

Let us check the validity of string id + id * id.

F + id * id Replaced F with id since F → id is a production
T + id * id Replaced F with T using production T → F
E + id * id Replaced T with E using production E → T
E + F * id Replaced id with F using production F → id
E + T * id Replaced F with using production T → F
E + T * F Replaced id with F using production F → id
E + T Replaced T * F with T using production T → T * F
E Replaced E + T with E using production E → E + T

FIGURE(b) : Reduction of id + id * id

Here since we are able to reduce to starting state E, so that id + id * id is accepted by the given grammar.

Note : There may be different ways of selecting as substring in sentential form. In our reduction we have used reverse of rightmost derivation shown in Figure(a).

The substring in right sentential form which causes reduction to starting state is known as handle and corresponding production is known as handle production. For example, in right sentential form E + T * id of Figure(b) we can either replace substring T with F using T → F or replace id with F using F → id. If we use the first reduction, the sentential form will become E + F * id. This will not lead to starting state. Hence here F is not handle. Where as if we reduce, the sentential form will be E + T * F which can be reduced to starting state using subsequent reductions. Hence here F is a handle and F → id is handle production.

In reduction process we have seen that we repeat the process of substitution until we get starting state. But some times several choices may be available for replacement. In this case we have to backtrack and try some other substring . For certain grammars it is possible to carry out the process in deterministic. (i. e., having only one choice at each time). LR grammars form one such subclass of context free grammars. Depending on the number of look ahead symbolized to determine whether a substring must be replaced by a non terminal or not, they are classified as LR(0) , LR(1).... and in general LR(k) grammars.

LR(k) stands for left to right scanning of input string using rightmost derivation in reverse order (we say reverse order because we use reduction which is reverse of derivation) using look ahead of k symbols.

8.4.1 LR(0) Grammar

LR(0) stands for left to right scanning of input string using rightmost derivation in reverse order using 0 look ahead symbols.

Before defining LR(0) grammars, let us know about few terms.

Prefix Property : A language L is said to have prefix property if whenever w in L, no proper prefix of w is in L. By introducing marker symbol we can convert any DCFL to DCFL with prefix property. Hence $L\$ = \{ w\$ \mid w \in L \}$ is a DCFL with prefix property whenever w is in L.

Example : Consider a language L = { cat, cart, bat, art, car } . Here, we can see that sentence cart is in L and its one of the prefixes car is also is in L. Hence, it is not satisfying property. But L$ = { cat $, cart $, bat $, art $, car $ }

Here, cart $ is in L$ but its prefix cart or car are not present in L$. Similarly no proper prefix is present in L$. Hence, it is satisfying prefix property.

Note : LR(0) grammar generates DCFL and every DCFL with prefix property has a LR(0) grammar.

LR Items

An item for a CFG is a production with dot any where in right side including beginning or end. In case of $\in$ production, suppose $A \to \in$, $A \to .$ is an item.

Example :

Consider the grammar,

$S' \rightarrow S$

$S \rightarrow cAd$

$A \rightarrow a \mid \in$

The items for this grammar are ,

$S' \rightarrow .S$

$S' \rightarrow S.$

$S \rightarrow .cAd$

$S \rightarrow c.Ad$

$S \rightarrow cA.d$

$S \rightarrow cAd.$

$A \rightarrow .a$

$A \rightarrow a.$

$A \rightarrow .$

An item indicates how much of a production we have seen at a given point in parsing process.

Valid Item : We say in item $A \rightarrow \alpha . \beta$ is valid for a viable prefix (i. e., most possible prefix) γ there is a rightmost derivation $S \underset{rm}{\overset{*}{\Rightarrow}} \delta A w \underset{rm}{\Rightarrow} \delta \alpha \beta w$ and $\delta\alpha = \gamma$.

Example :

$S \rightarrow cAt$

$A \rightarrow ar$

The sentence cart belongs to this grammar,

$S \overset{*}{\Rightarrow} CAt \Rightarrow cart$

The possible or viable prefixes for cart are { c, ca, car, cart } for the prefix ca $A \rightarrow a.r$. is valid item and for viable prefix car $A \Rightarrow ar$ is valid item.

Computing Valid Item Sets

The main idea here is to construct from a given grammar a deterministic finite automata to recognize viable prefixes. We group items together into sets which give to states of DFA. The items may be viewed as states of NFA and grouped items may be viewed as states of DFA obtained using subset construction algorithm.

To compute valid set of items we use two operations goto and closure.

Closure Operation

It I is a set of items for a grammar G, then closure (I) is the set of items constructed from I by two rules.

1. Initially, every item I is added to closure (I).
2. If $A \to \alpha.B\beta$ is in closure (I) and $B \to \delta$ is production then add item $B \to \delta$ to I, if it is not already there. We apply this rule until no more new items can be added to closure (I).

Example : For the grammar,

$$S' \to S$$
$$S \to cAd$$
$$A \to a$$

If $S' \to S$ is set of one item in state I then closure of I is,

$$I_1 : S' \to .s$$
$$S \to .cAD$$

The first item is added using rule 1 and $S \to .cAd$ is added using rule 2. Because '.' is followed by nonterminal S we add items having S in LHS. In $S \to .cAd$ '.' is followed by terminal so no new item is added.

Goto Function : It is written as goto (I, X) where I is set of items and X is grammar symbol.

If $A \to \alpha.X\beta$ is in some item set I then goto (I, X) will be closure of set of all item $A \to \alpha.X.\beta$.

For example,

goto (I_1, c)

closure $(S \rightarrow c.Ad)$

i. e., $S \rightarrow c.Ad$

$A \rightarrow a$

now let us see how all the valid sets of items are computed for the given grammar in example 1.

Initially I_0 will be the starting state. It contains only the item $S' \rightarrow .S$ we find its closure to find set of items in this state for each state I_1 and symbol β after '.' we apply goto (I_i, β), goto (I_0, S) and find its closure. This constitutes next state I_j. We continue this process goto (I_0, a) until no new states are obtained.

$I_0 : S' \rightarrow .S$

$S \rightarrow .Ad$

$I_1 : S' \rightarrow S.$

$I_2 : S \rightarrow c.Ad$

$A \rightarrow .a$

goto (I_2, A)

$I_3 : S \rightarrow cA.d$

goto (I_2, a)

$I_4 : A \rightarrow a.$

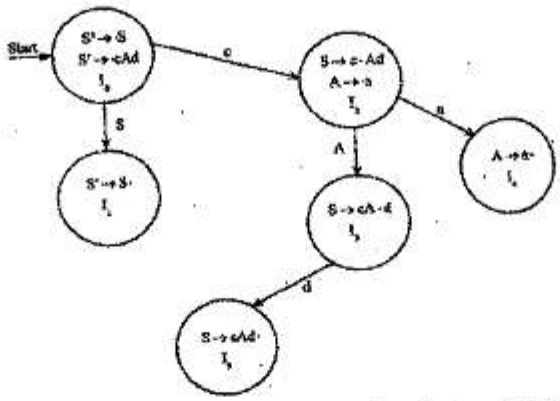
goto (I_3, d)

$I_5 : S \rightarrow cAd.$

This process is stopped because all possible complete items are obtained. A complete item is the one which has dot in rightmost position.

Ecah item set corresponds to a state of DFA. Hence, the DFA for given grammar will have six states corresponding to I_0 *to* I_5.

DFA :



FIGURE(a) : DFA whose States are the Sets of Valid Items

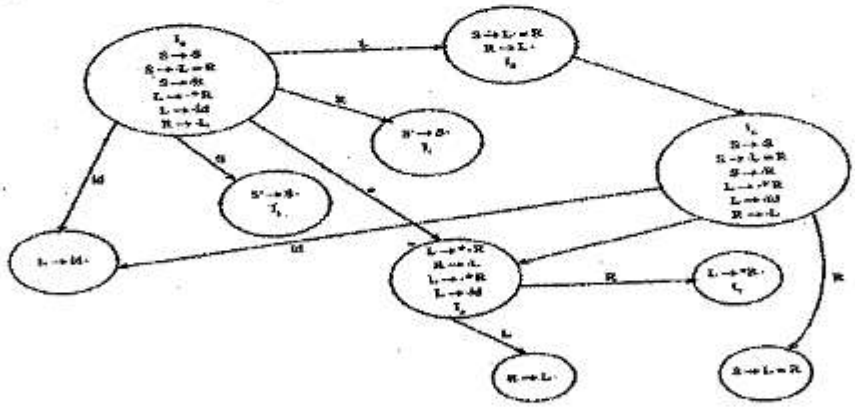
Definition of LR(0) Grammar : We say G is an LR (0) grammar if,

1. Its start symbol does not appear on the right hand side of any production and
2. For every viable prefix γ of G, whenever $A \to \alpha$ is a complete item valid for γ, then no other complete item nor any item with terminal to the right of the dot is valid for γ.

Condition 1 : For a grammar to be LR(0) it should satisfy both the conditions. The first condition can be made to satisfy by all grammars by introduction of a new production $S' \to S$ is known augmented grammar.

Condition 2 : For the DFA shown in Figure(a), the second condition is also satisfied because in the item sets I_1, I_4 and I_5 each containing a complete item, there are no other complete items nor any other conflict.

Example : Consider the DFA given in figure(b).



FIGURE(b) : DFA for the given Grammar

DFA for grammar,

$S \rightarrow L = R$

$S \rightarrow R$

$L \rightarrow *R$

$L \rightarrow id$

$R \rightarrow L$

i. The first condition of LR(0) grammar is satisfied.

ii. Consider state I_2 and viable prefixes of L = R { L, L = and L = R } for prefix LR → L. is a complete item and there is another item having the prefix L i. e., S → L . = R followed by terminal. Hence, violating second rule. So it is not LR(0) grammar.

8.5 DECIDABILITY OF PROBLEMS

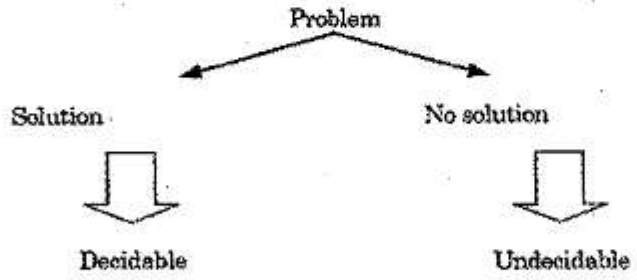
In our general life, we have several problems and some of these have solution also, but some have not. Simply, we say a problem is decidable if there is a solution otherwise undecidable.

Example : consider following problems and their possible answers.

1. Does the sun rise in the east ? (YES)
2. Does the earth move around the sun ? (YES)
3. What is your name ? (FLAT)
4. Will tomorrow be a rainy day ? (No answer)

We have solutions (answers) for all problems except the last. We can not answer the last problem, because we have no way to tell about the weather of tomorrow, but to some extent we can only predict. So, the last problem is undecidable and remaining problems are decidable.

So, if a problem can be solved or answered based on some algorithm then it is decidable otherwise undecidable.



Each problem P is a pair consisting of a set and a question, where the question can be applied to each element in the set. The set is called the domain of the problem, and its elements are called the instances of the problem.

Example :

Domain = { All regular languages over some alphabet Σ } ,
Instance : L = { w : w is a word over Σ ending in abb} ,
Question : Is union of two regular languages regular ?

8.5.1 Decidable and Undecidable Problems

A problem is said to be decidable if
1. Its language is recursive, or
2. It has solution

Other problems which do not satisfy the above are undecidable. We restrict the answer of decidable problems to " YES" or "NO" . If there is some algorithm exists for the problem, then outcome of the algorithm is either "YES" or "NO" but not both. Restricting the answers to only "YES" or "NO" we may not be able to cover the whole problems, still we can cover a lot of problems. One question here. Why we are restricting our answers to only "YES" or "NO"? The answer is very simple ; we want the answers as simple as possible.

Now, we say " If for a problem, there exists an algorithm which tells that the answer is either "YES" or "NO" then problem is decidable."

If for a problem both the answers are possible ; some times "YES" and sometimes "NO", then problem is undecidable.

8.5.2 Decidable Problems for FA, Regular Grammars and Regular Languages

Some decidable problems are mentioned below :
1. Does FA accept regular language ?
2. Is the power of NFA and DFA same ?
3. L_1 and L_2 are two regular languages. Are these closed under following :
   (a) Union
   (b) Concatenation
   (c) Intersection
   (d) Complement

(e) Transpose
(f) Kleene Closure (positive transitive closure)

4. For a given FA M and string w over alphabet Σ, is $w \in L(M)$? This is decidable problem.
5. For a given FM, is $L(M) = \phi$? This is a decidable problem.
6. For a given FA M and alphabet Σ, is $L(M) = \Sigma^*$? This is a decidable problem.
7. For given two FA M_1 and M_2, $L(M_1), L(M_2) \in \Sigma^*$, is $L(M_1) = L(M_2)$? This is a decidable problem.
8. For given two regular languages L_1 and L_2 over some alphabet Σ, is $L_1 \subset L_2$? This is a decidable problem.

8.5.3 Decidable And Undecidable Problems About CFLs, And CFGs

Decidable Problems

Some decidable problems about CFLs and CFGs are given below.

1. If L_1 and L_2 are two CFLs over some alphabet Σ, then $L_1 \cup L_2$ is CFL.
2. If L_1 and L_2 are two CFLs over some alphabet Σ, then $L_1 L_2$ is CFL.
3. If L is a CFL over some alphabet Σ, then L* is a CFL.
4. If L_1 is a regular language, L_2 is a CFL then $L_1 \cup L_2$ is CFL.
5. If L_1 is a regular language, L_2 is a CFL over some alphabet Σ, then $L_1 \cap L_2$ is CFL.
6. For a given CFG G, is $L(G) = \phi$ or not ?
7. For a given CFG G, finding whether L(G) is finite or not, is decidable.
8. For a given CFG G and a string w over Σ, checking whether $w \in L(G)$ or not is decidable.

Undecidable Problems

Following are some undecidable problems about CFGs and CFLs :

1. For two given CFLs L_1 and L_2, whether $L_1 \cap L_2$ is CFL or not, is undecidable.
2. For a given CFL L over some alphabet Σ, whether complement of L i. e. E * - L is CFL or not, is undecidable.
3. For a given CFG G, is L(G) ambiguous ? This is undecidable problem.
4. For two arbitrary CFGs G_1 and G_2, deciding $L(G_1) \cap L(G_2) = \phi$ or not, is undecidable.
5. For two arbitrary CFGs G_1 and G_2, deciding $L(G_1) \subseteq L(G_2)$ or not, is undecidable.

8.5.4 Decidability and Undecidability About TM

We have considered TM as a most powerful machine that can compute anything, which can recognize any language. So, from where undecidability comes and why ? These questions are really interesting. According to Church - Turing Thesis, we have considered TM as an algorithm and an algorithm as a TM . So, for a problem, if there is an algorithm (solution to find answer) then problem is decidable and TM can solve that problem. We have several problems related to computation and recognization that have no solution and these problems are undecidable.

Partial Decidable and Decidable Problems

A TM M is said to partially solve a given problem P if it provides the answer for each instance of the problem and the problem is said to be partially solvable. If all the computations of the TM are halting computations for P, then the problem P is said to solvable.

A TM is said to partially decide a problem if the following two conditions are satisfied.

(a) The problem is a decision problem, and

(b) The TM accepts a given input if and only if the problem has an answer "YES" for the input, that is the TM accepts the language L= { x : x is an instance of the problem, and the problem has the answer "YES" for x }.

A TM is said to decide a problem if it partially decides the problem and all its computations are halting computations.

The main difference between a TM M_1 that partially solves (partially decides) a problem and a TM M_2 that solves (decides) the same problem is that M_1 might reject an input by a non - halting computation, whereas M_2 can reject the input only by a halting computation.

A problem is said to be unsolvable if no algorithm can solve it, and a problem is said to be undecidable if it is a decision problem and no algorithm can decide it.

Decidable Problems about Recursive and Recursive Enumerable Languages

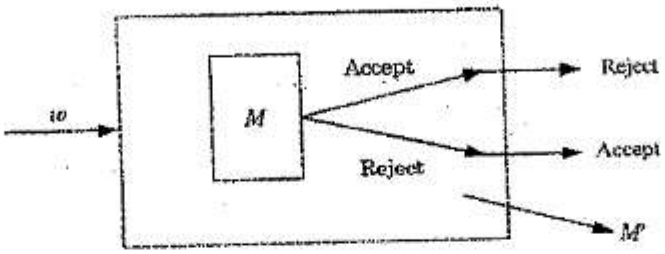
As we have discussed earlier that if a problem has a solution then it is decidable. In this section, we will discuss some decidable problems about recursive and recursive enumerable languages.

1. The complement of a recursive language L over some alphabet Σ is recursive.

Proof : We will discuss a constructive algorithm to prove that complement of a recursive language is also recursive i. e. recursive languages are closed under complementation.

As we know that for all strings $w \in L$, a TM always halts and rejects those strings that are not in L. So, " for all strings $w \in L$ " is always decidable.

We construct a TM M, which recognizes the language L. We construct another TM M' based on M such that M' accepts those strings which are rejected by M. It means, if M accepts then M' does not. M' rejects those strings that are accepted by M. It means, all strings $x \notin L$ are accepted by M' and for all strings $w \in L$ are rejected. So, M' also follows same kind of algorithm to decide whether a string $w \in L$ or not. Hence, complement of recursive language L i.e. $\Sigma^* - L$ is also recursive. The logic diagram of M' is shown in Figure(a).



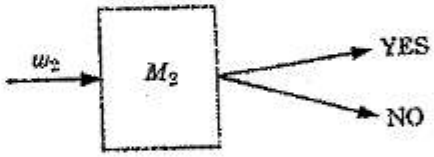
Figure(a)

In general, recursive languages are closed under complement operation.

2. The union of two recursive languages is recursive.

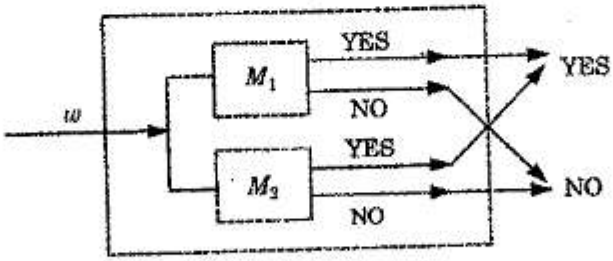
Proof : Let L_1 and L_2 be two recursive languages and Turing machines M_1 and M_2 recognize L_1 and L_2 respectively shown in Figure(b) and Figure(c).

Figure(b)



Figure(c)

We construct a third TM M_3, which follows either M_1 or M_2 as shown in figure(d).



Figure(d)

TM M_3 accepts if either M_1 accepts or M_2 accepts and rejects if either M_1 rejects or M_2 rejects. Since, M_1 and M_2 are based on algorithms, so M_3 is also based on the same kind of algorithm. Therefore, union of two recursive languages L_1 and L_2 is also recursive. In general, recursive languages are closed under union operation.

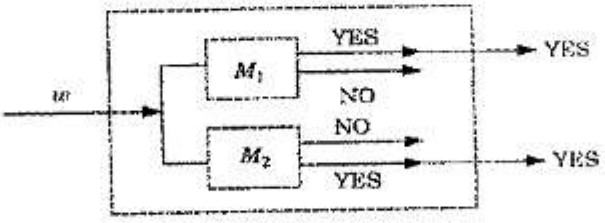
3. A language is recursive if and only if its complement is recursive.

4. The union of two recursively enumerable languages is recursive enumerable.

Proof : Let L_1 and L_2 be two recursively enumerable languages and recognized by M_1 and M_2 Turing machines. We construct another TM M_3, which accepts either L_1 or L_2. Now, as we know the problem about recursive enumerable languages that if w is not in L_1 and L_2, then M_3 can not decide. So, the problem of recursive languages is persistent with M_3 also. So, $N(M_3)$ is recursive enumerable language and hence $L_1 \cup L_2$ is recursive enumerable languages. In general, recursive enumerable languages are closed under union operation.

5. If a language L over some alphabet Σ and its complement $\overline{L} = \Sigma^* - L$ is recursive enumerable, then L and $\overline{L}$ are recursive languages.

Proof : We construct two Turing machines M_1 for L and M_2 for $\overline{L}$. Now, we construct a third TM M_3 based on M_1 and M_2 as shown in figure(e). TM M_3 accepts w if TM M_1 accepts and rejects w if M_2 accepts. It means, if $w \in L$, then w is accepted and if $w \notin L$ then it is rejected. Since, for all w, either w is accepted or rejected. Hence, M_3 is based on algorithm and produces either "YES" or "NO" for input string w, but not both. It means, M_3 decides all the strings over Σ. Hence, L is recursive. As we know that complement of a recursive language is also recursive and hence $\overline{L}$ is also recursive.



Figure(e)

6. We have following co - theorem based on above discussion for recursive enumerable and recursive languages.

Let L and $\overline{L}$ are two languages, where $\overline{L}$ the complement of L, then one of the following is true :

(a) Both L and $\overline{L}$ are recursive languages,

(b) Neither L nor $\overline{L}$ is recursive languages,

(c) If L is recursive enumerable but not recursive, then $\overline{L}$ is not recursive enumerable and vice versa.

Undecidable Problems about Turing Machines

In this section, we will first discuss about halting problem in general and then about TM.

Halting Problem (HP)

The **halting problem** is a decision problem which is informally stated as follows :

"Given a description of an algorithm and a description of its initial arguments, determine whether the algorithm, when executed with these arguments, ever halts. The alternative is that a given algorithm runs forever without halting."

Alan Turing proved in 1936 that there is no general method or algorithm which can solve the halting problem for all possible inputs. An algorithm may contain loops which may be infinite or finite in length depending on the input and behaviour of the algorithm. The amount of work done in an algorithm usually depends on the input size. Algorithms may consist of various number of loops, nested or in sequence. The HP asks the question :

Given a program and an input to the program, determine if the program will eventually stop when it is given that input ?

One thing we can do here to find the solution of HP. Let the program run with the given input and if the program stops and we conclude that problem is solved. But, if the program doesn't stop in a reasonable amount of time, we can not conclude that it won't stop. The question is : " how long we can wait .... ?" . The waiting time may be long enough to exhaust whole life. So, we can not take it as easier as it seems to be. We want specific answer, either "YES" or "NO", and hence some algorithm to decide the answer.

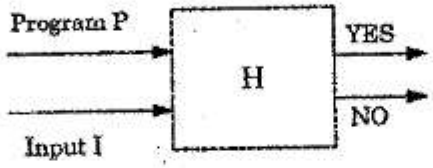
The importance of the halting problem lies in the fact that it is the first problem which was proved undecidable. Subsequently, many other such problems have been described.

Theorem : HP is undecidable.

Proof : This proof was devised by Alan Turing in 1936. Initially, we assume that HP is decidable and the algorithm (solution) for HP is H. The halting problem solution H takes two inputs :

1. Description of TM M i. e. program P and
2. Input I for the program P.

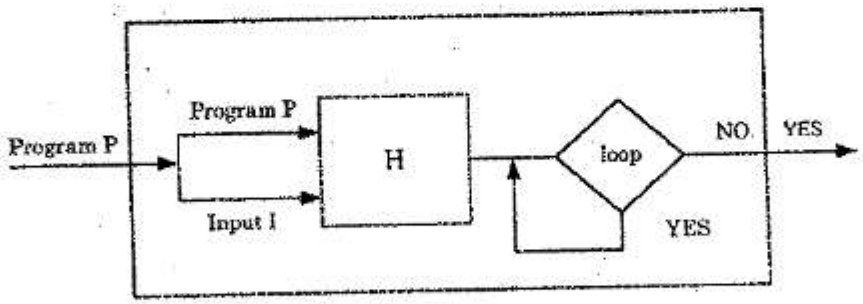
H generates an output "YES" if H determines that P stops on input I or it outputs "NO" if H determines that P loops as shown in figure(a).



Figure(a)

Note : When an algorithm is coded, it is expressed as a string of characters . Input is also coded into the same format. So, after coding, a program and data have no difference in their format of representation and so, a program can be treated a data sometimes and a data can be treated as a program sometimes.

So, now H can be modified to take P as both inputs (the program and its input) and H should be able to determine if P will halt on P as it's input shown in figure(b).



Figure(b)

Let us construct a new, simple algorithm Q that takes output of H as its input and does the following:

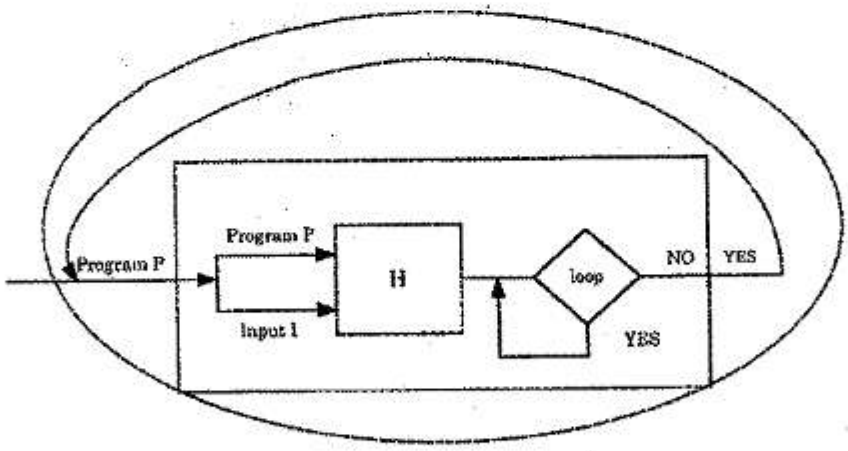
1. If H outputs "NO" then Q outputs "YES" and halts.
2. Otherwise H's output "YES" causes Q to loop forever.

If means, Q does the opposite what H does.

We define Q as follows:

```
Function Q( )
{
   if ( Function H ( ) = " NO")
   {
         return ("YES");
   }
   else
   {
         while (1);              // Loop for ever
   }
}        // End of the function Q
```

Since, Q is a program, now let us use Q as the input to itself as shown in figure(c).



Figure(c)

Now, we analyse the following :

1. If H outputs "YES" and says that Q halts then Q itself would loop (that's how we constructed it).
2. If H outputs "NO" and says that Q loops then Q outputs "YES" and will halts.

Since , in either case H gives the wrong answer for Q. Therefore, H cannot work in all cases and hence can't answer right for all the inputs. This contradicts our assumption made earlier for HP. Hence, HP is undecidable.

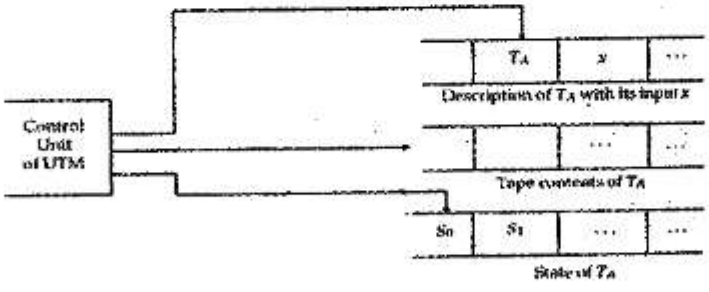
Theorem : HP of TM is undecidable.
Proof : HP of TM means to decide whether or not a TM halts for some input w. We can prove this following the similar steps discussed in above theorem.

8.6 UNIVERSAL TURING MACHINE

The Church - Turing thesis conjectured that anything that can be done on any existing digital computer can also be done by a TM. To prove this conjecture. A. M. Turing was able to construct a single TM which is the theoretical analogue of a general purpose digital computer. This machine is called a Universal Turing Machine (UTM). He showed that the UTM is capable of initiating the operation of any other TM, that is, it is a reprogrammable TM. We can define this machine in more formal way as follows :

Definition : A Universal Turing Machine (denoted as UTM) is a TM that can take as input an arbitrary TM T_A with an arbitrary input for T_A and then perform the execution of T_A on its input.

What Turing thus showed that a single TM can acts like a general purpose computer that stores a program and its data in memory and then executes the program. We can describe UTM as a 3 - tape TM where the description of TM, T_A and its input string $x \in A^*$ are stored initially on the first tape, t_1. The second tape, t_2 used to hold the simulated tape of T_A, using the same format as used for describing the TM, T_A. The third tape , t_3 holds the state of T_A



To construct a UTM, we thus require three essentials, viz.,

(i) a uniform method to describe or encode any TM into a string over a finite symbol set, I.
(ii) a similar method of encoding any input string for a TM into a string over I, and
(iii) a set of TM programs (i. e., a set of instructions for any TM) that describe the TMs basic cycle of operations.

Encoding an arbitrary TM

Since a TM can have only a finite number of configurations defined by $\langle s, a, b, s', d \rangle$, we can describe or encode any TM in terms of fixed symbols of universal Turing machine.

Let the internal states of a TM, T_A, is given by

$$S = \{S_0, S_1, S_2, ..., S_{n-1}, S_n\}$$

where S_0 is the initial state, and $S_n = H$, halting state.

Also, let the set of tape symbols be $A = \{a_0, a_1, ..., a_{m-1}\}$, where $a_0 = B$, a blank character.

We define the encoding for T_As configurations as follows :

Original	Code
S_i	1^{i+1}
a_j	1^{i+j}
R	R
L	L
N	N

We use the symbol '0' as a separator between each encoded symbol of a configuration.

For example, in the TM for parity checking , we have

$$S = \{S_0, S_1, S_2, H\}, \text{ and}$$
$$A = \{B, 0, 1, E, D,\}$$

therefore, the encoding for the configuration $\langle S_1, B, D, H, N \rangle$ will be 11010111110111 10N.

Now, suppose that a Turing machine, T_A, is consisting of a finite number of configurations, denoted by, $c_0, c_1, c_2, \ldots, c_p$ and let $\bar{c}_0, \bar{c}_1, \bar{c}_2, \ldots, \bar{c}_p$ represent the encoding of them. Then, we can define the encoding of T_A as follows :

$$* \bar{c}_0 \# \bar{c}_1 \# \bar{c}_2 \# \ldots\ldots \# \bar{c}_p *$$

Here, * and # are used only as separators, and cannot appear elsewhere. We use a pair of *'s to enclose the encoding of each configuration of TM, T_A.

The case where $\delta(s,a)$ is undefined can be encoded as follows :

$$\# \bar{s}\ 0\bar{a}\ 0\bar{B}\ \#$$

where the symbols $\bar{s}$, $\bar{a}$ and $\bar{B}$ stand for the encoding of symbols, s, a and B (Blank character), respectively.

Working of UTM

Given a description of a TM, T_A and its inputs representation on the UTM tape, t_1 and the starting symbol on tape, t_3, the UTM starts executing the quintuples of the encoded TM as follows :

1. The UTM gets the current state from tape, t_3 and the current input symbol from tape t_2.
2. then, it matches the current state - symbol pair to the state symbol pairs in the program listed on tape, t_1.
3. if no match occurs, the UTM halts, otherwise it copies the next state into the current state cell of tape, t_3, and perform the corresponding write and move operations on tape, t_2.
4. if the current state on tape, t_3 is the halt state, then the UTM halts, otherwise the UTM goes back to step 2.

8.7 POST'S CORRESPONDENCE PROBLEM (PCP)

Post's correspondence problem is a combinatorial problem formulated by Emil Post in 1946. This problem has many applications in the field theory of formal languages.

Definition :

A correspondence system P is a finite set of ordered pairs of nonempty strings over some alphabet.

Let Σ be an alphabet, then P is finite subset of $\Sigma^* \times \Sigma^+$. A match or solution of P is any string $w \in \Sigma$ such that pairs $(u_1, v_1), (u_2, v_2), ..., (u_n, v_n) \in P$ and $w = u_1 u_2 .... u_n = v_1 v_2 .... v_n$ for some n > 0. The selected pairs $(u_1, v_1), (u_2, v_2), ..., (u_n, v_n)$ are not necessarily distinct.

Let strings $u_1, u_2, ..., u_m$ are in U and strings $v_1, v_2, ..., v_m$ are in V, then

$U = \{ u_1, u_2, ..., u_m \}$ and $V = \{ v_1, v_2, ..., v_m \}$ for some m > 0.

PCP is to determine whether there is any match or not for a given correspondence system.

Theorem : PCP is undecidable

PCP is undecidable just like the HP of Turing machine.

Example 1: A correspondence system $P = \{ (b, a), (ba, ba), (bab^3, b^3) \}$.
Is there any solution for P ?

Solution : We represent the P as follows :

i	u_i	v_i
1	b	a
2	ba	ba
3	bab^3	b^3

Here, $u_1 = b$, $u_2 = ba$, $u_3 = bab^3$, $v_1 = a$, $v_2 = ba$, $v_3 = b^3$.

We have a solution $w = u_3 u_1 u_1 u_2 = v_2 v_3 v_3 v_2 = bab^3 b^3 a$.

Example 2 : Consider a correspondence system P = { (b, ca), (a, ab), (ca, a), (abc, c) }.
Find a match (if any).

Solution : We represent the P as follows :

i	u_i	v_i
1	b	ca
2	a	ab
3	abc	c

Here, $u_1 = b,\ u_2 = a,\ u_3 = abc,\ v_1 = ca,\ v_2 = ab,\ v_3 = c$.

We have a solution $w = u_3\ u_2 = v_2\ v_1 = abca$.

8.8 TURING REDUCIBILITY

Reduction is a technique in which if a problem A is reduced to problem B then any solution of B solves A. In general, if we have an algorithm to convert some instance of problem A to some instance of problem B that have the same answer then it is called A reduces to B.

FIGURE: Reduction

Definition : Let A and B be the two sets such that $A, B \subseteq N$ of natural numbers. Then A is Turing reducible to B and denoted as $A \leq_T B$.

If there is an oracle machine that computes the characteristic function of A when it is executed with oracle machine for B.

This is also called as A is B - recursive and B - computable. The oracle machine is an abstract machine used to study decision problem. It is also called as **Turing machine** with **black box.** We say that A is Turing equivalent to B and write $A \equiv_T B$ if $A \leq_T B$ and $B \leq_T A$.

Properties :

1. Every set is Turing equivalent to its complement.
2. Every computable set is Turing equivalent to every other computable set.
3. If $A \leq_T B$ and $B \leq_T C$ then $A \leq_T B$.

8.9 DEFINITION OF P AND NP PROBLEMS

A problem is said to be solvable if it has an algorithm to solve it. Problems can be categorized into two groups depending on time taken for their execution.

1. The problems whose solution times are bounded by polynomials of small degree. **Example:** bubble sort algorithm obtains n numbers in sorted order in polynomial time $P(n) = n^2 - 2n + 1$ where n is the length of input. Hence, it comes under this group.

2. Second group is made up of problems whose best known algorithm are non polynomial example, travelling salesman problem has complexity of $O(n^2\ 2^n)$ which is exponential. Hence, it comes under this group.

A problem can be solved if there is an algorithm to solve the given problem and time required is expressed as a polynomial p(n), n being length of input string. The problems of first group are of this kind.

The problems of second group require large amount of time to execute and even require moderate size so these problems are difficult to solve. Hence, problems of first kind are tractable or easy and problems of second kind are intractable or hard.

8.9.1 P - Problem

P stands for deterministic polynomial time. A deterministic machine at each time executes an instruction. Depending on instruction, it then goes to next state which is unique.

Hence, time complexity of deterministic TM is the maximum number of moves made by M is processing any input string of length n, taken over all inputs of length n.

Definition : A language L is said to be in class P if there exists a (deterministic) TM M such that M is of time complexity P(n) for some polynomial P and M accepts L.
Class P consists of those problem that are solvable in polynomial time by DTM.

8.9.2 NP - Problem

NP stands for nondeterministic polynomial time.

The class NP consists of those problems that are verifiable in polynomial time. What we mean here is that if we are given certificate of a solution then we can verify that the certificate is correct in polynomial time in size of input problem.

Example :

Hamiltonian circuit problem. Given a directed graph $G=<V, E>$, a certificate would be a sequence $<V_1,V_2,V_3,....V_n>$ of $|V|$ vertices. It is easy to verify in polynomial time that $(V_i+V_j+1) \in E$ for i = 1, 2, ... |V| -and $(V_{|V|}) \in E$ as well using a nondeterministic algorithm. Hence it is in class NP. There doesnot appear any deterministic algorithms to recognize those graphs with Hamiltonian circuit. Hence it is not in class P.

A nondeterministic machine has a choice of next steps. It is free to choose any move that it wishes and if the problem has a solution one of these steps will lead to solution.

Definition : A language L is in class NP if there is a nondeterministic TM such that M is of time complexity P(n) for some polynomial P and M accepts L.

The difference between P and NP problems is analogous to difference between efficiently finding a proof of a statement (such as "This graph has Hamiltonian circuit") and efficiently verifying a proof of a statement ("i. e., checking a particular circuit is Hamiltonian"). It is easier to check a proof than finding a one.

In other words class NP consists of problems for which solution are verified quickly. P consists of problems which can be solved quickly.

Any problem in P is also in NP, but it is not yet known that P = NP. Hence, commonly believed relationship between P and NP is,

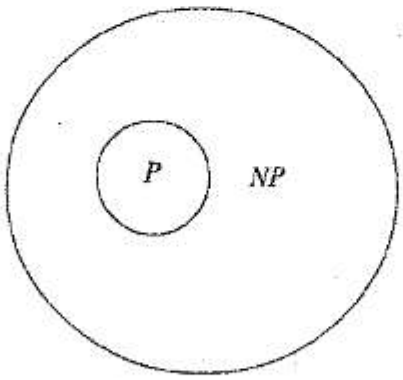


FIGURE: Relationship between P and NP Problems

8.10 NP - COMPLETE AND NP - HARD PROBLEMS

A problem S is said to be NP- Complete problem if it satisfies the following two conditions.

1. $S \in NP$, and
2. For every other problems $S_i \in NP$ for some $i = 1, 2, n$, there is polynomial - time transformation from S_i to S i.e. every problem in NP class polynomial - time reducible to S.

We conclude one thing here that if S_i is NP - complete then S is also NP - Complete.

As a consequence, if we could find a polynomial time algorithm for S, then we can solve all NP problems in polynomial time, because all problems in NP class are polynomial - time reducible to each other.

"A problem P is said to be NP - Hard if it satisfies the second condition as NP - Complete, but not necessarily the first condition .".

The notion of NP - hardness plays an important role in the discussion about the relationship between the complexity classes P and NP. It is also often used to define the complexity class NP - Complete which is the intersection of NP and NP - Hard. Consequently, the class NP - Hard can be understood as the class of problems that are NP - complete or harder.

Example : An NP - Hard problem is the decision problem SUBSET - SUM which is as follows.

" Given a set of integers, do any non empty subset of them add up to zero? This is a yes / no question, and happens to be NP - complete ".

There are also decision problems that are NP - Hard but not NP - Complete , for example, the halting problem of Turing machine. It is easy to prove that the halting problem is NP - Hard but not NP - Complete. It is also easy to see that halting problem is not in NP since all problems in NP are decidable but the halting problem is not (voilating the condition first given for NP - complete languages).

In Complexity theory, the **NP - complete** problems are the hardest problems in NP class, in the sense that they are the ones most likely not to be in P class. The reason is that if we could find a way to solve any NP - complete problem quickly, then you could use that algorithm to solve all NP problems quickly.

At present time, all known algorithms for NP - complete problems require time which is exponential in the input size. It is unknown whether there are any faster algorithms for these are not.

S. A. Cook in 1971 proved that the Boolean satisfiability problem is NP - Complete. After Cook's original results, thousands of other problems have been shown to be NP - complete by reductions from other problems previously shown to be NP - complete.

Example : Consider an interesting problem in graph theory known as " Graph isomorphism". Two graphs are isomorphic if one can be transformed into the other simply by renaming vertices. Consider these two problems given as follows :

Graph Isomorphism : Is graph G_1 isomorphic to graph G_2 ?

Subgraph Isomorphism : Is graph G_1 isomorphic to a subgraph of graph G_2 ?

The " Subgraph Isomorphism" problem is NP - complete, but the " Graph Isomorphism" problem is suspected to be neither in P nor in NP - Complete, though it is obviously in NP. This is an example of a problem that is thought to be hard, but it is not thought to be NP - Complete.

Following are some other NP - complete and NP - Hard problems :

(1) The Boolean Satisfiability Problem (SAT)

In mathematics, a formula of propositional logic is said to be satisfiable if truth - values can be assigned to its free variables in such a way that this assignment makes the formula true. The class of satisfiable propositional formulae is NP - Complete problem.

Consider the logical operators defined as follows :

And : This is denoted by $\wedge$ and $0 \wedge 1 = 1 \wedge 0 = 0$, $0 \wedge 0 = 0, 1 \wedge 1 = 1$,

OR : This is denoted by $\vee$ and $0 \vee 1 = 1 \vee 0 = 1$, $1 \vee 1 = 1, 0 \vee 0 = 0$, and

NOT : This is denoted by ' and $0' = 1, \ 1' = 0$.
Now, consider the expressions

(a) $E_1 = x' \vee y$, where x, y are variables ; either 0 or 1 So, $E_1 = 1$ if $x = 0$ or $y = 1$
Therefore, E_1 is satisfiable for $x = 0$ or $y = 1$.

(b) $E_2 = (x \vee y) \wedge x' \wedge y'$ is not satisfiable because every assignment for the variables x and y will make the value of $E_2 = 0$.

(2) The Travelling Salesman or Salesperson Problem

The problem is defined as follows .

"Given a number of cities and the cost of travelling from one to the other, what is the cheapest roundtrip route that visits each city and then returns to the starting city ?"

The most direct answer would be to try all the combinations and see which one is cheapest, but given that the number of combinations of cities is n ! (factorial n), this solution becomes impractical for larger n, where n is the number of cities.

How fast are the best known deterministic algorithms ?

This problem has been shown to be NP - Hard, and the decision version of it which is given below :

" Given the costs of routes between cities and a number N, decide whether there exists a tour program for salesman to visit all the cities so that the total cost is less than or equal to N."

The above version of salesman problem is NP - Complete problem.

(3) The Hamiltonian Cycle or Hamiltonian Circuit Problem

This problem is in graph theory to find a path through a given graph which starts and ends at the same vertex and includes each vertex exactly once.

This is a special case of the travelling salesman problem obtained by setting the distance between two cities to unity if they are adjacent and infinity otherwise. Like the traveling salesman problem, the Hamiltonian cycle problem is NP - Complete.

(4) The Vertex Cover Problem

This problem is stated as follows .

" Given a graph G and a natural number K, does there exist a vertex covering for G with K vertices."

This is NP - complete problem.

REVIEW QUESTIONS

Q1. Explain about chomsky hierarchy of languages.

Answer :

For Answer refer to Topic : 8.1, Page No : 8.1.

Q2. Explain about LBA with an example.

Answer :

For Answer refer to Topic : 8.2, Page No : 8.5.

Q3. Explain about CSLs with an example.

Answer :

For Answer refer to Topic : 8.3 , Page No : 8.8.

Q4. Write a short notes on LR(0) grammar.

Answer :

For Answer refer to Topic : 8.4.1, Page No : 8.11.

Q5. Write a short notes on Decidability.

Answer :

For Answer refer to Topic : 8.5, Page No : 8.16.

Q6. Explain Halting problem.

Answer :

For Answer refer to Page No : 8.22.

Q7. Briefly describe universal Turing machine.

Answer :

For Answer refer to Topic : 8.6, Page No : 8.25.

Q8. Explain PCP in detail.

Answer :

For Answer refer to Topic : 8.7, Page No : 8.27.

Q9. Explain the Turing reducibility in detail.

Answer :

For Answer refer to Topic : 8.8, Page No : 8.29.

Q10. Discuss about P and NP problems.

Answer :

For Answer refer to Topic : 8.9, Page No : 8.29.

Q11. Giving relevant examples explain NP Hard and NP complete problems.

Answer :

For Answer refer to Topic : 8.10, Page No : 8.32.

9. Choose which of the following is not correct:
   (a) Recursive languages are closed under complementation
   (b) Set of recursively enumerable languages is closed under union
   (c) If a language and its complement are both regular, then the language must be recursive
   (d) None of the above.
10. Given the following statements:
   (i) Every monotonic grammar G is equivalent to a type 1 grammar.
   (ii) A context sensitive language is recursive.
   (iii) There exists a recursive set which is not a context sensitive language over {0, 1}.
   Which of the following statements are true?
   (a) All (i), (ii) and (iii) (b) Only (i) and (iii)
   (c) Only (ii) and (iii) (d) Only (i) and (ii)
11. Which one is false:
   (a) L is recursive then TM halts for every x belongs to L.
   (b) If L & Complement L both are recursive Enumerable then L is recursive:
   (c) Complement of a recursive language is again recursive
   (d) Every recursive enumerable language is recursive.
12. The family of recursive languages is not closed under which of the following operations:
   (a) Complementation (b) Union
   (c) Intersection (d) None of these.
13. Any language generated by an unrestricted grammar is
   (a) Not recursively enumerable (b) Recursive
   (c) Recursively enumerable (d) None of these.
14. Let A= set of recursive languages B=set of recursively enumerable languages. Then
   (a) A and B are disjoint sets. (b) A and B are the same set
   (c) B is a subset of A (d) A is a subset of B
15. Which of the following statements is true?
   (a) A context sensitive language is recursive.
   (b) A set X is recursive if we have a algorithm to decide whether a given element belongs to X or not.
   (c) A recursive set is recursively enumerable.
   (d) All of the above.

16. The union of two recursively enumerable languages is:
(a) recursive enumerable (b) recursive
(c) not (b) (d) none

17. Which of the following properties of recursively enumerable set is not recursively enumerable,
(a) L contains atleast 10 members (b) $L - L_u \neq \phi$
(c) $L \neq \phi$ (d) $L = \Sigma^*$

18. Which of the following properties of recursively enumerable set is recursively enumerable.
(a) $L = \Sigma^*$
(b) L is recursive
(c) $L - L_u \neq \phi$, (L_u *is the universal language*)
(d) $L = \phi$

19. Universal Language is:
(a) recursive (b) decidable
(c) non-recursively enumerable (d) recursively enumerable

20. Let L *and* L' be a pair of complementary languages then,
(a) One of L and is recursively enumerable but not recursive & the other is not recursive enumerable
(b) Both L *and* L' are recursive
(c) Neither L *nor* L' is recursively enumerable
(d) Any one of the above.

21. Which of the following statements are false:
(a) If a language L and its complement L are both recursively enumerable, then L (*and hence* L') is recursive.
(b) The complement of a recursive language is recursive.
(c) The union of two recursive langauge is recursive. The union of two recursively enumerable languages is recursively enumerable.
(d) None.

22. The complement of recursive is :
(a) can't say (b) recursive (c) non-recursive (d) none.

23. A problem whose language is recursive is said to be:
(a) can't say (b) decidable (c) undecidable (d) none.

24. Select the false statement:
(a) The blank-tape halting problem is undecidable
(b) The Turing Machine Halting problem is Undecidable
(c) The Turing Machine Halting problem is decidable
(d) None of the above.

25. Which of the following statement is correct?
(a) If the emptiness problem is undecidable for Type 0 grammars, then it is also undecidable for Type 3 grammars.
(b) If the emptiness problem is decidable for Type 3 grammars, then it is also decidable for Type 0 grammars
(c) If the emptiness problem is decidable for Type 0 grammars, then it is also decidable for Type 3 grammars.
(d) None of the above.

26. The problem of determining that a Turing Machine would halt after giving a Yes/No output is
(a) Decidable (b) Unsolvable (c) Solvable (d) None of the above.

27. Which of the following subset relation doesn't hold?
(a) $L_0 \subseteq L_r$ (b) $L_{cs} \subseteq L_0$ (c) $L_{cf} \subseteq L_{cs}$ (d) $L_r \subseteq L_{cf}$

28. Let G be any unrestricted grammar. Then the problem of determining whether or not $L(G) = \phi$ is
(a) Cannot say (b) Decidable
(c) Undecidable (d) None of these.

29. What can you say about the membership problem for Type 0 grammars?
(a) Partially decidable (b) Decidable
(c) Undecidable (d) None of the above.

30. A Language is said to be recursive if:
(a) it is not recursively enumerable and its complement is not recursively enumerable.
(b) it is recursively enumerable and its complement is recursively enumerable.
(c) it is not recursively enumerable and its complement is recursively enumerable
(d) it is recursively enumerable and its complement is not recursively enumerable.

31. Let L be a language which is not recursively enumerbale. Then complement of L must be
(a) Not recursive (b) Recursive
(c) Recursively enumerable (d) None of these.

32. What can you say about the membership problem for Type 0 grammars?
(a) Partially decidable (b) Decidable
(c) Undecidable (d) None of the above.

33. The problem of determining whether or not a TM over {0,1} will ever print the symbol 1, with a given tape configuration is
(a) Unsolvable (b) Solvable (c) None of these

34. Given an arbitrary Turing machine T, it is undecidable "whether T accepts the empty word $\in$?"
(a) True (b) False

35. Given a type 0 grammar G and a word w, it is undecidable "whether G generates w?"
(a) True (b) False

36. Given a Turing Mahine T and word w, it is undecidable "whether T halts when the string w is placed on the input tape?"
(a) True (b) False

37. Let G be a context-sensitive grammar. It is decidable "whether any word $x \in L(G)$?"
(a) True (b) False

38. It is decidable "whether $L(G)$ is infinite, where G is any context-free grammar?"
(a) True (b) False

39. It is decidable "whether $L(G) = \phi$, where G is any context free grammar?"
(a) True (b) False

40. It is decidable "whether a given right-linear grammar $G_1 = \{N_1, T_1, S_1, P_1\}$ contains any useless non-terminals?"
(a) True (b) False

41. Find the odd man out
(a) Post Correspondence Problem.
(b) Blank Tape halting problem
(c) State-entry problem.
(d) The halting problem of turing machines

42. Decidability in Decidable and Undecidable Problems refers to
(a) Those problems whose answer are only either in "yes" or "no".
(b) Existence of algorithm which takes an instance of the problem and determines whether the answer to that instance is "yes" or "no"
(c) Existence of algorithm which generates the set of solutions.
(d) Unsolvability

43. Which of the following is decidable?
(a) The problem to decide whether CFG is ambiguous or not.
(b) The modified Post Correspondance Problem
(c) The problem to determine whether a string is in G or not, where G is unrestricted grammar.
(d) None of the above.

44. Which of the following is decidable?
(a) The problem to decide whether CFG is ambiguous or not.
(b) The modified Post Correspondance Problem
(c) The problem to determine whether a string is in G or not, where G is unrestricted grammar.
(d) None of the above.

45. Which of the following is decidable?
(a) The question whether or not $L(M)$ is finite, where M is a Turing Machine.
(b) The halting problem
(c) The problem of determining whether or not $L(G) = \phi$ where G is unrestricted grammar.
(d) None of above.

46. For post Correspondance problem(PCP) and Modified PCP(MPCP)
(a) Both are equivalent
(b) Both are unrelated
(c) If there exists MPCP solution then there is a PCP solution
(d) If there exists PCP solution then there is a MPCP solution

47. If we have a procedure to determine whether a given element belongs to a set X or not, then this set is called.....
(a) Context-sensitive (b) Complete.
(c) Recursively Enumerable (d) .Recursive

48. If we have an algorithm to determine whether a given element belongs to a set X or not, then this set is called.....
(a) Context-sensitive (b) Complete.
(c) Recursively Enumerable (d) .Recursive

49. If 2 lists are $x = \{b, bab^3, ba\}$ *and* $y = \{b^3, ba, a(a)\}$ then solution of PCP problem is:
(a) 1,2,3,3 (b) 2,1,1,1,3 (c) 2,1,1,3 (d) none

50. If PCP is decidable then MPCP is
(a) Can't say (b) decidable
(c) Undecidable (d) none

51. Which of the following statements is true?
(a) pcp with $x = (b^3, ab^2)$ *and* $y = (b^3, bab^3)$ *has a solution*
(b) PCP with {(01,011),(1,10),(1,11)} has a solution.
(c) PCP with {(0,10),(120,03),(021,10)} has a solution
(d) None of the above.

52. Which of the following statements is false?
(a) PCP over Σ *for* $|\Sigma| \geq 2$ is unsolvable
(b) PCP with two lists $x = (b, bab^3, ba)$ *and* $y = (b^3, ba, a)$ has no solution.
(c) PCP with two lists $x = (01,1,1)$ *and* $y = (01^2, 10, 1^2)$ has no solution.
(d) None of the above.

53. What can you say about the membership problem for Type1 grammars?
(a) Partially decidable (b) Decidable
(c) Undecidable (d) None of the above.

54. PCP is
(a) Sometimes undecidable (b) Decidable
(c) Undecidable (d) None.

55. Which of the following instances of the Post Correspondence Problem have a viable sequence?
(i) $\{(ab, abb), (ba, aaa), (aa, a)\}$
(ii) $\{(ab, aba), (baa, aa), (aba, baa)\}$
(iii) $\{(b, bb), (bb, bab), (bab, abb), (abb, babb)\}$
(a) 1,2 and 3. (b) 1 and 2 only (c) 2 only (d) 1 only

56. If L_1 and L_2 are any two context free languages (type 2) over an alphabet Σ and $|\Sigma| \geq 2$, there is no algorithm to determine whether or not
(a) $L_1 = L_2$ (b) $L_1 \subseteq L_2$
(c) $L_1 \cap L_2$, is a context free language (d) $L_1 \cap L_2 = \phi$
(e) All of the above (f) None of the above

57. The membership problem for Type 0 grammars is ?
(a) Partially decidable (b) Decidable
(c) Undecidable (d) None of the above

58. PCP is decidable for two set of strings w and v, if
(a) The strings consist of just one character repeated any no of times i.e., input symbol set is a sigleton
(b) Corresponding strings in w and v are of equal length
(c) Both the above.
(d) It is never decidable.

59. Which of the following problems is decidable?
(a) Whether L(M) is finite for a TM'M'
(b) Halting problem of TM
(c) Whether L(G) is empty for unrestricted grammar G
(d) None

60. The statement "halting problem is unsolvable"is
(a)still an open question (b) true
(c) false (d) All

61. Inherent ambiguity for CFLs is :
(a) Can't say (b) Decidable (c) Undecidable (d) none

62. If a problem P_1 is reducible to P_2 then which of the following does not hold?
(a) If P_2 is solvable then so is P_1.
(b) P_1 is at least as hard as P_2.
(c) If P_1 is non-recursively enumerable then so is P_2.
(d) If P_1 is unsolvable then so is P_2

63. Which of the following problems is decidable :
(a) Is a given CFL equal to Σ^* ? (b)Are 2 CFls the same?
(c) Is the intersection of 2 CFls empty? (d) Is a given grammar ambiguous?
(e) None of the above

64. Consider, the following modifications of the Post Correspondence Problem :
(i) There is an MPC-Solution if there is a sequence of integers such that
$w_i w_j ... w_k w_l = v_i v_j ... v_k v_l$
(ii) There is an MPC-Solution if there is a sequence of integers such that $w_1 w_2 w_i w_j ... w_k = v_1 v_2 v_i v_j ... v_k$. Then,
(a) 1 and 2 are un-decidable
(b) 1 is un-decidable but 2 is decidable
(c) 1 is decidable but 2 is un-decidable
(d) 1 and 2 are decidable

ANSWER KEY

1.(d)	2.(a)	3.(a)	4.(d)	5.(b)	6.(c)	7.(c)	8.(d)	9.(b)	10.(d)
11.(c)	12.(d)	13.(b)	14.(a)	15.(c)	16.(c)	17.(b)	18.(c)	19.(a)	20.(d)
21.(b)	22.(d)	23.(c)	24.(c)	25.(c)	26.(b)	27.(d)	28.(c)	29.(d)	30.(c)
31.(a)	32.(a)	33. (a)	34.(a)	35.(a)	36.(a)	37.(a)	38.(a)	39.(a)	40.(a)
41.(a)	42.(b)	43.(d)	44.(c)	45.(d)	46.(a)	47.(c)	48.(d)	49.(c)	50.(a)
51.(d)	52.(d)	53.(b)	54.(c)	55.(d)	56.(c)	57.(b)	58.(d)	59.(a)	60.(c)
61.(b)	62.(b)	63.(e)	64.(a)						