FOLLOW US ON...

@brocodeengineeringofficial

Brocode Engineering

@brocodeengineering

Brocode Engineering

Contents

THEORY OF COMPUTATION (V-SEM., CS-BRANCH)

PAGE NO.

UNIT - I : Introduction of Automata Theory

Examples of automata machines, Finite Automata as a language acceptor and translator, Moore machines and Mealy machines, composite machine, Conversion from Mealy to Moore and vice versa .....(03 to 36)

UNIT - II : Types of Finite Automata

Non-Deterministic Finite Automata (NDFA), Deterministic finite automata machines, conversion of NDFA to DFA, minimization of automata machines .....(37 to 57)

Regular expression, Arden's theorem, Meaning of union, intersection, concatenation and closure, 2 way DFA .....(58 to 100)

UNIT - III : Grammars

Types of grammar, context sensitive grammar, and context free grammar, regular grammar, Derivation trees, ambiguity in grammar, simplification of context free grammar, conversion of grammar to automata machine and vice versa, Chomsky hierarchy of grammar, killing null and unit productions, Chomsky normal form and Greibach normal form .....(101 to 150)

UNIT - IV : Push down Automata

Example of PDA, deterministic and non-deterministic PDA, conversion of PDA into context free grammar and vice versa, CFG equivalent to PDA, Petrinet model .....(151 to 184)

UNIT - V : Turing Machine

Techniques for construction, Universal Turing machine Multitape, multihead and multidimensional Turing machine .....(185 to 226)

N-P complete problems, Decidability and Recursively Enumerable Languages, decidability, decidable languages, undecidable languages, Halting problem of Turing machine & the post correspondence problem .....(226 to 248)

UNIT 1 INTRODUCTION OF AUTOMATA THEORY

EXAMPLES OF AUTOMATA MACHINES, FINITE AUTOMATA AS A LANGUAGE ACCEPTOR AND TRANSLATOR, MOORE MACHINES AND MEALY MACHINES, COMPOSITE MACHINE, CONVERSION FROM MEALY TO MOORE AND VICE VERSA

Q.1. What do you mean by theory of computation ?

Ans. Theory of computation is a branch of computer science that deals with how efficiently a problem can be solved on a model of computation using algorithm.

Q.2. Write short note on automata. ***(R.G.P.V., June 2010)***

Ans. An ***automaton*** is an abstract model of a digital computer. Every automaton includes some essential features. It has a mechanism for reading input. It is assumed that the input is a string over a given alphabet, written on a file known as ***input tape***, which the automaton can only read. This file is divided into cells, each of which can hold one symbol. The input mechanism can scan (or read) the file from left to right, one symbol at a time. Also the input mechanism can detect the end of the input string (sensing end-of-file condition). The output of some form can be produced by automaton. The automaton may have a temporary ***storage*** device, having an unlimited number of cells, each of which can hold a single symbol from an alphabet (not necessarily the same as the input alphabet). The automaton can read as well as change the contents of the storage cells. Finally, it has a ***control unit***, which can be in any one of a finite number of ***internal states***. The control unit can change state in some defined manner. Fig. 1.1 shows a diagram of a general automaton.

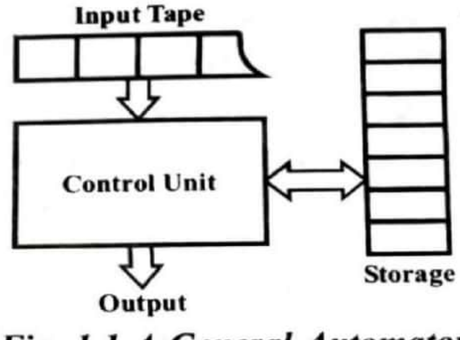


Fig. 1.1 A General Automaton

An automaton is supposed to operate in a discrete time frame. At some given time, the control unit is in some internal state, and the input mechanism is scanning a specific symbol on the input tape. The internal state of the control unit at the next step is obtained by the ***next-state*** or ***transition function.*** This transition function gives the next state in terms of the current state, the current input symbol on the input tape, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be generated or the information in the temporary storage changed. The term ***configuration*** refers to a particular state of the control unit, input tape, and temporary storage. The transition from one configuration to the next is called a ***move.***

Q.3. What do you mean by basic machine ? Explain it with example.

Ans. If we have to recognize the given set of input and generate the output than it will be a machine which has a very limited capacity is known as ***basic machine***.

Basic machine recognizes input 'I' and produce output 'O'.

It works as if a function to which input is argument and output is result. There is a one to one mapping from input to output. Such a function relating input and output is known as machine function (MAF).

Example – If we received 2 inputs and we have to generate output as an 'AND' gate then,

where, input, $I = \{i_1, i_2\}$

Output = O

I	(0, 0)	(0, 1)	(1, 0)	(1, 1)
O	0	0	0	1

The basic machine has a limited capacity, it will only react on the input. To improve the capacity of the machine it is very necessary then we should consider the state of mind during the design of the machine. There are internal states of the machine.

Q.4. Define finite state machine (FSM) with example.

Ans. This machine will consists of input output relation at every state and also the changes of the states that will occur on receiving the input at a particular state. Hence we will require the mapping in two forms. At a particular state, for a given input, what is the output.

$$S \times I \rightarrow O$$

and at a particular state on receiving the input, what is the next state.

$$S \times I \rightarrow S$$

These are respectively known as machine function (M.A.F.) and state function (S.T.F.). Both of them are functions of two arguments, current state, and current input but their results are different. The machine can be defined as,

(i) A finite set of states

$$S = \{S_0, S_1, S_2, ...\}$$

(ii) A special element of set 'S' so called as initial state or start state

(iii) An input alphabet, $I = \{i_0, i_1, ...\}$

(iv) An output alphabet, $O = \{O_0, O_1, ...\}$

(v) A function $S \times I \rightarrow S$ (S.T.F.)

(vi) A function $S \times I \rightarrow O$ (M.A.F.).

At any state the machine is in any of it's states and on arrival of an input symbol it will change the state using state function and will also give the output using machine function. As the number of states is finite, it is known as finite state machine. The class of machines is known as finite automaton.

Q.5. Define finite automaton.

Ans. A finite automaton is mathematical model of a system with discrete inputs and outputs consist of finite set of states and set of transition from state to state.

A finite automaton is collection of 5-tuples

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

where, Q = finite set of states

Σ = finite set of input alphabets.

q_0 = initial state such that $q_0 \in Q$

F = finite set of final states such that $F \subseteq Q$

δ = transition function.

Q.6. Explain with use transition graph and transition matrix.

Or

Differentiate between transition graph and transition matrix.

(R.G.P.V., June 2015)

Or

Write short note on transition graph vs transition matrix.

(R.G.P.V., Dec. 2017)

Ans. **Transition Graph** – A directed graph, called a ***transition graph*** or transition diagram, is used to visualize and represent finite automata. It is just a simple graph with vertices representing states and edges representing

transitions. The vertices are represented by circles and the edges are represented by lines with arrows connecting the vertices. The labels on the vertices are the names of the states, while the labels on the edges are the current input symbol values. For example, if q_0 and q_1 are internal states of a DFA M, then its associated graph will have two vertices labelled q_0 and q_1. An edge (q_0, q_1) labelled a represents the transition $\delta(q_0, a) = q_1$. The initial state is represented by an incoming unlabelled arrow not originating at any vertex. One or more, final states are shown by double circles.

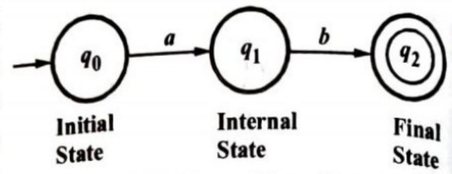


Fig. 1.2 Transition Graph

Formally, we can say, if $M = (Q, \Sigma, \delta, q_0, F)$ is a deterministic finite accepter, its associated transition graph G_M has exactly $|Q|$ vertices, each of which labelled with a different $q_i \in Q$. For each and every transition rule $\delta(q_i, a) = q_j$, the graph has an edge (q_i, q_j) labelled a. The vertex with label q_0 is called the ***initial vertex***, while the vertices labelled with $q_f \in F$ are the ***final vertices***. It is a trivial matter to convert from the $(Q, \Sigma, \delta, q_0, F)$ definition of a DFA to its corresponding transition graph and vice-versa. Fig. 1.2 represents the DFA,

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

where δ is given by

$$\delta(q_0, a) = q_1$$
$$\delta(q_1, b) = q_2$$

Transition Graph – A transition graph can be achieved by using on adjacency or a transition matrix. Such a matrix has its row's/columns labelled by the states $S_j \in S$. Along the intersection of the J-th row and K-th column in the array, we record the input symbol $\in I$ which causes the transition from a present state S_j to a new state S_k, this is followed by the punctuation '1' and the output symbol $\in O$. In case it is required to indicate the transition from a state S_j to a state S_k for for more than one symbol $\in I$, we use the symbol for logical or (V) and record this in addition, if there is no transition, we place the entry (–).

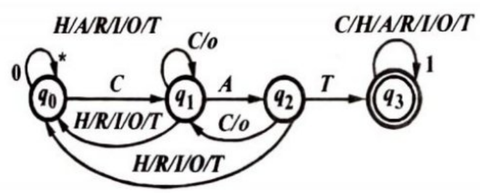


Fig. 1.3

State q_1 recognizes 'C' and q_2 recognize 'CA' thus when ever it reaches to q_3 machine recognize "CAT" thus machine remains in this state till the input gets over.

Table 1.1 Transition Matrix Table

Present State \ Next State	q_0	q_1	q_2	q_3
q_0	$H/_oV^A/_oV^R/_oV^I/_oV^O/_oV^T/_o$	C/o	–	–
q_0	$H/_oV^R/_oV^I/_oV^O/_oV^T/_o$	C/o	$A\backslash o$	–
q_2	$H/_oV^R/_oV^I/_oV^O/_oV^T/_o$	C/o	–	$T/1$
q_3	–	–	–	$C/1\ V^H/1\ V^A/1$ $V^R/1\ V^I/1\ V^O/1$ $V^T/1$

Where, H/O means, input symbol 'H' is associated with 'O' that is 'H' is at non final state. $T/1$ mean, input symbol 'T' is associated with a final state that is accepted state.

Q.7. How can you represent finite automata as a language acceptor and translator ?

Ans. Generally we say that computation is about taking input and producing output. Now we shall focus on a problem we can compute the answer to – decision problems, where output is restricted to 'Yes' or 'No'. Now we can further restrict the problems to those decision problems whose inputs are strings (like 'Nice'). These are kinds of questions that FSAs can be used to answer (but they can't answer all of them). The mathematical problems like 1 + 1 or 2 + 3 are not decision problems. We can make it decision problem as follows – Is string of the form '$x + y = z$', where x, y and z are decimal representations of integers X, Y and Z and $X + Y = Z$? This question and many like it cannot be answered using FSAs.

Stronger kinds of state machine exist; they are not finite. Examples include PDAs, LBAs and TMs. There are some decision problems like "does string X possess property Y ?" that not even Turing machines, the most powerful known kind of automata, cannot answer. One example is given by the halting problem – Given '$x(y)$' where x is a program and y is an input to program, does the program represented by X halt when passed the input y ? There is no Turing machine – that is, no algorithm to answer this question in the general case.

We can write an FSA that answers the question. "Is the string X a syntactically valid string. Naturally that depends on the rules of your language. Strings of the form "Number + Number + ... Number" can be recognized by an FSA, but an FSA can't tell us what the sum is ? In other words there is a difference between recognizing strings and computing results and FSAs typically are thought of as doing the former.

Q.8. What is a trap state in FA ? State and explain the properties of transition functions. **(R.G.P.V., Dec. 2015)**

Ans. The automata in fig. 1.4 remains in its initial state q_0 until the first b is encountered. If this is also the last symbol of the input, then the string is accepted. If not, the DFA goes into state q_2, from which it can never escape. Such a state is called a trap state.

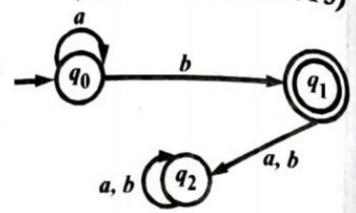


Fig. 1.4

The properties of transition functions are as follows –

(i) $\delta(q, \in) = q$ in a finite automaton. This means the state of the system can be changed only by an input symbol.

(ii) For all strings w and input symbols a,

$$\delta(q, aw) = \delta(\delta(q, a), w)$$

$$\delta(q, wa) = \delta(\delta(q, w), a)$$

This property gives the state after the automaton consumes or reads the first symbol of a string aw and the state after the automaton consumes a prefix of the string wa.

Q.9. How can we say that a string is acceptable by a finite automata or not ?

Ans. In an automata the final state 'F' is called an accepting state.

A string x is said to be accepted by a finite automat 'M', if $\delta(q_0, x) = q$, where $q \in F$. This is called acceptability of a string by final state.

Example – Consider a finite state machine M, whose transition function δ is given in table 1.2. Here $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1)$, F $\{q_0\}$. Give the entire sequence of states for the input string 110101. Also show that string is acceptable by automata or not.

Table 1.2 Transition Table

States	*Inputs*	
	0	*1*
$\rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Sol. We shall show that string is acceptable or not by finite automata by using properties of transition function. Starting from initial state, we get

$$\delta(q_0, 110101) = \delta(\delta(q_0, 1), 10101) \quad \text{(Property II (i))}$$

$$= \delta(q_1, 10101) \quad \text{(From transition table)}$$

$$= \delta(\delta(q_1, 1), 0101) \quad \text{(Property II (i))}$$

$$= \delta(q_0, 0101) \quad \text{(From transition table)}$$

$$= \delta(\delta(q_0, 0), 101 \quad \text{(Property II (i))}$$

$$= \delta(q_2, 101) \quad \text{(From transition table)}$$

$$= \delta(\delta(q_2, 1), 01) \quad \text{(Property II (i))}$$

$$= \delta(q_3, 01) \quad \text{(From transition table)}$$

$$= \delta(\delta(q_3, 0), 1) \quad \text{(Property II (i))}$$

$$= \delta(q_1, 1) \quad \text{(From transition table)}$$

$$= \delta(\delta(q_1, 1), \wedge) \quad \text{(Property II (i))}$$

$$= \delta(q_0, \wedge) \quad \text{(From transition table)}$$

$$= q_0 \quad \text{(Property I)}$$

The entire sequence of string is

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_0$$

Since, after processing the whole string by using transition function, we are reaching of an $q_0 \in F$ so we can say that string 110101 is acceptable by the finite automata M.

Q.10. Explain composite machine.

Ans. We can construct a composite machine by executing first one machine and then another. If M_1 and M_2 are two machines, with disjoint sets of non-interrupting states and transition functions δ_1 and δ_2, respectively, we write M_1M_2 to denote this composite machine. The set of states in composite machine is the union of the two sets. The composite machine M_1M_2 begins in the initial state of M_1 and executes the moves of M_1 upto the point at which M_1 interrupted; for any move that causes M_1 to halt, M_1M_2 executes the same move, except that it moves instead to the initial state of M_2. At this point, the position of reading head is at position of M_1 on which M_1 interrupted. From this point on, the moves of M_1M_2 are the moves of M_2 (using the transition function δ_2). If either M_1 or M_2 crashes during this process, then M_1M_2 crashes. M_1M_2 halts precisely if and when M_2 interrupted.

If we want to use this composite machine a larger context, in a manner similar to a transition diagram but without showing the states explicitly, we might also write $M_1 \rightarrow M_2$.

Q.11. Explain the finite automata with outputs. What are the differences between Moore machine and Mealy machine ?

Or

Formally define the following (with example) –

(i) Mealy machine (ii) Moore machine. ***(R.G.P.V., June 2010)***

Or

Explain Melay and Moore machines. ***(R.G.P.V., June 2017)***

Or

Explain the Mealy and Moore machines in brief. Also describe what are the differences between them. ***(R.G.P.V., Nov. 2019)***

Ans. The finite automata have binary output, i.e., they accept the string or do not accept the string. This acceptability is decided on the basis of reachability of the final state from the initial state. This restriction can be removed by considering the model where the output can be chosen from some other alphabets. There are two distinct approaches, first in which the output is associated with the current state, called a ***Moore machine*** and second in which the output is associated with the transition, called a ***Mealy machine***. In other words, the value of the output function $Z(t)$ in the general case, is a function of the current state $q(t)$ and the present input $a(t)$, i.e.,

$$Z(t) = \lambda\ (q(t),\ a(t))$$

where λ is called the output function. This generalized model is called ***Mealy machine***. If the output function $z(t)$ depends only on the present state and is independent of the current input, the output function may be written as –

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

This restricted model is called ***Moore machine***. It is more convenient to use Moore machine in automata theory.

Formally, we define each as follows –

(i) Moore Machine – The Moore machine is a six-tuple machine $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where

(a) Q is a finite set of states
(b) Σ is the input alphabet
(c) Δ is the output alphabet
(d) δ is the transition function which maps $\Sigma \times Q$ into Q
(e) λ is the output function mapping Q into Δ
(f) q_0 is the initial state.

The λ is a mapping from Q to Δ giving the output associated with each state. The output of M in response to input $a_1, a_2, \ldots, a_n$, $n \geq 0$ is $\lambda(q_0)\lambda(q_1), \ldots, \lambda(q_n)$, where $q_0, q_1, \ldots, q_n$ is the sequence of state such that $\delta(q_{i-1}, a_i) = q_i$ for $1 \leq i \leq n$. It is noted that machine gives output $\lambda(q_0)$

in response to input $\in$. The DFA may be viewed as a special case of Moore machine where the output alphabet is $\{0,1\}$ and state q is "accepting" if and only if $\lambda(q) = 1$.

For example, table 1.3 shows a Moore machine. The initial state q_0 is marked with an arrow. The table defines δ and λ. For the input string 0111, the transition of states is given by

$$q_0 \rightarrow q_3 \rightarrow q_0 \rightarrow q_1 \rightarrow q_2$$

The output string is 00010. For the input string $\in$, the output is $\lambda(q_0) = 0$.

Table 1.3 A Moore Machine

Present State	*Next State δ*		*Output λ*
	a = 0	*a = 1*	
$\rightarrow q_0$	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

(ii) Mealy Machine – The Mealy machine is also a six-tuple machine $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where all the symbols except λ have the same meaning as in the Moore machine. λ is the output function mapping $\Sigma \times Q$ into Δ. i.e., $\lambda(q, a)$ gives the output associated with the transition from state q on into a. The output of M in response to input $a_1, a_2 \ldots\ldots a_n$ is $\lambda(q_0, a_1)\lambda(q_1, a_2), \ldots\ldots\lambda(q_{n-1}, a_n)$, where $q_0, q_1, \ldots, q_n$ is the sequence of states such that $\delta(q_{i-1}, a_i) = q_i$ for $1 \leq i \leq n$. It is noted that this sequence has length n rather than length $n + 1$ as for the Moore machine, and on input $\in$ a Mealy machine gives output $\in$.

For example, table 1.4 shows a Mealy machine which defines δ and λ. For the input string 0111, the transition of states is given by

$$q_0 \rightarrow q_3 \rightarrow q_0 \rightarrow q_1 \rightarrow q_2$$

and the output string is 0010. In the case of Mealy machine, we get an output only on the application of an input symbol. So for input string $\in$, the output is only $\in$, while in the Moore machine the output is $\lambda(q_0)$.

Table 1.4 A Mealy Machine

Present State	*Next State*			
	a = 0		*a = 1*	
	State	*Output*	*State*	*Output*
$\rightarrow q_0$	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

Q.12. What is Moore machine ? How finite automata can be converted into Moore machine ? Explain with the help of example. ***(R.G.P.V., June 2015)***

Sol. **Moore Machine** – Refer to Q.11 (i).

A finite automata can be converted into a Moore machine by introducing output alphabet $\Delta = \{0, 1\}$ and defining the output function $\lambda(q)$, such that

$$\lambda(q) = \begin{cases} 1 & \text{if} \quad q \in F \\ 0 & \text{if} \quad q \notin F \end{cases}$$

For example, consider the automata given by table 1.5.

Let $\Delta = \{0, 1\}$ be the introduced alphabet. The value of output is defined by output function $\lambda(q)$, we have

$$\lambda(q_0) = 1,\ \lambda(q_1) = 0,\ \lambda(q_2) = 0,\ \lambda(q_3) = 0,$$

Table 1.5 Finite Automata

Present State	*Inputs*	
	a = 0	*x = 1*
$\rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Table 1.6 Moore Machine

Present State	*Next State*		*Output*
	a = 0	*x = 1*	
$\rightarrow q_0$	q_2	q_1	1
q_1	q_3	q_0	0
q_2	q_0	q_3	0
q_3	q_1	q_2	0

The Moore machine is constructed by writing output values calculated corresponding to states of FA. The Moore machine is shown in table 1.6.

Q.13. Explain the procedure for transforming a Mealy machine into a Moore machine with the help of an example.

Ans. Consider the Mealy machine described by the transition table given in table. 1.7.

Table 1.7 An Example of Mealy Machine

Present State	*Next State*			
	Input a = 0		*Input a = 1*	
	State	*Output*	*State*	*Output*
$\rightarrow q_1$	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

The construction of the Moore machine which is equivalent to the Mealy machine is as follows –

(i) Starting with initial state, determine the no. of output associated with that state in next state column. Do this for all other states too.

For example – In table 1.7,

(a) q_1 is associated with output 1 only

(b) q_2 is associated with outputs 0 and 1

(c) q_3 is associated with output 0 only

(d) q_4 is associated with outputs 0 and 1

(ii) Split the state q_i into several different states, the number of such states is the number of different outputs associated with q_i.

For example – Since q_2 and q_4 have more than one output. So split q_2 and q_4 into q_{20}, q_{21} and q_{40}, q_{41} respectively.

The reconstructed version of table 1.7 is shown in table 1.8.

Table 1.8 Intermediate State Table

Present State	*Next State*			
	Input a = 0		*Input a = 1*	
	State	*Output*	*State*	*Output*
$\rightarrow q_1$	q_3	0	q_{20}	0
q_{20}	q_1	1	q_{40}	0
q_{21}	q_1	1	q_{40}	0
q_3	q_{21}	1	q_1	1
q_{40}	q_{41}	1	q_3	0
q_{41}	q_{41}	1	q_3	0

(iii) Now, rearrange the intermediate state table such that association of output with every state in next state column is transferred to same state in present state column.

For example, the rearrangement of table 1.8 is shown in table 1.9.

(iv) Now, check whether the initial state is associated with lowest or initial output or not.

(a) Yes, the rearranged table is transition table or required Moore machine.

(b) No, this means that with input $\in$, we get the output that is not minimum. The machine accepts null sequence. To overcome this, either neglect the response of Moore machine to the input $\in$, or add another state q_0 whose transitions are identical with initial state.

For example – In table 1.9 q_1 is initial state having output 1. So, we have to add another state q_0 whose transitions are identical to state q_1 but have the output 0 as shown in table 1.10.

Table 1.9 Rearranged State Table

Present State	Next State		Output
	$a = 0$	$a = 1$	
$\rightarrow q_1$	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Table 1.10 An Equivalent Moore Machine

Present State	Next State		Output
	$a = 0$	$a = 1$	
$\rightarrow q_0$	q_3	q_{20}	0
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Table 1.10 is required Moore machine equivalent to the Mealy machine of table 1.7.

It is cleared from the procedure, if we have m-output, n-state Mealy machine, the corresponding m-output Moore machine has no more than $mn + 1$ states.

Q.14. Explain the procedure for transforming a Moore machine into a Mealy machine with the help of an example.

Ans. The following procedure is adopted to construct an equivalent Mealy machine from a given Moore machine.

(i) We have to define the output function λ' for Mealy machine as a function of present state and input symbol. λ' is defined as –

$$\lambda'(q,a) = \lambda(\delta(q,a)) \text{ for all states } q \text{ and input symbol } a.$$

(ii) The transition function is the same as that of the given Moore machine.

Table 1.11 An Example of Moore Machine

Present State	Next State		Output
	$a = 0$	$a = 1$	
$\rightarrow q_0$	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

Table 1.12 An Equivalent Mealy Machine

Present State	Next State			
	Input $a = 0$		Input $a = 1$	
	State	Output	State	Output
$\rightarrow q_0$	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

Consider the Moore machine described by the transition table given in the table 1.11.

We must follow the reverse procedure of converting Mealy machine into Moore machine. In case of the Moore machine, for every input symbol we form the pair consisting of the next state and the corresponding output and reconstruct the table for Mealy machine.

For example, in table 1.11 the states q_3 and q_1 in the next state column should be associated with outputs 0 and 1, respectively. The transition table for Mealy machine is given in table 1.12.

It is noted that the number of states can be reduced in any model by considering states with identical transition. If two states have identical transitions (i.e., the rows corresponding to the two-states are identical), then we can remove one of them.

NUMERICAL PROBLEMS

Prob.1. Design a FA which accepts set of strings containing four 1's in every string over alphabet $\Sigma = \{0, 1\}$. ***(R.G.P.V., Dec. 2008)***

Sol. The finite automation M is given by

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{0, 1\}, \delta, q_0, \{q_4\})$$

The finite automation is shown in fig. 1.5.

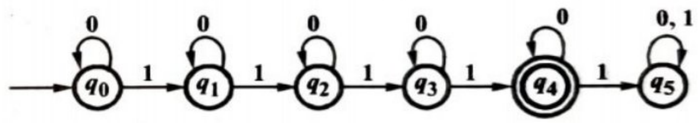


Fig. 1.5 Finite Automation

Prob.2. Design DFA that accepts all strings with at most 3a's. ***(R.G.P.V., June 2016, Dec. 2017)***

Sol. The finite automation M is given by

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, (a, b), \delta, q_0, \{q_0, q_1, q_2, q_3\})$$

The finite automation is shown in fig. 1.6.

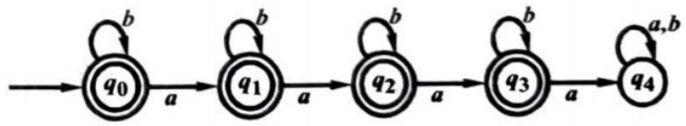


Fig. 1.6 DFA

Prob.3. Give DFA accepting the language over alphabet {0, 1} such that all strings of 0 and 1 ending in 101. ***(R.G.P.V., Dec. 2006)***

Sol. The expression (0 + 1)*101 denotes all strings of 0's and 1's ending in 101. Now we construct a DFA accepting the regular expression (0, 1)*101 corresponding to given language. The set of input alphabet is {0, 1}.

Fig. 1.7 shows a transition diagram for the FA.

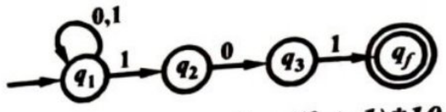


Fig. 1.7 FA Accepting (0 + 1)*101

Now, we constructed a transition table for fig. 1.7.

Table 1.13 Transition Table for FA

State	*0*	*1*
q_1	q_1	$q_1 q_2$
q_2	q_3	–
q_3	–	q_f
q_f	–	–

Now, we convert above NFA into DFA as –

Table 1.14 Transition Table for DFA

State	*0*	*1*
$[q_1]$	$[q_1]$	$[q_1\ q_2]$
$[q_1\ q_2]$	$[q_1\ q_3]$	$[q_1\ q_2]$
$[q_1\ q_3]$	$[q_1]$	$[q_1\ q_2\ q_f]$
$[q_1\ q_2\ q_f]$	$[q_1\ q_3]$	$[q_1\ q_2]$

State diagram for required DFA which accepted strings ending in 101 shown in fig. 1.8.

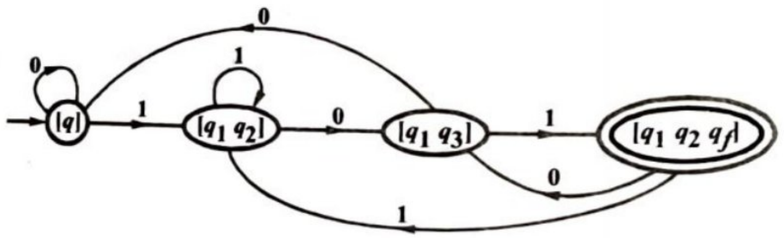


Fig. 1.8 DFA Accepting Strings Ending in 101

Prob.4. Design deterministic finite automation accepting the following languages over the alphabet {0, 1} –

(i) The set of all words ending in 00.

(ii) The set of all words except ∈.

(iii) The set of all words that begin with 0.

(R.G.P.V., Dec. 2015)

Sol. (i) The expression (0 + 1)* 00 denotes all strings of 0's and 1's ending in 00. Fig. 1.9 shows a transition diagram for the DFA.

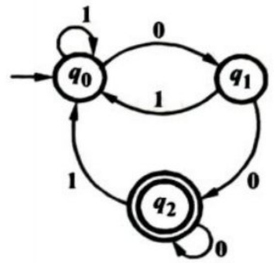


Fig. 1.9

(ii) The DFA accepting the set of all words except ∈ over the alphabet {0, 1} is shown in fig. 1.10.

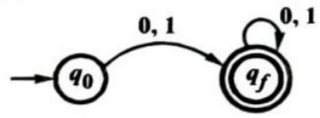


Fig. 1.10

(iii) The DFA accepting the set of all words that begin with 0 over the alphabet {0,1} is shown in fig. 1.11.

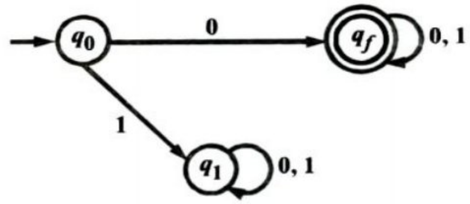


Fig. 1.11

Prob.5. Design DFA accepting the language over the alphabet 0, 1 that have the set of all strings ending in 00. ***(R.G.P.V., Dec. 2016)***

Sol. Refer sol. of Prob.4 (i).

Prob.6. Draw a deterministic finite automata for alphabet {a, b}, for the language {w/w contains the substring abab}. ***(R.G.P.V., Dec. 2007)***

Sol. Given language is,

L = {w/w contains the substring $abab$}

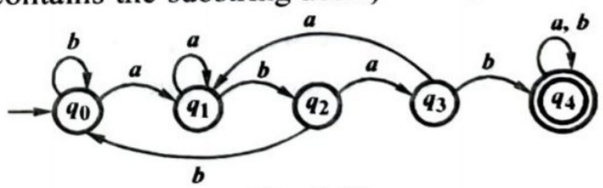


Fig. 1.12

Fig. 1.12 gives a transition diagram for this DFA that is able to recognize string in L.

Prob.7. Construct a finite automata for the language
$\{0^n \mid n \bmod 3 = 2, n \geq 0\}$.

(R.G.P.V., Dec. 2014)

Sol. No. of strings generated by language are –
00, 00000, 00000000, 00000000000, ...

So, finite automata for the given language is shown in fig. 1.13.

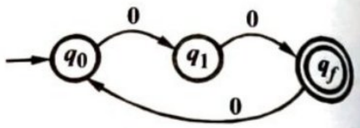


Fig. 1.13

Prob.8. Construct DFA for –
(i) Binary integer divisible by 3
(ii) $a^n b \mid n \geq 0$.

(R.G.P.V., Dec. 2009)

Sol. (i) In binary number when we want to find the corresponding value receiving a new bit then remember the original value is doubled and new bit is added.

Table 1.15

Q \ Σ		0	1
Start	$\rightarrow S$	q_0	q_1
(rem '0')	q_0	q_0	q_1
(rem '1')	q_1	q_2	q_0
(rem '2')	q_2	q_1	q_2

In starting state if we receive '0', then we go to state where remainder is '0', if we get '1' then the next state is remainder '1'.

Formula – 2 × Old remainder + Next bit

In state q_0 – 2 × 0 (rem '0') + 0 (next bit)
= 0 + 0
= 0 (remainder '0', i.e. state q_0)
2 × 0 (rem '0') + 1 (next bit)
= 0 + 1 = 1

In state q_1 – 2 × 1 (rem '1') + 0 (next bit)
= 2 + 0 = 2
2 × 1 (rem '1') + 1(next bit)
= 2 + 1 = 3

3 is divisible by '3' and remainder is '0' i.e., state 'q_0'

In state q_2 – 2 × 2 (old remainder) + 0 (next bit)
= 4 + 0
= 4 (remainder '1' i.e., q_1)
2 × 2 (old remainder '2') + 1(next bit)
= 4 + 1
= 5 (remainder '2' i.e., state q_2)

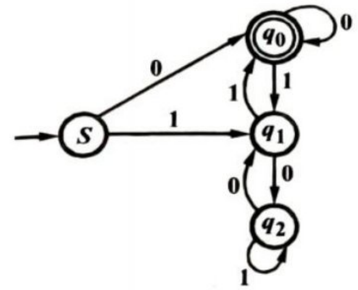


Fig. 1.14

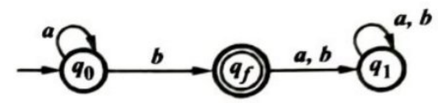


Fig. 1.15

(ii) No. of strings generated by language $a^n b | n \geq 0$ are– *b*, *ab*, *aab*, *aaab*, *aaaab*, ...

So, DFA for the given language is shown in fig. 1.15.

Prob.9. Construct a finite automaton that will accept those strings of a binary number that are divisible by three. ***(R.G.P.V., Dec. 2010)***

Sol. Refer sol. of Prob.8 (i).

Prob.10. Design a machine which checks whether a given decimal number is "EVEN".

Sol. There will be two states one for EVEN and second one for odd. Symbols, 0, 2, 4, 6, 8 indicate that they are even numbers because when we divide these numbers by '2' we get remainder '0' otherwise odd number.

Table 1.16 State Table

	State \ Input	0, 2, 4, 6, 8	1, 3, 5, 7, 9
Starting state	$\rightarrow q_0$	q_0	q_1
odd	q_1	q_2	q_1
even	q_2	q_2	q_1

Note – '→' This symbol shows that given state is starting state i.e., $\rightarrow q_0$ and $\rightarrow q_0$

'◎' Final state which is accepting state.

Therefore, $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
$Q = \{q_0, q_1, q_2\}$
δ = The transition of states
q_0 = Start state
$F = \{q_2\}$

M.A.F. →

Table 1.17

Q \ Σ	0, 2, 4, 6, 8	1, 3, 5, 7, 9
q_0	1	0
q_1	1	0
q_2	0	0

1 → even

0 → 0

Transition Diagram –

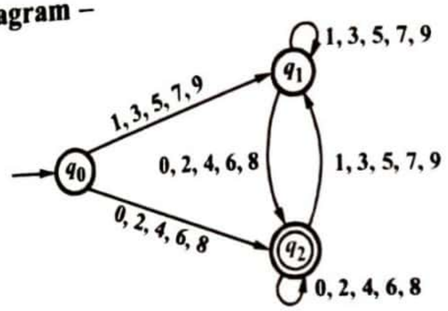


Fig. 1.16

Suppose the number is – 246710321

$\underline{q_0}$ 246710321

2 $\underline{q_2}$ 46710321

24 $\underline{q_2}$ 6710321

246 $\underline{q_2}$ 710321

2467 $\underline{q_1}$ 10321

24671 $\underline{q_1}$ 0321

246710 $\underline{q_2}$ 321

2467103 $\underline{q_1}$ 21

24671032 $\underline{q_2}$ 1

246710 321 $\boxed{\underline{q_1}}$

As we end up in the state q_1 which is a non final state therefore given number is odd.

Prob.11. Design a machine which can check that given string is divisible by '4'.

Sol. To check divisibility by '4' there will be four states with remainder '0', remainder '1', remainder '2' and remainder '3' and one more state for starting state.

Table 1.18

Q \ Σ	0, 4, 8	1, 5, 9	2, 6	3, 7
S	q_0	q_1	q_2	q_3
Rem '0' q_0	q_0	q_1	q_2	q_3
Rem '1' q_1	q_2	q_3	q_0	q_1
Rem '2' q_2	q_0	q_1	q_2	q_3
Rem '3' q_3	q_2	q_3	q_0	q_1

Note – We can merge two states by one, if both states have same transition and both of them are final or non final states.

In table 1.18 row-q_1 and row-q_3 both are identical and both of them are non final states. So we are replacing state q_3 by q_1

Table 1.19

Q \ Σ	0, 4, 8	1, 5, 9	2, 6	3, 7
Start	q_0	q_{13}	q_2	q_{13}
q_0	q_0	q_{13}	q_2	q_{13}
q_{13}	q_2	q_{13}	q_0	q_{13}
q_2	q_0	q_{13}	q_2	q_{13}

we can observe that two columns in table 1.19 are same i.e., for 1, 5, 9 and 3, 7. Hence they can be grouped into one.

Table 1.20

Q \ Σ	0, 4, 8	1, 3, 5, 7, 9	2, 6
Start (S)	q_0	q_{13}	q_2
q_0	q_0	q_{13}	q_2
q_{13}	q_2	q_{13}	q_0
q_2	q_0	q_{13}	q_2

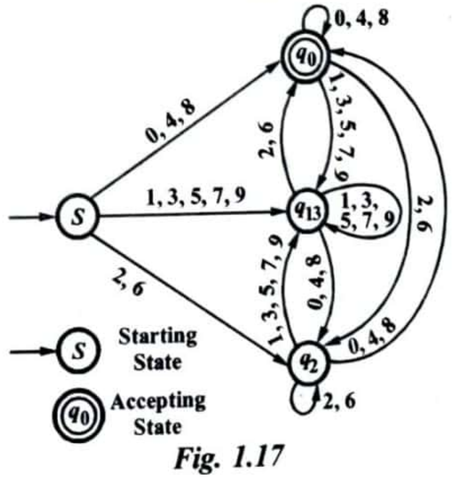


Fig. 1.17

Prob.12. Design FA which accepts even no. of 0's and even no. of 1's. ***(R.G.P.V., Dec. 2011)***

Or

Construct DFA over input alphabet $\Sigma = \{0, 1\}$ to accept string which contains no. of 0 is even and no. of 1 is even. ***(R.G.P.V., June 2010)***

Or

Construct a DFA accepting set of all strings containing even no. of a's and even no. of b's over alphabet {a, b}. ***(R.G.P.V., June 2008)***

Sol. Consider the transition diagram of fig. 1.18. This FA is denoted by $M = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $F = \{q_0\}$ and δ is shown in fig. 1.19.

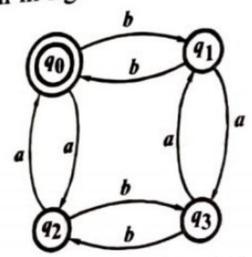


Fig. 1.18 The Transition Diagram of a Finite Automaton

States	Inputs a	Inputs b
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Fig. 1.19 δ for the FA of Fig. 1.18

Suppose *bbabab* is input to *M*. Since $\delta(q_0, b) = q_1$ and $\delta(q_1, b) = q_0$, thus $\delta(q_0, bb) = \delta(\delta(q_0, b), b) = \delta(q_1, b) = q_0$

Therefore, *bb* is in *L(M)*, but we are interested in *bb abab*. We continue by $\delta(q_0, a) = q_2$. Thus,

$$\delta(q_0, bba) = \delta(\delta(q_0, bb), a) = \delta(q_0, a) = q_2$$

Continuing in this fashion, we find that

$$\delta(q_0, bbab) = q_3, \ \delta(q_0, bbaba) = q_1$$

and finally $\delta(q_0, bbabab) = q_0$

The entire state of sequence is

$$q_0 \overset{b}{} q_1 \overset{b}{} q_0 \overset{a}{} q_2 \overset{b}{} q_3 \overset{a}{} q_1 \overset{b}{} q_0$$

Thus *bbabab* is in *L(M)*.

Prob.13. Construct a finite automaton accepting all string over {0, 1}

(i) Having odd number of 0's

(ii) Having even number of 0's and even number of 1's.

(R.G.P.V., June 2017)

Sol. (i) Having Odd Number of 0's -- The finite automaton *M* is given by

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

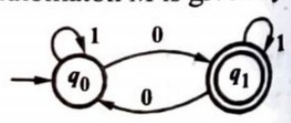


Fig. 1.20

(ii) Having Even Number of 0's and Even Number of 1's – Refer sol. of Prob.12.

Prob.14. Construct DFA for the following set –

(i) All strings of {0, 1} such that third symbol from right hand side is "1".

(ii) All strings of {a, b} such that 'aa' is not the sub-string.

(iii) All strings of {a, b} where number of a's are multiple of three and number of b's are multiple of two.

(R.G.P.V., June 2007)

Sol. (i) The required regular expression is –

$$(0 + 1)^* \ 1 \ (0 + 1) \ (0 + 1).$$

Now, we construct the transition system and get the NFA accepting the set of string. The transition system is shown in fig. 1.21.

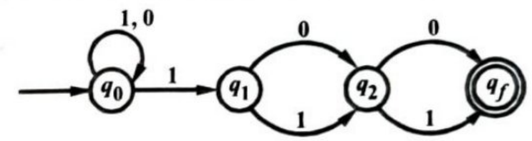


Fig. 1.21 NFA for (0 + 1)* 1 (0 + 1) (0 + 1)

The equivalent DFA is given in table 1.21.

Table 1.21 DFA Equivalent to (0 + 1)* 1 (0 + 1) (0 + 1)

State	*0*	*1*
$[q_0]$	$[q_0]$	$[q_0q_1]$
$[q_0q_1]$	$[q_0q_2]$	$[q_0q_1q_2]$
$[q_0q_2]$	$[q_0q_f]$	$[q_0q_1q_f]$
$[q_0q_f]$	$[q_0]$	$[q_0q_1]$
$[q_0q_1q_2]$	$[q_0q_2q_f]$	$[q_0q_1q_2q_f]$
$[q_0q_1q_f]$	$[q_0q_2]$	$[q_0q_1q_2]$
$[q_0q_2q_f]$	$[q_0q_f]$	$[q_0q_1q_f]$
$[q_0q_1q_2q_f]$	$[q_0q_2q_f]$	$[q_0q_1q_2q_f]$

(ii) First creates a DFA for the set of all strings of $\{a, b\}$ which contain '*aa*' as substring. The DFA which accept strings that contains '*aa*' as substring is shown in fig. 1.22.

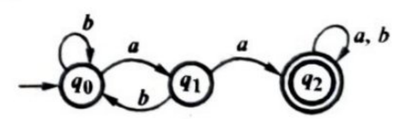


Fig. 1.22 DFA having "aa" as Substring over {a, b}

Now, to construct the DFA that does not accept strings having '*aa*' as substring, make all non-final states to final and final states to non-final states.

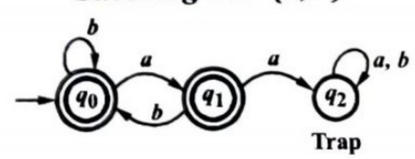


Fig. 1.23 DFA Accepts Strings which does not contains 'aa' as Substring

The resultant DFA which accepts all strings of $\{a, b\}$ such that '*aa*' is not the substring, is shown in fig. 1.23.

(iii) The required DFA is shown in fig. 1.24.

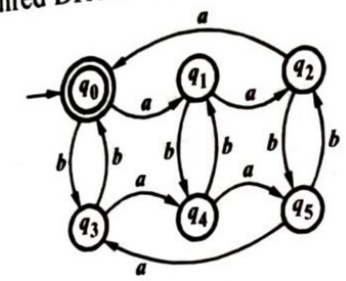


Fig. 1.24

Prob.15. Construct an DFA accepting the set of all string over the alphabet {0, 1}, such that number of 0's divisible by 5 and number of 1's divisible by 3. **(R.G.P.V., Dec. 2012)**

Or

Design a DFA that accepts the string such that number of zero divisible by five and number of one divisible by three. **(R.G.P.V., Dec. 2013)**

Sol. Consider the transition diagram of fig. 1.25. This FA is denoted by $M = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}, q_{13}, q_{14}\}$ and $F = \{q_0\}$, $\Sigma = \{0, 1\}$ and δ is shown in table 1.22.

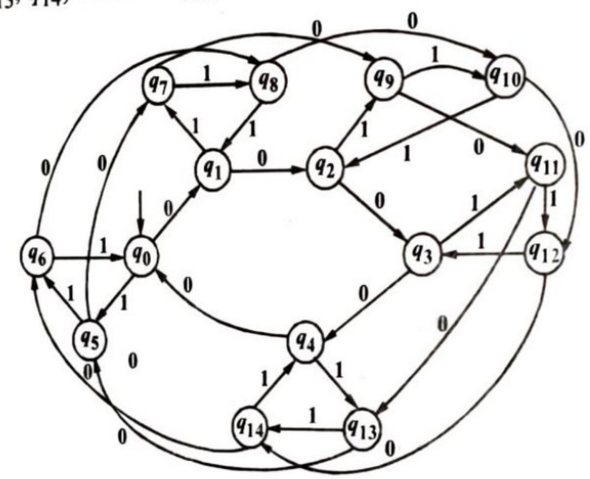


Fig. 1.25

Table 1.22 Transition Table

Q \ Σ	0	1
First Zero q_0	q_1	q_5 First One
Second Zero q_1	q_2	q_7 First One
Third Zero q_2	q_3	q_9 First One
Fourth Zero q_3	q_4	q_{11} First One
Fifth Zero q_4	q_0	q_{13} First One
Second One q_5	q_7	q_6
Third One q_6	q_8	q_0
Second One q_7	q_9	q_8
Third One q_8	q_{10}	q_1
Second One q_9	q_{11}	q_{10}
Third One q_{10}	q_{12}	q_2
Second One q_{11}	q_{13}	q_{12}
Third One q_{12}	q_{14}	q_3
Second One q_{13}	q_5	q_{14}
Third One q_{14}	q_6	q_4

Prob.16. Construct a NDFA accepting all strings in {a, b} with either two consecutive a's or two consecutive b's. **(R.G.P.V., May 2018)**

Sol. Consider the transition diagram of fig. 1.26. The NFA is denoted by $M = \{Q, \Sigma, \delta, q_0, F\}$, where $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $F = \{q_0\}$ and δ is shown in table 1.23.

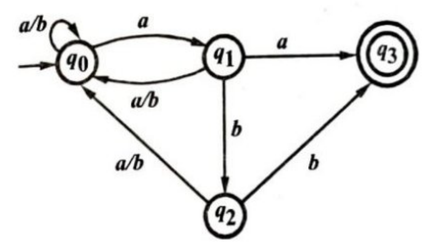


Fig. 1.26 Transition Diagram of NFA

Table 1.23 Transition Table of NFA

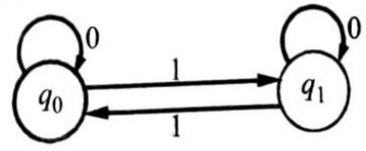
States	Input a	Input b
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_0, q_2\}$	$\{q_0, q_2\}$
q_2	$\{q_0\}$	$\{q_0, q_3\}$
q_3	–	–

Prob.17. Construct a Mealy machine which can output EVEN, ODD according as the total number of 1's encountered is even or odd. The input symbols are 0 or 1. **(R.G.P.V., Dec. 2010)**

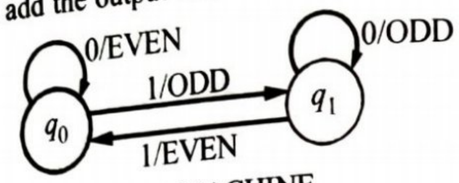
Sol. We require two states q_0 and q_1. q_0 denotes even number of ones and q_1 denotes odd number of ones. First we construct F.S.M.

	Q \ Σ	0	1
even no. of '1'	q_0	q_0	q_1
odd no. of '1'	q_1	q_1	q_0

Transition diagram for F.S.M.



Now, we will add the output with each input-symbol.



MEALY MACHINE

We can check that given string containing even numbers of ones or odd numbers of ones. Suppose input string is → 01110

$\underline{q_0}\,1\,1\,1\,0 \Rightarrow$ O/P → EVEN

$0\,\underline{q_0}\,1\,1\,1\,0 \Rightarrow$ O/P → ODD

$0\,1\,\underline{q_1}\,1\,1\,0 \Rightarrow$ O/P → EVEN

$0\,1\,1\,\underline{q_0}\,1\,0 \Rightarrow$ O/P → ODD

$0\,1\,1\,1\,\underline{q_0}\,0 \Rightarrow$ O/P → ODD

$0\,1\,1\,1\,0\,\boxed{\underline{q_1}}$

q_1 shows that given string containing odd number of ones.

Prob.18. Define Mealy and Moore machines. Construct a Mealy machine which can output EVEN, ODD according as the total number of 1's encountered is even or odd. The input symbols are 0 and 1.

(R.G.P.V., Nov. 2018)

Sol. **Mealy and Moore Machines** – Refer to Q.11.

Construction of Mealy Machine – Refer to Prob.17.

Prob.19. What do you mean by automata with output capability ? Draw a Mealy machine equivalent to the following circuit.

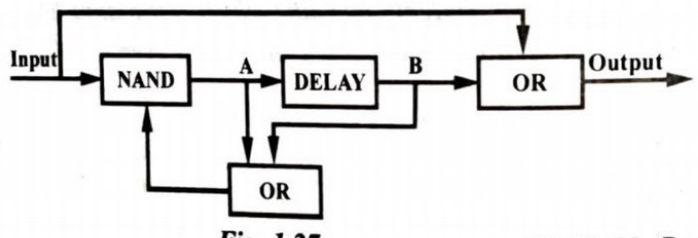


Fig. 1.27

(R.G.P.V., Dec. 2015)

Sol. Refer to Q.11.

We denote current in wire by 1 and no current by 0. We identify four states based on whether current is available or not.

The operation of this circuit is such that after an input of 0's and 1's, the states changes according to the following rules –

Q	Input	
	A	B
q_0	0	0
q_1	0	1
q_2	1	0
q_3	1	1

New B = Old A (∵ D Flip flop)

New A = (Input) NAND (old A or old B)

Output = (Input) OR (old B)

Suppose we are in state q_0 and we are not receiving current i.e., input = 0

New B = Old A = 0 (Intable, A = 0, in state q_0)

$\boxed{\text{New } B = 0}$

New A = (Input) NAND (old A OR old B)

New A = 0 NAND (0 OR 0)

New A = 0 NAND 0

$\boxed{\text{New } A = 1}$

Output = (Input) OR (old B)

Output = 0 OR 0

$\boxed{\text{Output} = 0}$

The next state is q_2 because, new A = 1 and new B = 0.

If we are in state q_0 and we receive current input = 1.

$\boxed{\text{New } B = \text{Old } A = 0}$ (from table, A = 0)

New A = (Input) NAND (old A OR old B)

New A = 1 NAND (0 OR 0)

New A = 1 NAND 0

$\boxed{\text{New } A = 1}$

Output = 1 OR 0

$\boxed{\text{Output} = 1}$

The next state is q_2 (because new A = 1, new B = 0)

If we are in state q_1

Let, Input = 0 (No current)

$\boxed{\text{New } B = \text{Old } A = 0}$ (from table)

New A = (Input) NAND (old A OR old B)

New A = 0 NAND 1

$\boxed{\text{New } A = 1}$

Output = 0 OR 1

$\boxed{\text{Output} = 1}$

therefore next state is q_2 (∵ A = 1, B = 0)

when Input = 1 (with current)

$\boxed{\text{New } B = \text{Old } A = 0}$

New $A = 1$ NAND (0 OR 1)
New $A = 1$ NAND 1

$\boxed{\text{New } A = 0}$

Output = 1 OR 1

$\boxed{\text{Output} = 1}$

therefore next state is q_0 ($\because A = 0, B = 0$)

If we are in state q_2

when input = 0 (No current) (from table)

$\boxed{\text{New } B = \text{Old } A = 1}$

New $A = 0$ NAND (1 OR 0)
New $A = 0$ NAND 1

$\boxed{\text{New } A = 1}$

Output = 0 OR 0

$\boxed{\text{Output} = 0}$

therefore next state is q_3 ($\because A = 1, B = 1$)

when input = 1 (with current)

$\boxed{\text{New } B = \text{Old } A = 1}$

New $A = 1$ NAND (1 OR 0)
New $A = 1$ NAND 1

$\boxed{\text{New } A = 0}$

Output = 1 OR 0

$\boxed{\text{Output} = 1}$

therefore next state is q_1 ($\because A = 0, B = 1$)

If we are in state q_3

when input = 0 (No current)

$\boxed{\text{New } B = \text{Old } A = 1}$

New $A = 0$ NAND (1 OR 1)
New $A = 0$ NAND 1

$\boxed{\text{New } A = 1}$

Output = 0 OR 1

$\boxed{\text{Output} = 1}$

therefore next state is q_3 ($\because A = 1, B = 1$)

when Input = 1

$\boxed{\text{New } B = \text{Old } A = 1}$

New $A = 1$ NAND (1 OR 1)
New $A = 1$ NAND 1

$\boxed{\text{New } A = 0}$

Output = 1 OR 1

$\boxed{\text{Output} = 1}$

therefore next state is q_1 ($\because A = 0, B = 1$)

Table 1.24 Transition Table

Old State	*Input = 0*		*Input = 1*	
	New State	*Output*	*New State*	*Output*
q_0	q_2	0	q_2	1
q_1	q_2	1	q_0	1
q_2	q_3	0	q_1	1
q_3	q_3	1	q_1	1

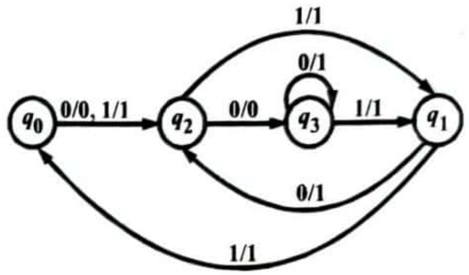


Fig. 1.28 Transition Diagram

Prob.20. Construct a Moore machine which calculates mod-4 for each binary string treated as binary integer. ***(R.G.P.V., Dec. 2008)***

Sol. Since last two digits of every binary integer decides whether integer is divisible by 4 or not. So that it can also decide what will be the remainder if the integer is divisible by 4 as

Last Two Digit of Binary Integer	*Remainder*
00	0
01	1
10	2
11	3

So, Moore machine which calculate Mod-4 for each binary string treated as binary integer is shown in fig. 1.29.

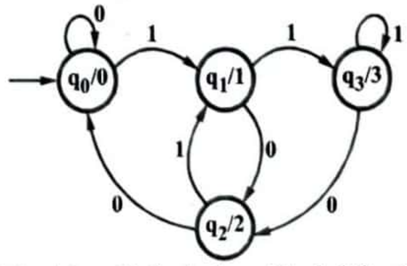


Fig. 1.29 A Moore Machine Calculating Mod-4 for Each Binary Integer

Prob.21. Construct the Mealy machine equivalent to the given Moore machine –

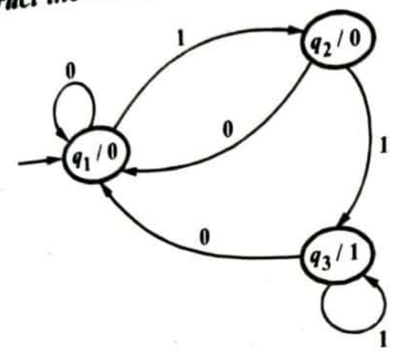


Fig. 1.30 Moore Machine

Sol. First, we convert the transition diagram into the transition table as shown in table 1.25.

Table 1.25 Transition Table for Moore Machine

Present State	*Next State*		*Output*
	a = 0	*a = 1*	
$\rightarrow q_1$	q_1	q_2	0
q_2	q_1	q_3	0
q_3	q_1	q_3	1

Now, we construct the transition table as shown in table 1.26 by associating the output with the transitions. The procedure is described in Q.14.

Table 1.26 Intermediate Transition Table

Present State	*Next State*			
	a = 0		*a = 1*	
	State	*Output*	*State*	*Output*
$\rightarrow q_1$	q_1	0	q_2	0
q_2	q_1	0	q_3	1
q_3	q_1	0	q_3	1

In table 1.26, the rows corresponding to q_2 and q_3 are identical. So, we can delete one of the two states, i.e., q_2 or q_3. We reconstruct the table by deleting q_3 as shown in table 1.27.

Table 1.27 Mealy Machine Equivalent to the Given Moore Machine

Present State	*Next State*			
	a = 0		*a = 1*	
	State	*Output*	*State*	*Output*
$\rightarrow q_1$	q_1	0	q_2	0
q_2	q_1	0	q_2	1

Since, we have deleted q_3-row from table 1.26, so we replaced q_3 by q_2 in table 1.27 under the input column $a = 1$. The corresponding transition diagram is shown in fig. 1.31.

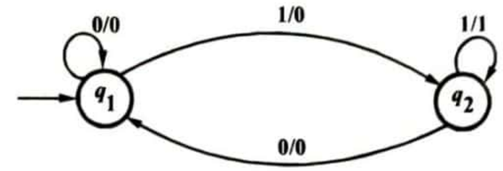


Fig. 1.31 Mealy Machine

Prob.22. Construct a Mealy machine which is equivalent to the Moore machine given below –

Table 1.28

Present State	*Next State*		*Output*
	a = 0	*a = 1*	
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

(R.G.P.V., Dec. 2009)

Sol. As given above table, suppose we are at state q_0 and q_0 can reached state q_1 or q_2 depends upon input symbol value i.e., $a = 0$, $a = 1$. When we see that q_1 has output 0 and q_2 has output '1', therefore q_0 to q_1, edge value will be input/output, 0/0 and from q_0 to q_2 the edge value will be 1/1. We can find remaining values by this rule.

Table 1.29

Present State	*Next State*			
	a = 0		*a = 1*	
	Next State	*Output*	*Next State*	*Output*
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 –

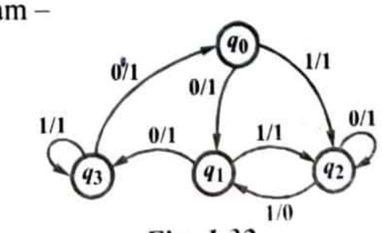


Fig. 1.32

Prob.23. Translate the following Mealy machine into Moore machine. ***(R.G.P.V., June 2007)***

Sol. Let us convert the transition diagram into the transition table. For the given problem – q_1 is associated with 1 and q_2 is associated with two different outputs 0 and 1. Similarly, q_3 is associated with 0 and 1, whereas q_4 is associated with 1. The state table with new states q_1, q_{20}, q_{21}, q_{30}, q_{31} and q_4 is reconstructed as shown in table 1.31.

Table 1.30 Transition Table for Mealy Machine

Present State	Next State			
	a = 0		a = 1	
→ q_1	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

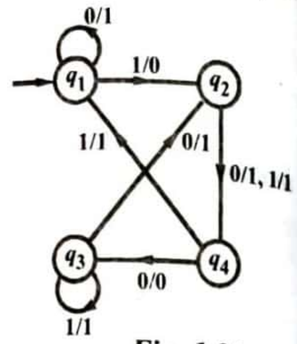


Fig. 1.33

Table 1.31 Intermediate State Table

Present State	Next State			
	a = 0		a = 1	
q_1	q_1	1	q_{20}	0
q_{20}	q_4	1	q_4	1
q_{21}	q_4	1	q_4	1
q_{30}	q_{21}	1	q_{31}	1
q_{31}	q_{21}	1	q_{31}	1
q_4	q_{30}	0	q_1	1

Table 1.32 Rearranged State Table

Present State	Next State		Output
	a = 0	a = 1	
q_1	q_1	q_{20}	1
q_{20}	q_4	q_4	0
q_{21}	q_4	q_4	1
q_{30}	q_{21}	q_{31}	0
q_{31}	q_{21}	q_{31}	1
q_4	q_{30}	q_1	1

The pair of states and outputs in the next state column can be rearranged as given in table 1.32.

Prob.24. Convert the following Mealy machine into its equivalent Moore machine –

Present State	Next State			
	a = 0		a = 1	
	State	Output	State	Output
q_0	q_1	n	q_2	n
q_1	q_1	y	q_2	n
q_2	q_1	n	q_2	y

(R.G.P.V., Dec. 2012)

Sol. We construct the transition table as shown in table 1.33.

Table 1.33 Intermediate State Table

Present State	Next State			
	a = 0		a = 1	
	State	Output	State	Output
q_0	q_{1n}	n	q_{2n}	n
q_{1n}	q_{1y}	y	q_{2n}	n
q_{1y}	q_{1y}	y	q_{2n}	n
q_{2n}	q_{1n}	n	q_{2y}	y
q_{2y}	q_{1n}	n	q_{2y}	y

The pair of states and outputs in the next state column can be rearranged as given in table 1.34 and this table is equivalent to the Moore machine.

Table 1.34 Moore Machine Equivalent to the Given Mealy Machine

Present State	Next State		Output
	a = 0	a = 1	
q_0	q_{1n}	q_{2n}	–
q_{1n}	q_{1y}	q_{2n}	n
q_{1y}	q_{1y}	q_{2n}	y
q_{2n}	q_{1n}	q_{2y}	n
q_{2y}	q_{1n}	q_{2y}	y

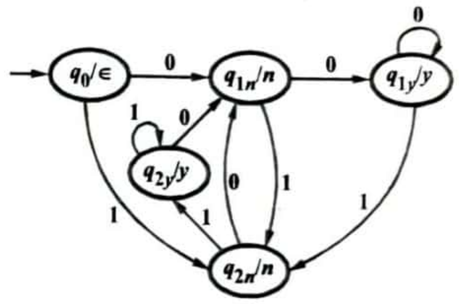


Fig. 1.34 Moore Machine

The corresponding transition diagram is shown in fig. 1.34.

Prob.25. Construct Moore machine for the following Mealy machine.

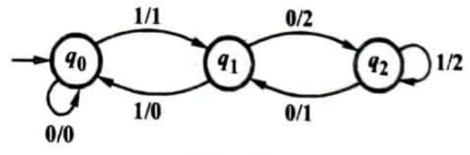


Fig. 1.35

(R.G.P.V., June 2016, Dec. 2017)

Sol. Transition table for fig. 1.35 is shown in table 1.35.

Table 1.35

State	Input			
	0		1	
	State	Output	State	Output
→ q_0	q_0	0	q_1	1
q_1	q_2	2	q_0	1
q_2	q_1	1	q_2	2

Table 1.36 is rearranged of transition table 1.35.

Table 1.36

Present State	Next State $a = 0$	Next State $a = 1$	Output
q_{00}	q_{00}	q_1	0
q_{01}	q_{00}	q_1	1
q_1	q_2	q_{01}	1
q_2	q_1	q_2	2

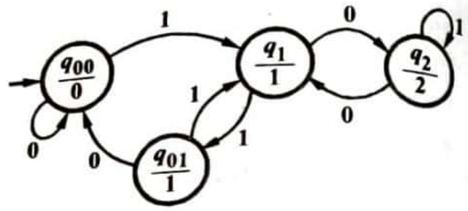


Fig. 1.36

Resultant Moore machine is shown in fig. 1.36.

Prob.26. Construct Moore machine equivalent to the given Mealy machine –

Table 1.37 Transition Table for Mealy Machine

State	Input 0: State	Input 0: Output	Input 1: State	Input 1: Output
$\rightarrow q_1$	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

(R.G.P.V., Dec. 2003, 2009)

Sol. The transition diagram corresponding to table 1.37 is shown in fig. 1.37.

Now, we construct the transition table as shown in table 1.38. The procedure is described in Q.13.

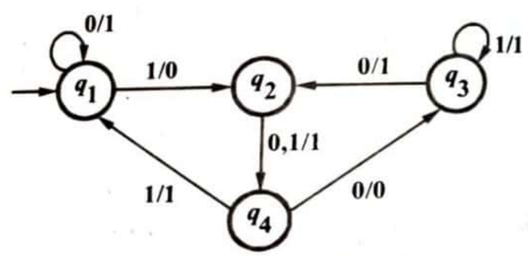


Fig. 1.37 Mealy Machine

Table 1.38 Intermediate State Table

Present State	Next State $a = 0$: State	Next State $a = 0$: Output	Next State $a = 1$: State	Next State $a = 1$: Output
$\rightarrow q_1$	q_1	1	q_{20}	0
q_{20}	q_4	1	q_4	1
q_{21}	q_4	1	q_4	1
q_{30}	q_{21}	1	q_{31}	1
q_{31}	q_{21}	1	q_{31}	1
q_4	q_{30}	0	q_1	1

The pair of states and outputs in the next state column can be rearranged as given in table 1.39.

Table 1.39 Rearranged State Table

Present State	Next State $a = 0$	Next State $a = 1$	Output
$\rightarrow q_1$	q_1	q_{20}	1
q_{20}	q_4	q_4	0
q_{21}	q_4	q_4	1
q_{30}	q_{21}	q_{31}	0
q_{31}	q_{21}	q_{31}	1
q_4	q_{30}	q_1	1

Table 1.40 Moore Machine Equivalent to the Given Mealy Machine

Present State	Next State $a = 0$	Next State $a = 1$	Output
$\rightarrow q_0$	q_1	q_{20}	0
q_1	q_1	q_{20}	1
q_{20}	q_4	q_4	0
q_{21}	q_4	q_4	1
q_{30}	q_{21}	q_{31}	0
q_{31}	q_{21}	q_{31}	1
q_4	q_{30}	q_1	1

Since, the initial state q_1 is associated with output 1. This means that with input $\in$ we get an output of 1, if the machine starts at state q_1. To overcome this situation, we add a new starting state q_0, whose state transitions are identical with those of q_1 but whose output is 0. So, table 1.39 is transformed to table as shown in table 1.40.

The corresponding transition diagram is shown in fig. 1.38.

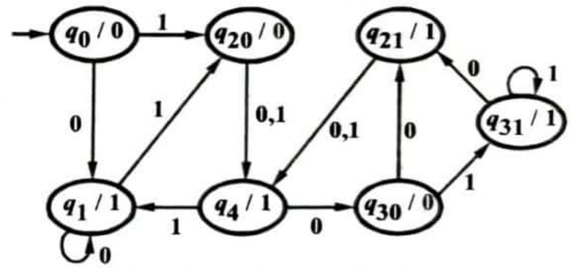


Fig. 1.38 Moore Machine

Prob.27. Give Mealy and Moore machine for the input from (0 + 1)* if the input ends in '000', output A; if the input ends in '111', output B; otherwise output C. ***(R.G.P.V., Dec. 2013)***

Sol. This machine is designed in such a way that if the input ends in 000, the output is A, if the input ends in 111, output is B. Otherwise output is C. A partial design of it can be made as follows –

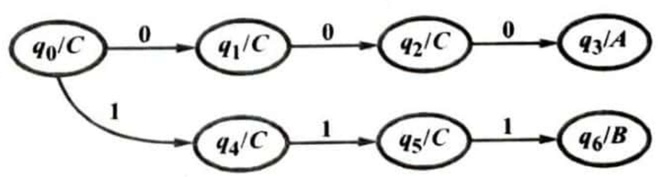


Fig. 1.39

Now, we will insert the possibilities of 1's and 0's for each state. Then the Moore machine becomes

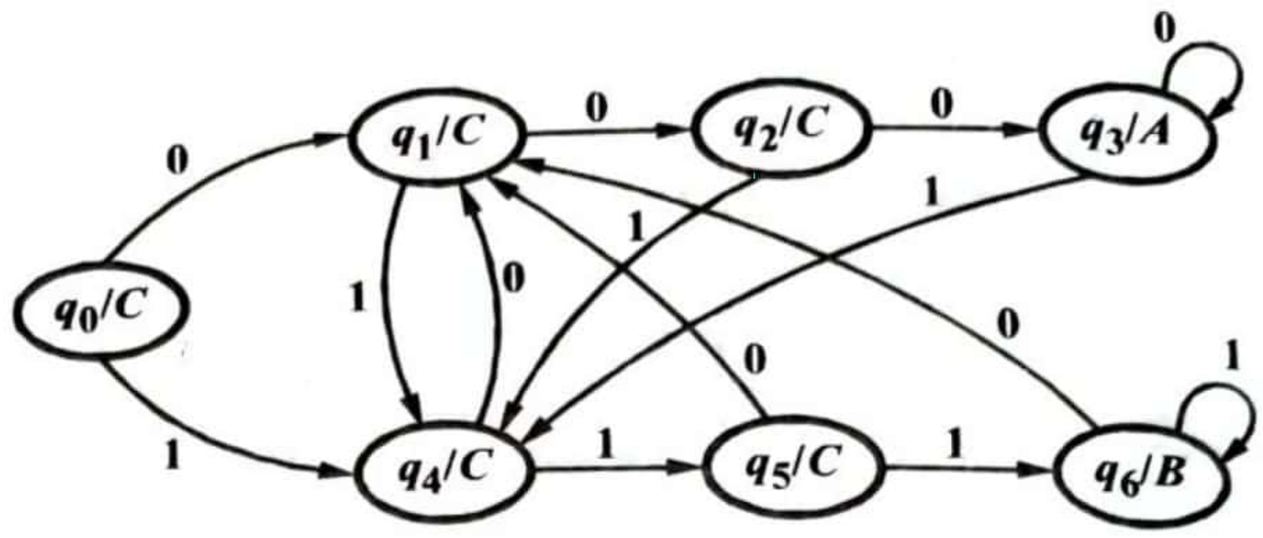


Fig. 1.40 Moore Machine

Now the Mealy machine will be as follows –

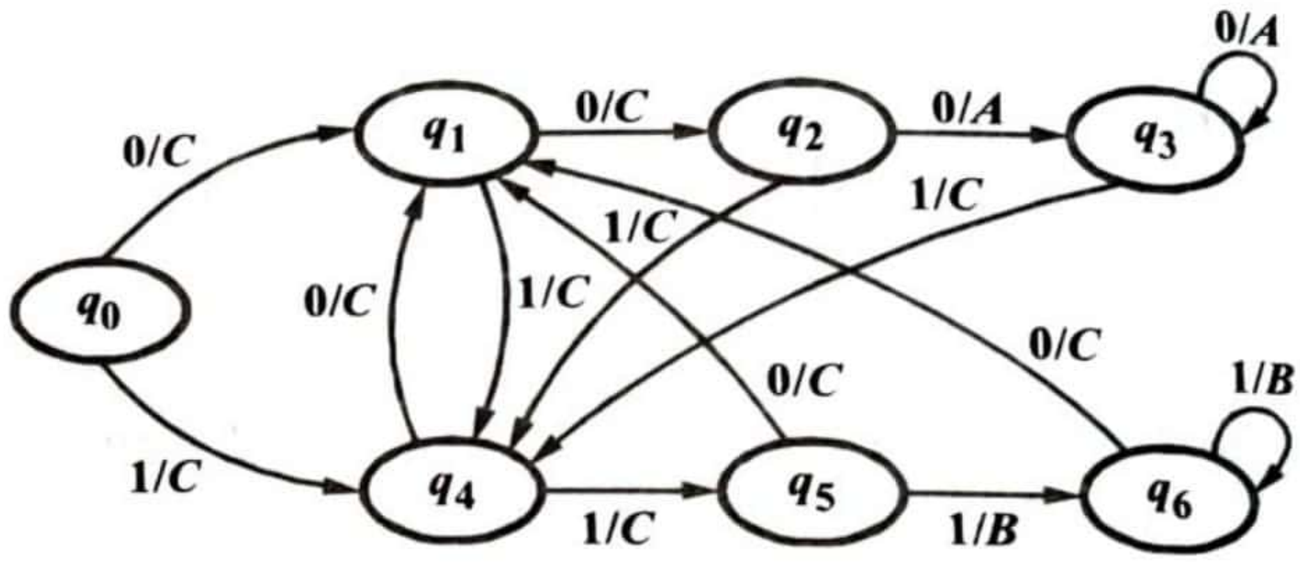


Fig. 1.41 Mealy Machine

UNIT 2

TYPES OF FINITE AUTOMATA

NON-DETERMINISTIC FINITE AUTOMATA (NDFA), DETERMINISTIC FINITE AUTOMATA MACHINES, CONVERSION OF NDFA TO DFA, MINIMIZATION OF AUTOMATA MACHINES

Q.1. Write a short note on non-deterministic finite automata.

(R.G.P.V., Dec. 2002)

Ans. Non-determinism means a choice of moves for an automaton. Instead of prescribing a unique move in every situation, this allows a set of possible moves. Formally, this can be achieved by defining the transition function such that its range is a set of possible states.

A non-deterministic finite accepter or NDFA is defined as a 5-tuple machine –

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

where, (i) Q is a finite nonempty set of states

(ii) Σ is a finite nonempty set of input symbols

(iii) δ is the transition function mapping from $Q \times \Sigma$ into 2^Q which is the power set of Q, i.e., the set of all subsets of Q

(iv) q_0 is the initial state and $q_0 \in Q$

(v) F is the set of final states and $F \subseteq Q$.

The transition function δ is also defined as,

$$\delta : Q \times (\Sigma \cup \{\Lambda\}) \rightarrow 2^Q$$

There are three major differences between DFA and NDFA. In an NDFA, the range of δ is in the power set 2^Q, so that its value is not a single symbol of Q, but a subset of it, which defines the set of all possible states that can be reached by the transition. For example, if the current state is q_1, the symbol read is a, and

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

then either q_0 or q_2 could be the next state of the NDFA. Also, we allow Λ as the second argument of δ. It means that the NDFA can make a transition

without consuming an input symbol. Although in NDFA also, the input mechanism can only travel to the right it may become stationary on some moves. Moreover, in an NDFA, the transition δ (q_i, a) may be empty, i.e., there is no transition defined for this particular situation.

Like DFAs, NDFAs can be represented by transition graphs. The vertices can be determined by Q, while an edge (q_i, q_j) with label b is in the graph if and only if $\delta(q_i, b)$ contains q_j. But, since b may be the empty string, so there can be some edges with labelled Λ or $\in$ or λ.

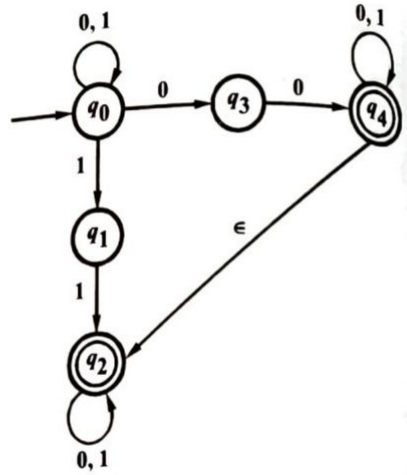


Fig. 2.1 The Transition Diagram for an NDFA

For example, consider the transition diagram for an NDFA given in fig. 2.1.

The sequence of states for the input string 0100 is shown in fig. 2.2.

An input string is accepted by an NDFA, if there is some sequence of possible moves that put the machine in a final state at the end of the string (i.e., after having the last symbol been read), and is rejected only when there is no possible sequence of moves by which any final state can be reached.

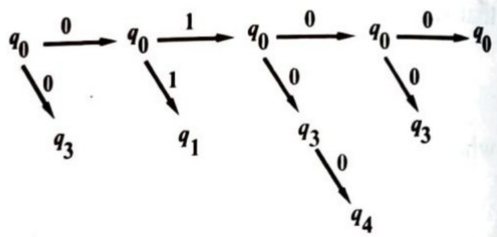


Fig. 2.2 Sequence of States while Processing 0100

In the above example shown in fig. 2.1 and fig. 2.2, since q_4 is the final state, the input string 0100 will be accepted by the NDFA through the transition $\delta(q_0, 0100) = \{q_0, q_3, q_4\}$.

Q.2. Give a detailed explanation of a deterministic finite acceptor.

Or

Discuss the finite state system with the help of a appropriate example. ***(R.G.P.V., June 2003)***

Or

Write a short note on one way finite automata. ***(R.G.P.V., Dec. 2002)***

Or

Write the definition of DFA. ***(R.G.P.V., Dec. 2014)***

Or

What do you understand by DFA (Deterministic Finite Automata) and how is it represented ? ***(R.G.P.V., June 2015)***

Ans. A finite state automaton or finite automaton deterministic finite state machine (DFSM) or deterministic finite accepter (DFA) is a 5 tuple machine that can be represented as,

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

where, (i) Q is a finite non-empty set of states

(ii) Σ is a finite non-empty set of input symbols called terminals or input alphabets

(iii) δ is a function which maps $Q \times \Sigma$ into Q and is usually called direct transition function. This function describes the change of states during the transition. Usually, this mapping is represented by a transition table or a transition diagram

(iv) q_0 is the initial state and $q_0 \in Q$

(v) F is the set of final state and $F \subseteq Q$.

For example, the operation of a deterministic finite accepter is as follows –

Initially, it is assumed to be in the initial state q_0, with its input mechanism on the leftmost symbol of the input string. During every move of the automaton, the input mechanism shifts one position to the right, so each move consumes one input symbol. At the end of the string (i.e., after having all input symbols been read), it is accepted if the automaton is in one of its final states, otherwise it is rejected. The input mechanism can only move from left to right, it cannot move in reverse direction. Also, it can read only one input symbol at a time (i.e., on each step). The transition function δ governs the transition from one internal state to another. For example, if $\delta(q_0, a) = q_1$, then, if initially the DFA is in state q_0 and the current input symbol is a, the DFA will go into state q_1.

Q.3. Define DFA. List three household applications of finite automata. ***(R.G.P.V., Dec. 2015)***

Ans. **DFA** – Refer to Q.2.

There are a variety of software design problems that are simplified by automatic conversion of regular expression notation to an efficient computer implementation of the corresponding finite automaton. Three such applications are as follows –

(i) Lexical Analyzers – The tokens of a programming language are almost without exception expressible as regular sets.

(ii) Text Editors – Certain text editors and similar programs permit the substitution of a string for any string matching a given regular expression.

(iii) Transducers – Automata with input and output are sometimes called as transducers which are used in electronics.

Q.4. Explain deterministic and non-deterministic finite automata with example. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.1. and Q.2.

Q.5. Explain finite automaton and its various types. (R.G.P.V., June 2017)

***Ans.* Finite Automaton** – Refer to Q.5, Unit-I.

Types – There are two types of finite automaton –

(i) ***DFA*** – Refer to Q.2.

(ii) ***NDFA*** – Refer to Q.1.

Q.6. What is the difference between deterministic and non-deterministic automaton ?

Ans. A ***deterministic automaton*** is one, in which each move is uniquely determined by the current configuration. If we are having the internal state, the input, and the contents of the temporary storage, we can predict the next or future behaviour of the automaton.

In a ***non-deterministic automaton***, the above said statement does not hold. At each point, it may have several possible moves, thus we can only predict a set of possible actions.

Q.7. Write short note on equivalent of DFA and NDFA.
(R.G.P.V., Dec. 2002, June 2010)

Ans. Since every DFA is an NDFA, it is clear that the class of languages accepted by NDFAs includes the regular sets (the language accepted by DFAs). However, it turns out that these are the only sets accepted by NDFAs. For every NDFA, we can construct an equivalent DFA that accepts the same language. The way a DFA simulates an NDFA is to allow the states of DFA to correspond to sets of states of the NDFA. The constructed DFA keeps track in its finite control of all states that the NDFA could be in after reading the same input as the DFA has read.

Two finite accepters M_1 and M_2 are said to be equivalent if

$$L(M_1) = L(M_2)$$

i.e., if they both accept the same languages.

Let us take an example of the DFA shown in fig. 2.3. It is equivalent to the NDFA as shown in fig. 2.4. Because they both accept the same language.

$$L(M) = \{w/w \text{ ends with } ab\}$$

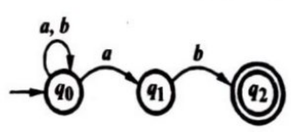


Fig. 2.3

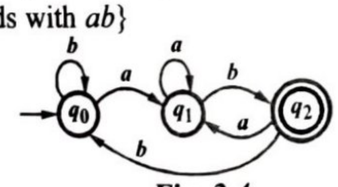


Fig. 2.4

Q.8. Prove that for every NDFA, there exists a DFA which simulates the behaviour of NDFA.

Or

Prove that if L is the set accepted by NDFA, then these exists a DFA which also accepts L.

***Ans.* Proof** – Let $M_N = (Q, \Sigma; \delta, q_0, F)$ be an NDFA accepting L. We construct a DFA, M_D, as follows –

$$M_D = (Q', \Sigma, \delta', q_0', F')$$

where, (i) $Q' = 2^Q$ (any state in Q' is denoted by $[q_1, q_2, q_3, \ldots\ldots q_j]$ where $q_1, q_2, q_3, \ldots\ldots q_j \in Q$)

(ii) $q_0' = [q_0]$

(iii) F' is the set of all subsets of Q containing an element of F.

Before going to define δ', we look at the construction of Q', q_0', and F'. M_N is initially at q_0. But, by applying an input symbol, say a, M_N can reach any of the states in $\delta(q_0, a)$. To describe M_N just after applying the input symbol a, we require all the possible states that M_N can reach after the application of a. So, M_D has to remember all these possible states at any instant of time. Hence, the states of M_D are defined as subsets of Q. As M_N starts with initial state q_0, q_0' is defined as $[q_0]$. A string u belongs to $L(M_N)$, if a final state is one of the possible states M_N reaches on processing u. So, a final state in M_D (i.e., an element of F') is any subset of Q containing some final state of M_N.

Now, we can define δ' as,

(iv) $\delta'([q_1, q_2, q_3, \ldots\ldots, q_i], a)$

$$= \delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a) \cup \ldots\ldots \cup \delta(q_i, a)$$

Equivalently,

$$\delta'([q_1, q_2, q_3, \ldots\ldots, q_i], a) = [p_1 \ldots\ldots p_j]$$

if and only if

$$\delta(\{q_1, \ldots\ldots, q_i\}, a) = \{p_1, p_2, p_3 \ldots\ldots, p_j\}$$

Before proving $L = L(M_D)$ we prove an auxiliary result,

$$\delta'(q_0', x) = [q_1, \ldots\ldots\ldots\ldots, q_i]$$

if and only if $\delta(q_0, x) = \{q_1, \ldots\ldots, q_i\}$ for all x in Σ^* ...(i)

We prove by induction on $|x|$, the "if" part, i.e.,

$$\delta'(q'_0, x) = [q_1, q_2, \ldots\ldots, q_i], \text{ if } \delta(q_0, x) = \{q_1, q_2 \ldots\ldots q_i\} \quad \text{...(ii)}$$

When $|x| = 0$, $\delta(q_0, \in) = \{q_0\}$, and by definition of δ, $\delta'(q_0', \in) = q_0' = [q_0]$. So, equation (ii) is true for x with $|x| = 0$. Thus, there is a basis for induction. Assume that the equation (ii) is only true for all strings y with $|y| \le m$. Let x be a string of length $m + 1$. We may write x as ya, where $|y| = m$ and $a \in \Sigma$. Let $\delta(q_0, y) = \{p_1, \ldots\ldots, p_j\}$ and $\delta(q_0, ya) = \{r_1, r_2, r_3, \ldots\ldots r_k\}$. As $|y| \le m$, then by inductive assumption, we have –

$$\delta(q_0', y) = [p_1, \ldots\ldots\ldots\ldots p_j] \quad \text{.....(iii)}$$

Also, $\{r_1, r_2, r_3, \ldots\ldots\ldots\ldots, r_k\}$

$$= \delta(q_0, ya)$$

$$= \delta(\delta(q_0\, y), a) = \delta(\{p_1, \ldots\ldots\ldots\ldots p_j\}, a).$$

By definition of δ' –

$$\delta'([p_1, \ldots\ldots\ldots p_j], a) = [r_1, r_2, \ldots\ldots\ldots, r_k] \quad \text{....(iv)}$$

Hence,

$$\delta'(q_0', ya) = \delta'(\delta'(q'_0, y), a)$$
$$= \delta'([p_1, \ldots\ldots, p_j], a) = [r_1, r_2, \ldots\ldots, r_k]$$

[by equation (iii) and equation (iv)]

Thus we have proved –

$$\delta'(q_0', x) = (q_1, q_2, q_3, \ldots\ldots\ldots q_i],$$

for $x = ya$

By induction, this is true for all strings.

The other part (i.e., "only if") can be proved similarly.

Now, $x \in L(M_N)$ if and only if $\delta(q, x)$ contains a state of F. As we have shown in equation (i), $\delta(q_0, x)$ contains a state of F if and only if $\delta'(q_0', x)$ is in F'. Hence $x \in L(M_N)$ if and only if $x \in L(M_D)$. The proves that DFA, M_D accepts L.

Q.9. Write the step-by-step procedure of removing Λ-moves or ∈-moves.

Ans. The transition systems can be generalized by permitting Λ-transitions or Λ-moves which are associated with a null symbol Λ. These moves are also known as $\in$-moves or $\in$-transition and are associated with a empty symbol $\in$. These transitions can occur when no input is applied. But we can convert transition system with $\in$-moves into an equivalent transition system without $\in$-moves. We can give a simple method of doing it with the help of an example.

Suppose we want to replace a $\in$-move from vertex v_1 to vertex v_2. Then we proceed as follows –

(i) Step 1 – Find all the edges starting from v_2

(ii) Step 2 – Duplicate all the edges starting from v_1, without changing the labels on the edges

(iii) Step 3 – If v_1 is an initial state, then make v_2 also as initial state

(iv) Step 4 – If v_2 is a final state, then make v_1 as the final state.

Q.10. Write a procedure that converts an NDFA to DFA.

Ans. An NDFA can be converted into DFA by following steps –

(i) Create a graph G_D having vertex $\{q_0\}$. Identify this vertex as the initial one.

(ii) Repeat the following steps (a to e) until no more edges are missing–

(a) Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of graph G_D that has no outgoing edge for some $a \in \Sigma$.

(b) Compute $\delta^*(q_i, a), \delta^*(q_j, a), \ldots\ldots\ldots, \delta^*(q_k, a)$.

(c) From the union of all these δ^*, yielding the set $\{q_l, q_m, \ldots\ldots, q_n\}$.

(d) Create a vertex for graph G_D labelled $\{q_l, q_m, \ldots\ldots, q_n\}$ if it does not already present.

(e) Add an edge to graph G_D from $\{q_i, q_j, \ldots\ldots, q_k\}$ to $\{q_l, q_m, \ldots\ldots, q_n\}$ and label it by a.

(iii) Every state of the graph G_D whose label contains any $q_f \in F_N$ is defined as a final vertex.

(iv) If M_N accepts $\in$, the vertex $\{q_0\}$ in graph G_D is also made a final vertex.

The procedure also terminates. Each pass through the loop in step (ii) adds an edge to graph G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$edges, so that the loop eventually stops.

Q.11. Give method of constructing minimum automaton.

Ans. A step-by-step method for the construction of minimum automaton is as follows –

(i) Step 1 (Construction of π_0) – By definition of 0-equivalence, $\pi_0 = \{Q_1^0, Q_2^0\}$, where Q_1^0 is the set of all final states and $Q_2^0 = Q - Q_1^0$ i.e., the set of remaining all non-final states.

(ii) Step 2 (Construction of π_{k+1} from π_k) – Let Q_i^k be any subset in π_k. If q_1 and q_2 are in Q_i^k, they are $(k+1)$-equivalent (if $\delta(q_1, a)$ and $\delta(q_2, a)$ are k-equivalent). Find out, whether $\delta(q_1, a)$ and $\delta(q_2, a)$ are in the same equivalence class in π_k for every $a \in \Sigma$. If yes, q_1 and q_2 are $(k+1)$-equivalent. Thus, in this way Q_i^k is further divided into $(k+1)$-equivalence classes. Repeat this step for every Q_i^k in π_k to get all the elements of π_{k+1}.

(iii) Step 3 (Construction of π_n) – Construct π_n for $n = 1, 2, 3, \ldots\ldots$ until $\pi_n = \pi_{n+1}$.

(iv) Step 4 (Construction of Minimum Automaton) – For the required minimum state automaton, the states are the equivalence classes (obtained in step 3), i.e., the elements of π_n. The state table can be obtained by replacing a state q by the corresponding equivalence class $[q]$.

The construction of π_0, π_1, etc., is easy when the transition table is given. $\pi_0 = \{Q_1^0, Q_2^0\}$, where $Q_1^0 = F$ and $Q_2^0 = Q - F$. The subset in π_1 can be obtained by partitioning subsets of π_0 further. If $q_1, q_2 \in Q_1^0$, then consider the states in each a-column, where $a \in \Sigma$ corresponding to q_1 and q_2. If q_1 and q_2 are in the same subset of π_0, then they are 1-equivalent. If the states under some a-column are in different subsets of π_0, then q_1 and q_2 are not 1-equivalent. Generally, $(k+1)$-equivalent states are obtained by applying the above method for q_1 and q_2 in Q_i^k.

Q.12. Write short note on epsilon transition. **(R.G.P.V., Nov. 2019)**

Ans. An epsilon transition (also epsilon move or lambda transition) allows an automaton to change its state spontaneously, i.e. without consuming an input symbol. It may appear in almost all kinds of nondeterministic automaton in formal language theory, in particular – Nondeterministic turing machine, nondeterministic pushdown automaton, nondeterministic finite automaton. Epsilon transition is denoted by $\in$ symbol.

NUMERICAL PROBLEMS

Prob.1. Design a NFA for $\{cbab^n \mid n >= 0\}$. **(R.G.P.V., June 2016)**

Sol. NFA for the language

$$L = \{cbab^n \mid n >= 0\}$$

is

$$M = (\{q_0, q_1, q_2, q_3\}, \{a, b, c\}, \delta, q_0, \{q_3\})$$

Here, the NFA is such that it accepts all strings of the type *cba*, *cbab*, *cbabb*, *cbabbb*, ...

So, NFA for the given language is shown in fig. 2.5.

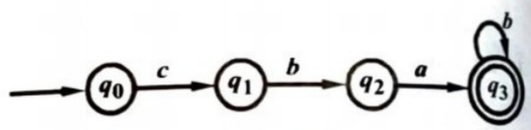


Fig. 2.5

Prob.2. Convert the following NFA with $\in$ to equivalent DFA.

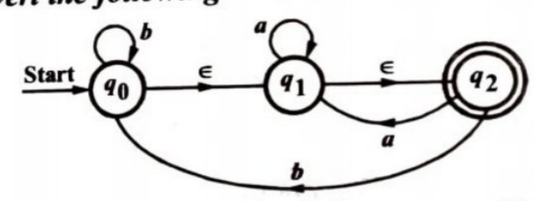


Fig. 2.6 **(R.G.P.V., June 2011)**

Sol. The conversion of NDFA to DFA is given step-by-step in fig. 2.7.

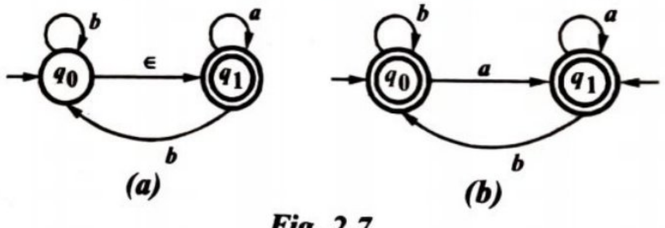


Fig. 2.7

Fig. 2.8

The DFA shown in fig. 2.7 (b) can be redrawn as fig. 2.8.

Prob.3. Construct NFA for the following grammar –

$$S \rightarrow Ab|ab,\ A \rightarrow Ab|Bb,\ B \rightarrow Ba|a$$ **(R.G.P.V., June 2016)**

Sol. The above grammar is left linear grammar which accepts the language of the form $\{a^n b^m \mid n, m \geq 1\}$. It should be converted to right linear grammar.

When the above grammar is converted to right linear grammar. It is represented as

$$S \rightarrow bA|ba,\ A \rightarrow bA|bB,\ B \rightarrow aB|a$$

Let $M = (Q, \Sigma, \delta[S], [\in])$, where $Q = \{[S], [bA], [A], [ba], [a], [bB], [B], [aB], [\in]\}$, $T = \{a, b\}$ and $\in$ is

$$\delta([S], \in) = [bA], [ba]$$
$$\delta([A], \in) = [bA], [bB]$$
$$\delta([B], \in) = [aB], [a]$$
$$\delta([bA], b) = [A]$$
$$\delta([ba], b) = [a]$$
$$\delta([a], a) = [\in]$$
$$\delta([bB], b) = [B]$$
$$\delta([aB], a) = [B]$$

The NFA for right linear grammar is shown in fig. 2.9. To get the final NFA for the given left linear grammar, we reverse the NFA and the result obtained is shown in fig. 2.10.

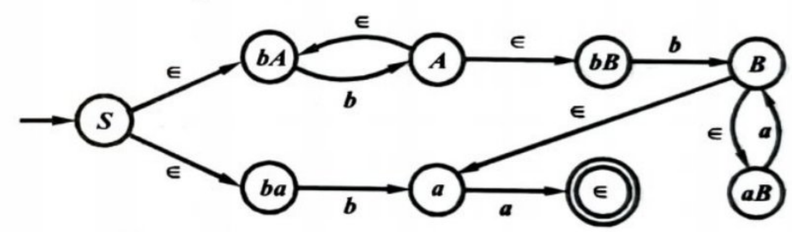


Fig. 2.9 NFA for Right Linear Grammar

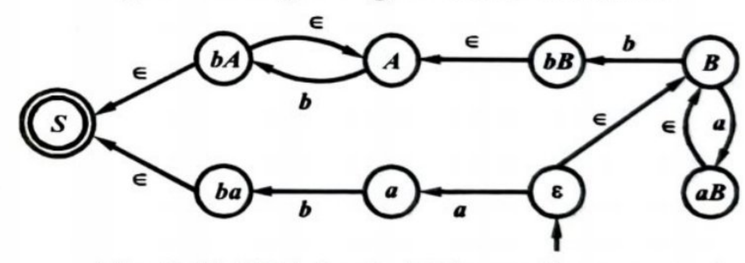


Fig. 2.10 NFA for Left Linear Grammar

Prob.4. Construct DFA equivalent to the NFA.

$$M = (\{p, q, r, s\}, \{0, 1\}, \delta, p, \{q, s\})$$

where δ is defined in the following table.

δ	0	1
p	{q, s}	{q}
q	{r}	{q, r}
r	{s}	{p}
s	–	{p}

(R.G.P.V., Dec. 2014)

Sol. The DFA M_1 equivalent to M is defined as follows –

$$M_1 = \{2^Q, \{0, 1\}, \delta, p, F\}$$

where, $F = \{[q, s], [p, q, s], [r, q, s], [p, q, r, s]\}$

We start the construction by considering $[p]$ first. The state table is given in table 2.1.

Table 2.1 State Table of DFA

State	0	1
$[p]$	$[q,s]$	$[q]$
$[q]$	$[r]$	$[q,r]$
$[q,s]$	$[r]$	$[q,r,p]$
$[r]$	$[s]$	$[p]$
$[q,r]$	$[r,s]$	$[q,r,p]$
$[q,r,p]$	$[q,s,r]$	$[p,q,r]$
$[s]$	ϕ	$[p]$
$[r,s]$	$[s]$	$[p]$
$[q,s,r]$	$[r,s]$	$[p,q,r]$

Prob.5. Define deterministic finite automata. Draw DFA that accepts any string which ends with 1 or it ends with an even number of 0's following the last 1. Alphabets are {0, 1}. (R.G.P.V., Dec. 2007)

Table 2.2

State	Input 0	Input 1
$\rightarrow q_0$	q_0	q_1
q_1	q_2	q_1
q_2	q_1	q_1

Sol. DFA – Refer to Q.2.

A DFA can be determined as,

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

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

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

$F = \{q_1\}$

and δ is defined in form of transition table 2.2.

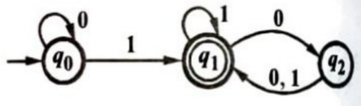


Fig. 2.11

Class of language accepted by DFA M is

$$L = \{0, 1\}^*\{1\} \text{ or } \{0, 1\}^*\{1\}\ \{0, 0\}^*$$

Prob.6. Construct as deterministic accepter equivalent to $M_N = (\{q_0, q_1\}, \{a, b\}, \delta, q_0, \{q_0\})$. δ is defined by its state table given in table 2.3.

Table 2.3 State Table of M_N

State	Input a	Input b
$\rightarrow q_0$	q_0	q_1
q_1	q_1	q_0, q_1

Sol. For the deterministic accepter M_D –

(i) The states are subsets of $\{q_0, q_1\}$, i.e., ϕ, $[q_0]$, $[q_0, q_1]$, $[q_1]$.

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

(iii) $[q_0]$ and $[q_0, q_1]$ are the final states, as they are the only states having $[q_0]$.

(iv) δ is defined by the state table given in table 2.4.

q_0 and q_1 appear in the rows corresponding to q_0 and q_1 and the column corresponding to a. So, $\delta([q_0, q_1], a) = [q_0, q_1]$.

When M_N has n states, the corresponding finite accepter has 2^n states. However, we need not construct δ for all these states, but only for those states reachable from $[q_0]$, this is because, we have to construct only M_D accepting $L(M_N)$. So, we start the construction of δ for $[q_0]$. We proceed by considering only states appearing earlier under input columns and constructing δ for them. We halt when no more new states appear under the input columns.

Table 2.4 State Table of M_D

State	Input a	Input b
ϕ	ϕ	ϕ
$[q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

Prob.7. Convert the following NFA into an equivalent deterministic machine –

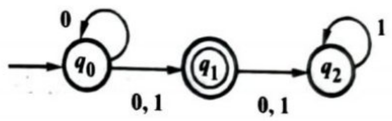


Fig. 2.12

Sol. The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1)$ in G_D and add an edge labeled 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled 1 between it and $\{q_0\}$.

Similarly,

$$\delta_N(\{q_1\}, 0) = \{q_2\}$$

$$\delta_N(\{q_1\}, 1) = \{q_2\}$$

$$\delta_N(\{q_0, q_1\}, 0) = \delta(\{q_0\}, 0) \cup \delta(\{q_1\}, 0)$$

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

$$\delta_N(\{q_0, q_1, q_2\}, 0) = \delta(\{q_0\}, 0) \cup \delta(\{q_1\}, 0) \cup \delta(\{q_2\}, 0)$$

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

$$\delta_N(\{q_0, q_1, q_2\}, 1) = \{q_1, q_2\}$$

$$\delta_N(\{q_0, q_1,\}, 1) = \delta(\{q_0\}, 1) \cup \delta(\{q_1\}, 1)$$

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

$$\delta_N(\{q_1, q_2\}, 0) = \{q_2\}$$

$$\delta_N(\{q_1, q_2\}, 1) = \{q_2\}$$

$$\delta_N(\{q_2\}, 0) = \phi$$

$$\delta_N(\{q_2\}, 1) = \{q_2\}$$

The result, shown in fig. 2.13, is a DFA, equivalent to the NFA.

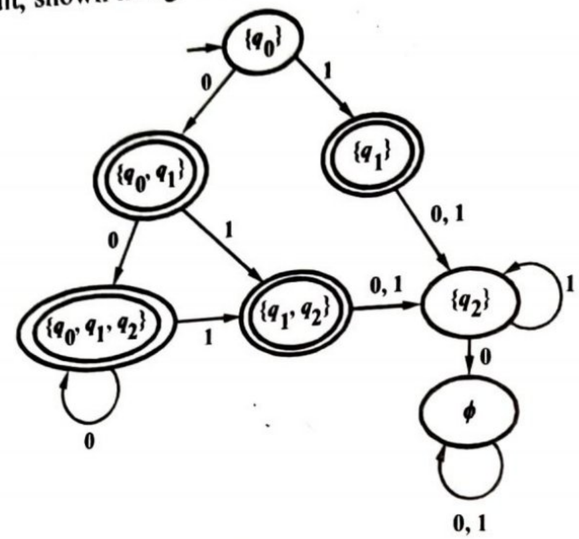


Fig. 2.13

Prob.8. Convert the following NDFA to deterministic finite automata–

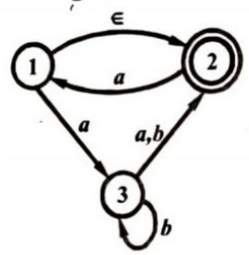


Fig. 2.14 ***(R.G.P.V., Dec. 2007)***

Sol. We construct the transition table corresponding to NDFA without ∈-moves as given in table 2.5.

The successor table is constructed for DFA and given in table 2.6.

Table 2.5

State	Input	
	a	b
①	1, 3	–
②	1	–
3	2	3, 2

Table 2.6 Transition Table for the DFA

State	Input	
	a	b
[1]	[1, 3]	ϕ
[1, 3]	[1, 2, 3]	[2, 3]
[2, 3]	[1, 2]	[2, 3]
[1, 2]	[1, 3]	ϕ
[1, 2, 3]	[1, 2, 3]	[2, 3]

The state diagram for the successor table is the required DFA as indicated in fig. 2.14 since ① and ② are the only final states of NDFA, therefore all states are the final states of DFA.

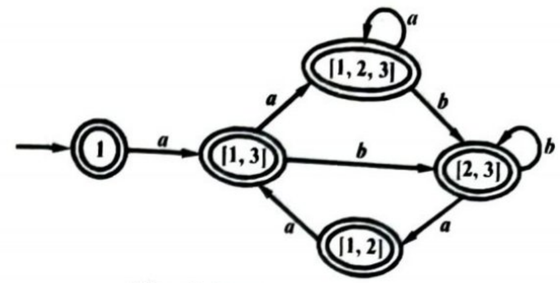


Fig. 2.15 Constructed DFA

Prob.9. Construct a DFA M′ equivalent to NFA M = ({q0, q1, q2, q3}, {a, b}, δ, q0, {q3}) where δ is given by table 2.7.

Table 2.7 State Transition Table for NFA M

State	Input	
	a	b
→ q_0	q_0, q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_3
q_3	–	q_2

(R.G.P.V., Nov. 2018)

Sol. **Construction of DFA** – Table 2.7 shows the transition table of the given NFA.

Table 2.8 Transition Table for the DFA

State	Input	
	a	b
→ $[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$
$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$
$[q_0, q_1, q_3]$	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$

The successor table is constructed for the DFA, and given in table 2.8. Fig. 2.16 shows the transition diagram reduced DFA.

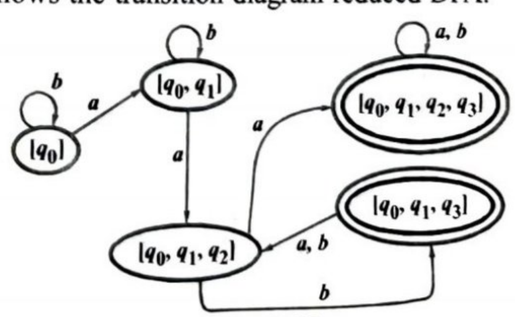


Fig. 2.16

Prob.10. Determine the DFA equivalent to the given NFA.

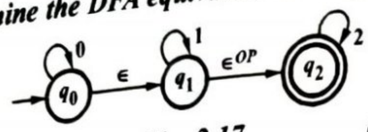


Fig. 2.17 **(R.G.P.V., Dec. 2016)**

Or

Convert the following NFA with ∈-move to equivalent DFA.

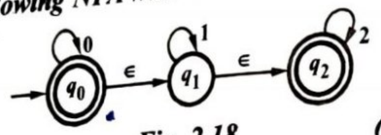


Fig. 2.18 **(R.G.P.V., Dec. 2017)**

Sol. We know that

∈-closure (q) = set of all vertices p such that there is a path from q to p labeled ∈.

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

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

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

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

Let $\{q_0, q_1, q_2\} = A_1$, $\{q_1, q_2\} = A_2$ and $\{q_2\} = A_3$

So, $\hat{\delta}(A_1, 0) = \in\text{-closure}(\delta(\hat{\delta}(A_1, \in), 0)$

$= \in\text{-closure}(\delta(\hat{\delta}(q_0, \in) \cup \hat{\delta}(q_1, \in) \cup \hat{\delta}(q_2, \in), 0))$

$= \in\text{-closure}(\delta(A_1 \cup A_2 \cup A_3, 0))$

$= \in\text{-closure}(\delta(A_1, 0))$ $(\because A_3 \subseteq A_2 \subseteq A_1)$

$= \in\text{-closure}(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))$

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

$= A_1$

Similarly $\hat{\delta}(A_1, 1) = A_2$

$\hat{\delta}(A_1, 2) = A_3$

$\hat{\delta}(A_2, 0) = \phi$

$\hat{\delta}(A_2, 1) = A_2$

$\hat{\delta}(A_2, 2) = A_3$

$\hat{\delta}(A_3, 0) = \phi$

$\hat{\delta}(A_3, 1) = \phi$

$\hat{\delta}(A_3, 2) = A_3$

Table 2.9

States \ Inputs	0	1	2
q_0, q_1, q_2	q_0, q_1, q_2	q_1, q_2	q_2
q_1, q_2	ϕ	q_1, q_2	q_2
q_2	ϕ	ϕ	q_2

Transition matrix is shown in table 2.9. So, the required DFA is shown in fig. 2.19.

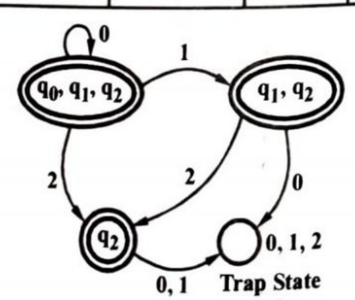


Fig. 2.19

Prob.11. Consider the FA below and construct the, smallest DFA which accepts the same language.

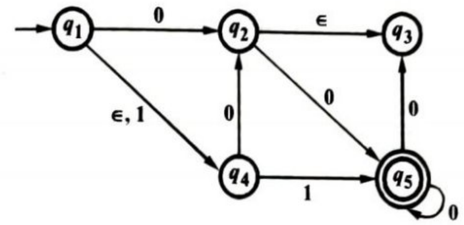


Fig. 2.20 **(R.G.P.V., Dec. 2013)**

Sol. The given FA is redrawn as –

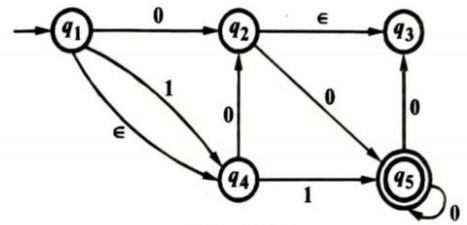


Fig. 2.21

In this transition diagram, there are ∈ moves, so we have to remove ∈ moves. We first remove ∈ move from q_1 to q_4 to get fig. 2.22.

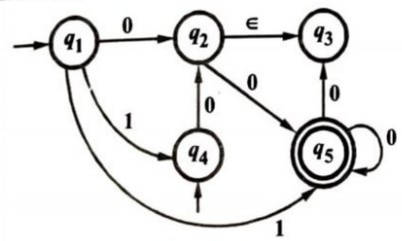


Fig. 2.22

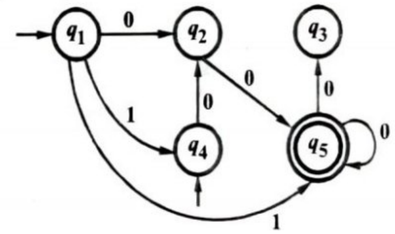


Fig. 2.23

Now, we eliminate the ∈ move from q_2 to q_3 in fig. 2.22 to get fig. 2.23.

Now, we construct the transition table as given in table 2.10 corresponding to fig. 2.23.

Table 2.10

States/Σ	0	1
→ q_1	q_2	q_4, q_5
q_2	q_5	–
q_3	–	–
→ q_4	q_2	–
q_5	q_3, q_5	–

Table 2.11

Q	Q_0	Q_1
→ $[q_1]$	$[q_2]$	$[q_4, q_5]$
$[q_2]$	$[q_5]$	ϕ
→ $[q_4, q_5]$	$[q_2, q_3, q_5]$	ϕ
$[q_5]$	$[q_3, q_5]$	ϕ
$[q_2, q_3, q_5]$	$[q_3, q_5]$	ϕ
$[q_3, q_5]$	$[q_3, q_5]$	ϕ

The successor table is constructed for DFA and given in table 2.11.

The state diagram for the successor table is the required DFA as shown in fig. 2.24.

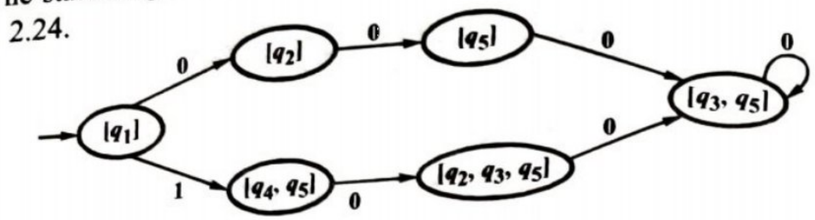


Fig. 2.24

Prob.12. Construct a minimum state automaton equivalent to the finite automaton given in fig. 2.25.

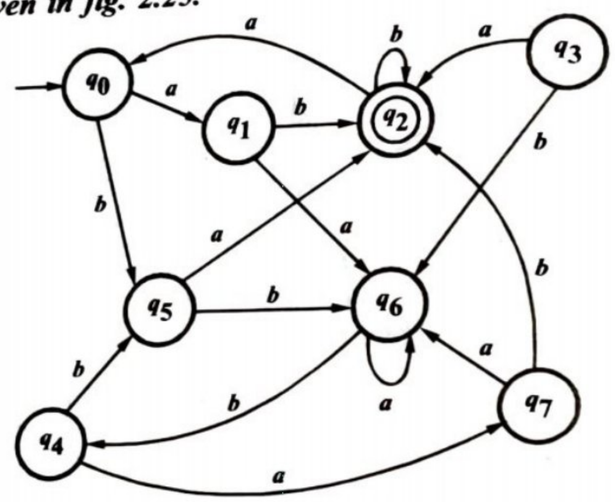


Fig. 2.25

Sol. It will be easier if we construct the transition table as shown in table 2.12.

Table 2.12 Transition Table

State	Input	
	a	b
→ q_0	q_1	q_5
q_1	q_6	q_2
q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

Application of Step 1, which is defined in Q.11, gives –

$Q_1^0 = F = \{q_2\}$, $Q_2^0 = Q - Q_1^0$

So, $\pi_0 = \{\{q_2\}, \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}\}$

If we observe π_0, we find that $\{q_2\}$ cannot be further partitioned. So, $Q_1' = \{q_2\}$. Consider q_0 and $q_1 \in Q_2^0$. The entries under a-column corresponding to q_0 and q_1, are q_1 and q_6; they lie in Q_2^0. The entries under b-column are q_5 and q_2. $q_2 \in Q_1^0$ and $q_5 \in Q_2^0$. Therefore, q_0 and q_1 are not 1-equivalent. In the same way, q_0 is not 1-equivalent to q_3, q_5, q_7.

Now take q_0 and q_4. The entries under a-column are q_1, and q_7. Both are in Q_2^0. The entries under b-column are q_5, q_5. So q_0 and q_4 are 1-equivalent.

Similarly, q_0 and q_6 are 1-equivalent. $\{q_0, q_4, q_6\}$ is a subset in π_1. So, $Q_2' = \{q_0, q_4, q_6\}$.

Repeat the construction by considering this time, q_1 and any one of the states q_3, q_5 and q_7. We find, the q_1 is not 1-equivalent to q_3 and q_5 but it is 1-equivalent to q_7. Hence $Q_3' = \{q_1, q_7\}$. The elements left in Q_2^0 are q_3 and q_5. By considering the entries under a-column and b-column, we find that q_3 and q_5 are 1-equivalent. So $Q_4' = \{q_3, q_5\}$.

Therefore –

$$\pi_1 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

$\{q_2\}$ will be in π_2 as it cannot be further partitioned. Now, the entries under a-column corresponding to q_0 and q_4 are q_1 and q_7, and these lie in the same equivalence class in π_1. The entries under b-column are q_5, q_5. So q_0 and q_4 are 2-equivalent. But there is no 2-equivalence between q_0 and q_6. Hence, $\{q_0, q_4, q_6\}$ is partitioned into $\{q_0, q_4\}$ and $\{q_6\}$. q_1 and q_7 are 2-equivalent. Also, q_3 and q_5 are 2-equivalent. Thus,

$$\pi_2 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

q_0 and q_4 are 3-equivalent. Also q_1 and q_7 are 3-equivalent. And q_3 and q_5 are also 3-equivalent. Therefore,

$$\pi_3 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

As, $\pi_2 = \pi_3$, π_2 gives the equivalence classes, the minimum state automaton is given as,

$$M_D = \{Q', \{a, b\}, \delta', q_0', F'\}$$

where, $Q' = \{[q_2], [q_0, q_4], [q_6], [q_1, q_7], [q_3, q_5]\}$

$q_0' = [q_0, q_4]$, $F' = [q_2]$

and δ' is given by the transition table shown in table 2.13.

Table 2.13 Transition Table of Minimum State Automaton

State	Input	
	a	b
$[q_0, q_4]$	$[q_1, q_7]$	$[q_3, q_5]$
$[q_1, q_7]$	$[q_6]$	$[q_2]$
$[q_2]$	$[q_0, q_4]$	$[q_2]$
$[q_3, q_5]$	$[q_2]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_0, q_4]$

The transition graph (or diagram) for the minimum state automaton is shown in fig. 2.26.

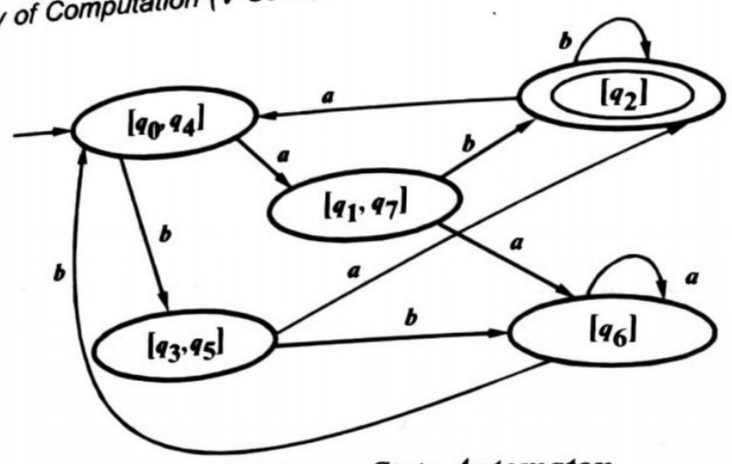


Fig. 2.26 Minimum State Automaton

The states q_0 and q_4 are identified and treated as one state. Also q_1, q_7 and q_3, q_5. But the transitions in both the diagrams (i.e., of fig. 2.25 and fig. 2.26) are the same. If there is an arrow from q_i to q_j with label a (or b), then there is an arrow from $[q_i]$ to $[q_j]$ with the same label a (or b) in the diagram for minimum state automaton.

Symbolically, if $\delta(q_i, a) = q_j$, then $\delta'([q_i], a) = [q_j]$.

Prob.13. Minimize the states of the following finite automata.

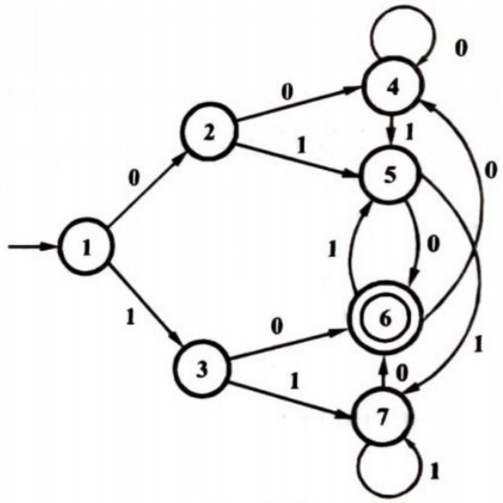


Fig. 2.27 ***(R.G.P.V., Dec. 2005)***

Sol. We can construct the transition table shown in table 2.14.

Since there is only one final state 6,

$$Q_1^0 = \{6\},\ Q_2^0 = Q - Q_1^0$$

Hence, $\pi_0 = \{\{6\}, \{1,2,3,4,5,7\}\}$

As $\{6\}$ cannot be partitioned further,

$$Q_1^1 = \{6\}$$

Now 1 is 1-equivalent to 2, 4, but not to 3, 5, 7 and so $Q_2^1 = \{1, 2, 4\}$. 3 is 1-equivalent to 5,7.

Hence, $Q_3^1 = \{3,5,7\}$. Thus

$$\pi_1 = \{\{6\}, \{1,2,4\}, \{3,5,7\}\}$$

$$Q_1^2 = \{6\}$$

As 1 is 2-equivalent to 2,4. So

$$Q_2^2 = \{1,2,4\}$$

As 3 is 2-equivalent to 5,7 So

$$Q_3^2 = \{3,5,7\}$$

Therefore,

$$\pi_2 = \{\{6\}, \{1,2,4\}, \{3,5,7\}\}$$

As $\pi_2 = \pi_1$, π_1 gives us the equivalence classes, the minimum state automaton is –

$$M = \left(Q', \{0, 1\}, \delta', q'_0, F'\right), \text{ where}$$

$$Q' = \{[6], [1,2,4], [3,5,7]\}$$

$$q'_0 = [1,2,4],\ F' = [6]$$

δ is given by table 2.15.

Table 2.14 Transition Table

State	Input 0	Input 1
→1	2	3
2	4	5
3	6	7
4	4	5
5	6	7
6	4	5
7	6	7

Table 2.15 Transition Table of Minimum State Automaton

State	Input 0	Input 1
→[1,2,4]	[1,2,4]	[3,5,7]
[3,5, 7]	[6]	[3,5,7]
[6]	[1,2,4]	[3,5,7]

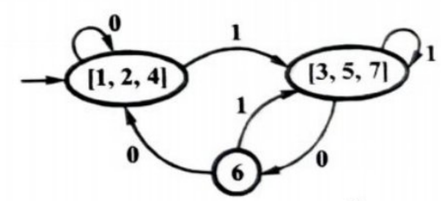


Fig. 2.28 Minimum State Automata

The transition diagram of M is given in fig. 2.28.

Prob.14. Construct a minimum state automaton equivalent to given automaton, whose transition graph is as ahead.

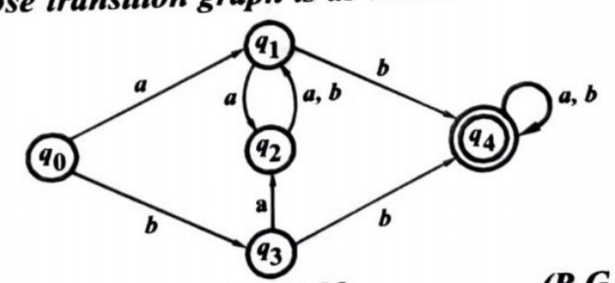


Fig. 2.29 ***(R.G.P.V., June 2008)***

Sol. We construct the transition table in table 2.16.

Table 2.16 Transition Table

State/Σ	a	b
→ q_0	q_1	q_3
q_1	q_2	q_4
q_2	q_1	q_1
q_3	q_2	q_4
q_4	q_4	q_4

By applying step (i) we get

$$Q_1^0 = F = \{q_4\}$$

$$Q_2^0 = Q - Q_1^0$$

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

So, $\pi_0 = \{\{q_4\}, \{q_0, q_1, q_2, q_3\}\}$

As $\{q_4\}$ cannot be partitioned further,

$$Q'_1 = \{q_4\}$$

Now q_0 is 1-equivalent to q_2 but not to q_1, q_3 and so $Q'_2 = \{q_0, q_2\}$.

q_1 is 1-equivalent to q_3. Hence $Q'_3 = \{q_1, q_3\}$. Thus,

$$\pi_1 = \{\{q_4\}, \{q_0, q_2\}, \{q_1, q_3\}$$

$$Q_1^2 = \{q_4\}$$

q_0 is 2-equivalent to q_2

$$Q_2^2 = \{q_0, q_2\}$$

q_1 is 2-equivalent to q_3

$$Q_3^2 = \{q_1, q_3\}$$

Thus, $\pi_2 = \{\{q_4\}, \{q_0, q_2\}, \{q_1, q_3\}\}$

As $\pi_2 = \pi_1$, π_2 gives us equivalence classes, the minimum state automaton is $M' = (Q', \{a, b\}, \delta', q'_0, F')$, where

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

$$q'_0 = \{q_0, q_2\}$$

$$F' = \{q_4\}$$

δ' is given by table 2.17.

Table 2.17

State/Σ	a	b
$[q_0, q_2]$	$[q_1, q_3]$	$[q_1, q_3]$
$[q_1, q_3]$	$[q_0, q_2]$	$[q_4]$
$[q_4]$	$[q_4]$	$[q_4]$

Prob.15. Construct a minimum state automaton for the following DFA –

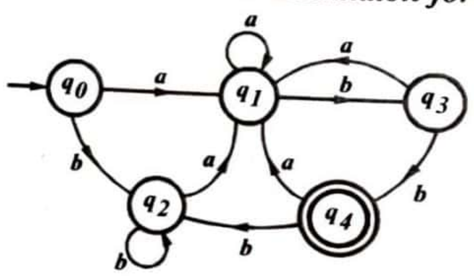


Fig. 2.30 ***(R.G.P.V., June 2011)***

Sol. We can construct the transition table shown in table 2.18.

Table 2.18 Transition Table

State	Input a	Input b
→ q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
q_4	q_1	q_2

Table 2.19 Transition Table for Minimum State Automaton

State	Input a	Input b
$[q_0, q_2]$	$[q_1]$	$[q_0, q_2]$
$[q_1]$	$[q_1]$	$[q_3]$
$[q_3]$	$[q_1]$	$[q_4]$
$[q_4]$	$[q_1]$	$[q_0, q_2]$

As there is only one final state, i.e., q_4

$$Q_1^0 = \{q_4\}, Q_2^0 = Q - Q_1^0 = \{q_0, q_1, q_2, q_3\}$$

Hence, $\pi_0 = \{\{q_4\}, \{q_0, q_1, q_2, q_3\}\}$

As $\{q_4\}$ cannot be further partitioned

$$Q'_1 = \{q_4\}$$

Now q_0 is 1-equivalent to q_0, q_1, q_2 but not to q_3, so

$$Q'_2 = \{q_0, q_1, q_2\}$$

q_3 cannot be further partitioned, so

$$Q'_3 = \{q_3\}$$

Thus, $\pi_1 = \{\{q_4\}, \{q_0, q_1, q_2\}, \{q_3\}\}$

$$Q_1^2 = \{q_4\}$$

q_0 is 2-equivalent to q_1 but not to q_2, so

$$Q_2^2 = \{q_0, q_2\}$$

$$Q_3^2 = \{q_1\}$$

$$Q_4^2 = \{q_3\}$$

Thus, $\pi_2 = \{\{q_4\}, \{q_0, q_2\}, \{q_1\}, \{q_3\}\}$

$$Q_1^3 = \{q_4\}$$

Now, q_0 is 3-equivalent to q_2, so

$$Q_2^3 = \{q_0, q_2\}$$

$$Q_3^3 = \{q_1\}$$

$$Q_4^3 = \{q_3\}$$

Thus, $\pi_3 = \{\{q_4\}, \{q_0, q_2\}, \{q_1\}, \{q_3\}\}$

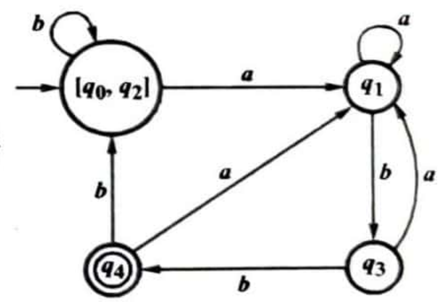


Fig. 2.31 Minimum State Automaton

Since $\pi_3 = \pi_2$, π_2 gives us the equivalence classes, the minimum state automaton is –

$$M_D = (Q', \{a, b\}, \delta', q'_0, F')$$

where, $Q' = \{[q_4], [q_0, q_2], [q_1], [q_3]\}$

$q'_0 = [q_0, q_2]$, $F' = [q_4]$ and δ' is given by table 2.19.

The transition diagram for M_D is given in fig. 2.31.

REGULAR EXPRESSION, ARDEN'S THEOREM, MEANING OF UNION, INTERSECTION, CONCATENATION AND CLOSURE, 2 WAY DFA

Q.13. Explain the term alphabet and string. ***(R.G.P.V., June 2015)***

Ans. **Alphabet** – An alphabet is a finite set of symbols. In the case of a common language such as English, we would want the alphabet to include 26 letters, both uppercase and lowercase, as well as blanks and various punctuation symbols. In the case of a programming language, we would want to add the 10 numeric digits.

String – A string (or word) is a finite sequence of symbols juxtaposed. For example, a, b, and c are symbols and $abcb$ is a string.

The ***length*** of a string w, denoted $|w|$, is the number of symbols in the string. For example, $abcb$ has length 4. The empty string denoted by $\in$, is the string consisting of zero symbols.

Thus, $|\in| = 0$

A ***prefix*** of a string is any number of leading symbols of that string and a ***suffix*** is any number of trailing symbols. For example, string abc has prefixes $\in$, a, ab and abc, and its suffixes are $\in$, c, bc and abc. A prefix or suffix of a string other than the string itself, is called a proper prefix or suffix.

The ***concatenation*** of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1a_2 \ldots\ldots a_n$$

and

$$v = b_1b_2 \ldots\ldots b_m$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1a_2 \ldots\ldots a_n\ b_1b_2 \ldots\ldots b_m$$

The empty string is the identity for the concatenation operator. That is $\in w = w\in = w$ for each string w. The ***reverse*** of a string is obtained by writing the symbols in reverse order, if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \ldots a_2a_1$$

Q.14. What do you mean by empty string and null set ?

Ans. Empty string is defined as string with no letters. It is denoted by the symbol $\in$. It is also denoted as Λ or λ. No matter what alphabet we are considering, the null string is always $\in$.

The null set is the set that has no words or strings. It is denoted by ϕ.

$$\phi = \{\ \}$$

even $\in$ is not belongs to ϕ.

One more intresting fact is that, for any language L, $L + \phi = L$ but $L + \{\in\} \neq L$

here '+' denotes "union of sets".

ϕ is a language even though it has no words.

Q.15. What do you mean by a language ?

Ans. A language (formal) is a set of strings of symbols from some alphabet. The empty set ϕ and the set consisting of the empty string $\{\in\}$ are languages. The set of palindromes (strings that read the same forward and backward) over the alphabet $\{0, 1\}$ is an infinite language. Some members of this languages are $\in$, 0, 1, 00, 11, 010 and 1101011. The set of all palindromes over an infinite collection of symbols is technically not a language because its strings are not collectively built from an alphabet.

Another definition of language is the set of all strings over a fixed alphabet Σ. We denote this language by Σ^*. For example, if

$$\Sigma = \{a\}$$

then $\Sigma^* = \{\in, a, aa, aaa, \ldots\ldots\}$

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

then $\Sigma^* = \{\in, 0, 1, 00, 01, 10, 11, 000, \ldots\ldots\}$.

Q.16. Define regular set. Explain it with examples.

Or

Write short note on regular languages. ***(R.G.P.V., June 2005, 2008)***

Ans. A regular set is the set represented by a regular expression. For example in $a, b \in \Sigma$, then a denotes the set $\{a\}$, $a + b$ denotes $\{a, b\}$, ab denotes $\{ab\}$, a^* denotes the set $\{\in, a, aa, aaa, \ldots\ldots\}$ and $(a + b)^*$ denotes $\{a, b\}^*$.

For example, the following sets can be described as regular expression –

(i) $\{1\}$, $\{0\}$ are represented by **1** and **0**, respectively. 101 is obtained by concatenating 1, 0 and 1. So, $\{101\}$ is represented by **101**

(ii) $\{abba\}$ is represented by ***abba***

(iii) As $\{ab, ba\}$ is the union of $\{ab\}$ and $\{ba\}$, $\{ab, ba\}$ is represented by $\boldsymbol{ab + ba}$

(iv) The set $\{\in, 10\}$ is represented by $\boldsymbol{\in + 10}$

(v) The set $\{abb, a, b, bba\}$ is represented by $\boldsymbol{abb + a + b + bba}$

(vi) As $\{\in, a, aa, aaa, \ldots\ldots\}$ is simply $\{a\}^*$, it is represented by $\boldsymbol{a^*}$

(vii) Any element in $\{1, 11, 111, \ldots\ldots\}$ can be obtained by concatenating 1 and any element of $\{1\}^*$. Hence $\mathbf{1(1)^*}$ represents $\{1, 11, 111, \ldots\ldots\}$. It is denoted as $\{1\}^+$.

Q.17. Explain regular grammars. ***(R.G.P.V., May 2018)***

Ans. A formal grammar, in which every production is restricted to have following forms –

(i) $A \to \alpha C$

(ii) $A \to \alpha$, where A and C are non-terminals, and α is a terminal symbol.

This is known as regular grammar.

Since, if the format of every production in context-free grammar is restricted to $A \to \alpha C$ or $A \to \alpha$, then the grammar can specify only regular sets, and hence there exist a finite automata accepting L(G), if G is regular, therefore it is called regular grammar.

Equivalence of Regular Grammars and Finite Automata –

A finite automata accepting L(G) from a regular grammar G can be obtained as follows –

(i) If there are n non terminals then the no of states of the automata will be n + 1, it means that there will be one state for each non terminal and one more state will be there, which is the final state of automata.

The state which corresponds to the start symbol of the grammar willl be the initial state of the automata. If L{G} contains $\in$ then make start state as the final state.

(ii) The transition in the automata can be obtained as follows –

if $A \to \alpha C$,

then, make, $\delta(A, \alpha) = C$

and if $A \to \alpha$

then, make, $\delta(A, \alpha)$ = final state

Q.18. Describe regular expressions. Also, give the formal definition of a regular expression.

Or

Write short note on regular expressions. (R.G.P.V., Dec. 2007, June 2010)

Or

Define regular expressions. ***(R.G.P.V., Dec. 2016)***

Ans. The languages accepted by finite automata are easily described by simple expressions called regular expressions. Regular expressions are useful for representing certain sets of strings in an algebraic fashion. This notation of regular expression involves a combination of strings of symbols from some alphabet over Σ, parenthesis, and the operators such as + and *.

Formal Definition of a Regular Expression – Regular expressions are constructed from primitive constituents by repeatedly applying certain recursive rules. This is much similar to the way the familiar arithmetic expressions are constructed. The regular expressions over Σ and the sets that they denote are defined recursively as follows –

(i) Any terminal symbol (i.e., an element of Σ), $\in$, ϕ, are regular expressions. These are called ***primitive regular expressions***

(ii) The union of two regular expressions R_1 and R_2, written as $R_1 + R_2$, is also a regular expression

(iii) The concatenation of two regular expressions R_1 and R_2, written as R_1R_2, is also a regular expression

(iv) The iteration (or closure) of a regular expression R, written as R^*, is also a regular expression

(v) If R is a regular expression, then (R) is also a regular expression.

In the absence of parenthesis, we have the hierarchy of operations as follows – iteration (closure), concatenation, and union. i.e., in evaluating a regular expression involving various operations, we perform iteration first, then concatenation, and finally union.

Q.19. Enlist the identities for regular expressions.

Ans. Two regular expressions, say P and Q, are equivalent (i.e., $P = Q$) if P and Q represent the same set of strings.

Following are the identities for regular expressions. These are useful for simplifying regular expressions –

(i) $\phi + R = R$ (ii) $\phi R = R\phi = \phi$

(iii) $\in R = R\in = R$ (iv) $\in^* = \in$ and $\phi^* = \in$

(v) $R + R = R$ (vi) $R^*R^* = R^*$

(vii) $RR^* = R^*R$ (viii) $(R^*)^* = R^*$

(ix) $\in + RR^* = R^* = \in + R^*R$ (x) $(PQ)^* P = P(QP)^*$

(xi) $(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$

(xii) $(P + Q) R = PR + QR$ and $R(P + Q) = RP + RQ$.

('Set P' means that the set represented by the regular expression P)

Q.20. Define the language associated with regular expressions.

Ans. Regular expressions can be used to describe some simple languages. If $\boldsymbol{R}$ is a regular expression, we will let $L(R)$ denote the language associated with $\boldsymbol{R}$. This language is defined as follows –

The language $L(R)$ denoted by any regular expression R is defined by the following rules –

(i) ϕ is a regular expression, denoting the empty set

(ii) $\in$ is a regular expression, denoting $\{\in\}$

(iii) For every $a \in \Sigma$, $\boldsymbol{a}$ is a regular expression denoting $\{a\}$.

If $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ are regular expressions, then –

(iv) $L(R_1 + R_2) = L(R_1) \cup L(R_2)$ (v) $L(R_1 R_2) = L(R_1) L(R_2)$

(vi) $L((R_1)) = L(R_1)$ (vii) $L(R_1^*) = (L(R_1))^*$.

The last four rules are used to reduce $L(R)$ to simpler components recursively. While the first three are the termination conditions for this recursion.

For example, we can exhibit the language $L(a^*(a + b))$ in set notion as follows –

$$\begin{aligned} L(a^* (a + b)) &= L(a^*)\, L(a + b) \\ &= (L(a))^* \, (L(a) \cup L(b)) \\ &= \{\in, a, aa, ...\} \; \{a, b\} \\ &= \{a, aa, aaa \; ..... \; , b, ab, aab, \; .....\} \end{aligned}$$

But, there is a problem with rules (iv) to (vii) in the definition. They define a language precisely if $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ are given, but there may be some ambiguity in dividing a complicated expression into parts. For example, take the regular expression $\boldsymbol{ab + c}$, made up of $\boldsymbol{R_1 = ab}$ and $\boldsymbol{R_2 = c}$. In this case, we find $L(ab + c) = \{ab, c\}$. But, there is nothing in the definition to stop us from taking $\boldsymbol{R_1 = a}$ and $\boldsymbol{R_2 = b + c}$. Now, we get a different result, $L(ab + c) = \{ab, ac\}$. To get overcome this, we require that all expressions be fully parenthesized, but this makes the result cumbersome. Thus, we use a convention familiar from mathematics and programming languages. We can make a set of precedence rules for evaluation in which star-closure (*) precedes concatenation and concatenation precedes union.

Q.21. State and prove the theorem describing the relation between transition systems and regular expressions.

Ans. The languages accepted by finite automata are precisely the languages denoted by regular expressions. So that there is an NFA with $\in$-transitions denoting the same language.

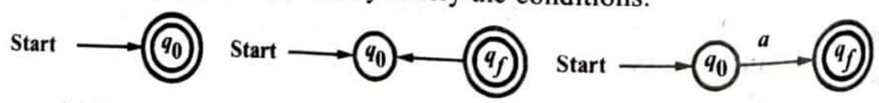
The following theorem describes the relation between transition systems and regular expressions –

Theorem. Let $\boldsymbol{R}$ be a regular expression. Then there exists an NFA with $\in$-transitions that accepts $L(R)$.

Proof. We show by induction on the number of operators in the regular expression $\boldsymbol{R}$ that there is an NFA with $\in$-transitions, having one final state and no transitions out of this final state, such that

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

Basis – The expression $\boldsymbol{R}$ must be $\in$, ϕ, or a for some a in Σ. The NFAs in fig. 2.32 (a), (b) and (c) clearly satisfy the conditions.



(a) $R = \in$ *(b) $R = \phi$* *(c) $R = a$*

Fig. 2.32 Transition Systems Recognizing Elementary Regular Sets

Induction Step – Assume that the theorem is true for regular expressions with fewer than i operators, $i \geq 1$. Let $\boldsymbol{R}$ have i operators. There are three cases depending on the form of $\boldsymbol{R}$.

(i) Case 1 – For union, i.e., $\boldsymbol{R} = \boldsymbol{R_1} + \boldsymbol{R_2}$. Both $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ must have fewer than i operators. Thus there are NFAs $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$ with $L(M_1) = L(R_1)$ and $L(M_2) = L(R_2)$. Since, we may rename states of an NFA at will we may assume Q_1 and Q_2 are disjoint. Let q_0 be a new initial state and f_0 a new final state. Then we construct

$$M = (Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma_1 \cup \Sigma_2, \delta, q_0, \{f_0\})$$

where δ is defined by,

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

(b) $\delta(q,a) = \delta_1(q,a)$ for q is $Q_1 - \{f_1\}$ and a in $\Sigma_1 \cup \{\in\}$

(c) $\delta(q,a) = \delta_2(q,a)$ for q is $Q_2 - \{f_2\}$ and a in $\Sigma_2 \cup \{\in\}$

(d) $\delta(f_1, \in) = \delta_1(f_2, \in) = \{f_0\}$.

Recall by the inductive hypothesis that there are no transitions out of f_1 or f_2 in M_1 or M_2. Thus all the moves of M_1 and M_2 are present in M. The construction of M is depicted in fig. 2.33. It follows that there is a path labelled x in M from q_0 to f_0 if and only if there is a path labelled x in M_1 from q_1 to f_1 or a path in M_2 from q_2 to f_2. Hence $L(M) = L(M_1) \cup L(M_2)$ as desired.

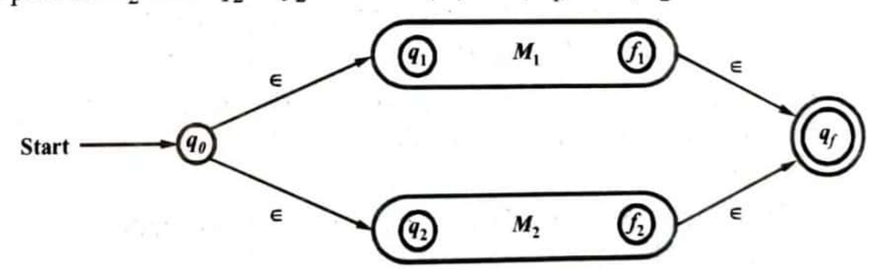


Fig. 2.33 Transition System Recognizing $R = R_1 + R_2$

(ii) Case 2 – For concatenation i.e., $\boldsymbol{R} = \boldsymbol{R_1 R_2}$. Let M_1 and M_2 be as in case 1 and then we construct

$$M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, \{q_1\}, \{f_2\})$$

where δ is given by,

(a) $\delta(q, a) = \delta_1(q, a)$ for q in $Q_1 - \{f_1\}$ and a in $\Sigma_1 \cup \{\in\}$

(b) $\delta(f_1, \in) = q_2$

(c) $\delta(q, a) = \delta_2(q, a)$ for q in Q_2 and a in $\Sigma_2 \cup \{\in\}$

The construction of M is depicted in fig. 2.34. Every path in M from q_1 to f_2 is a path labelled by some string x from q_1 to f_1, followed by the edge from f_1 to q_2 labelled $\in$, followed by a path labelled by some string y from q_2 to f_2. Thus $L(M) = L(M_1)\, L(M_2)$ as desired.

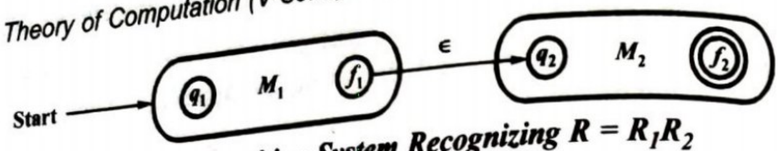


Fig. 2.34 Transition System Recognizing $R = R_1R_2$

(iii) Case 3 – For closure, i.e., $R = R_1^*$. Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ and $L(M_1) = R_1$. Then we construct

$$M = (Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\}$$

where δ is given by –

(a) $\delta(q_0, \in) = \delta(f_1, \in) = \{q_1, f_0\}$

(b) $\delta(q, a) = \delta_1(q, a)$ for q in $Q_1 - \{f_1\}$ and a in $\Sigma_1 \cup \{\in\}$

The construction of M is depicted in fig. 2.35. It follows that there is a path in M from q_0 to f_0 labelled x if and only if we can write $x = x_1, x_2 \ldots x_j$ for some $j \geq 0$ (the case $j = 0$ means $x = \in$) such that each x_i is in $L(M_1) = L_1(M_1)^*$ as desired.

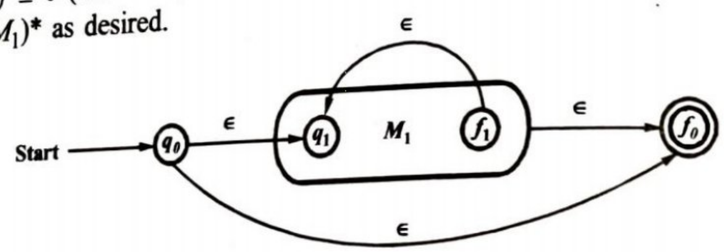


Fig. 2.35 Transition System Recognizing $R = R_1^*$

This proof is in essence an algorithm for converting a regular expression into an equivalent finite automaton.

Q.22. State and prove the theorem describing the relation between a finite automaton M and a regular expression.

Ans. **Statement** – Any set L accepted by a finite automaton M is represented by a regular expression.

Proof. Let $M = (\{q_1 \ldots\ldots q_m\}, T, \delta, q_1, F)$

The construction given above, can be better understood in terms of the state diagram of M. If a string $u \in T^*$ is accepted by M, then there is a path from q_1 to some final state with path value u. So, to each final state (say q_j), there corresponds a subset of T^* having path values of paths from q_1 to q_j. As $L(M)$ is the union of such subsets of T^*, it is enough to represent them by regular expressions. Therefore, the main part of the proof lies in the construction of subsets of path values of paths from q_i to q_j.

Let P_{ij}^k = the set of path values of paths from q_i to q_j whose intermediate vertices lie in $\{q_1 \ldots q_k\}$. We construct P_{ij}^k recursively for $k = 0, 1, \ldots n$, as follows –

$$P_{ij}^0 = \{a \in T : \delta(q_i, a) = q_j\} \quad \ldots(i)$$

$$P_{ij}^0 = \{a \in T : \delta(q_i, a) = q_i\} \cup \{\lambda\} \quad \ldots(ii)$$

$$P_{ij}^k = P_{ik}^{k-1} \left(P_{kk}^{k-1}\right)^* P_{kj}^{k-1} \cup P_{ij}^{k-1} \quad \ldots(iii)$$

The construction can be better understood in terms of the state diagram. P_{ij}^0 simply denotes the set of path values of edges from q_i to q_j. In P_{ij}^0, we include λ in addition to labels of self loops from q_i.

Fig. 2.36 A Path from q_i to q_j

Consider a path from q_i to q_j whose intermediate vertices lie in $\{q_1, \ldots\ldots, q_k\}$. If there is no path through q_k, then its path value lies in P_{ij}^{k-1}. Otherwise, the path passes through q_k possibly more than once. The path can be divided into several paths with path values $u_1, u_2, \ldots u_l$ as shown in fig. 2.36. $u = u_1 u_2 \ldots u_l$. u_l is the path value of the path from q_i to q_k (without passing through q_k, i.e., q_k is not an intermediate vertex). $u_2, \ldots\ldots, u_{l-1}$ are the path values of paths from q_k to q_j itself without passing through q_k. So u_1 is in P_{ik}^{k-1}, $u_2, \ldots\ldots, u_{l-1}$ are in P_{kk}^{k-1} and u_l is in P_{kj}^{k-1}.

We prove that the sets – P_{ij}^0, P_{ii}^0 and P_{ij}^k are represented by regular expressions by induction on k (for all i and j). P_{ij}^0 is a finite subset of T, say $\{a_1, \ldots\ldots, a_t\}$. Then P_{ij}^0 is represented by –

$$P_{ij}^0 = a_1 + a_2 + \ldots\ldots\ldots\ldots + a_t.$$

In the same way, we can construct P_{ii}^0 representing P_{ii}^0. Thus, we have basis for induction.

Let us assume the result for $k - 1$, i.e., P_{ij}^{k-1} is represented by regular expression P_{ij}^{k-1}, for all i and j. From previous equations (iii), we have

$$P_{ij}^k = P_{ik}^{k-1} \left(P_{kk}^{k-1}\right)^* P_{kj}^{k-1} \cap P_{ij}^{k-1}$$

Thus, it is obvious that P_{ij}^k is represented by –

$$P_{ij}^k = P_{ik}^{k-1} \left(P_{kk}^{k-1}\right)^* P_{kj}^{k-1} \cup P_{ij}^{k-1}$$

Therefore, the result is true for all k. By the induction principle, the sets constructed by equations (i), (ii) and (iii) are represented by regular expressions.

As $Q = \{q_1, \ldots\ldots, q_m\}$, P_{1j}^m represents the set of path values of all paths from q_1 to q_j. If $F = \{q_{f_1}, \ldots\ldots, q_{f_n}\}$, then $L(M) = \bigcup_{j=1}^{n} P_{1f_j}^m$. So $L(M)$ is represented by the regular expression, $\bigcup_{j=1}^{n} P_{1f_j}^m$. Thus, $L = L(M)$ is represented by a regular expression.

Q.23. Write a two step procedure of constructing a finite automaton equivalent to a regular expression.

Ans. This method for the construction of a finite automaton equivalent to a regular expression is known as ***subset method.*** It involves two steps as follows –

(i) Step 1 – Construct a transition graph (i.e., transition system) equivalent to the given regular expression using $\in$-moves.

(ii) Step 2 – Construct the transition table for the graph obtained in step 1. Now construct the equivalent DFA. Reduce the number of states, if possible.

Q.24. State and prove Arden's theorem.

Ans. It is very much useful in simplifying regular expressions.

It can be given as follows –

Let P and Q be two regular expressions over Σ. If P does not contain Λ, then the following equation –

$$R = Q + RP \qquad \text{....(i)}$$

has a unique solution (i.e., one and only one solution) given by –

$$R + QP^*$$

Proof –

$$Q + (QP^*)\,P = Q\,(\in + P^*P)$$
$$= QP^* \qquad \text{(by identity (ix))}$$

Hence equation (i) is satisfied when $R = QP^*$. It means that $R = QP^*$ is a solution of equation (i).

Now, to prove uniqueness, take equation (i). Here, we replace R by $Q + RP$ on the R.H.S., we get

$$Q + RP = Q + (Q + RP)\,P$$
$$= Q + QP + RPP$$
$$= Q + QP + RP^2$$
$$= Q + QP + QP^2 + \ldots\ldots\ldots\ldots + QP^i + RP^{i+1}$$
$$= Q\,(\in + P + P^2 + \ldots\ldots\ldots\ldots P^i) + RP^{i+1}$$

From equation (i),

$$R = Q\,(\in + P + P^2 + \ldots + P^i) + RP^{i+1} \quad \text{for } i \geq 0 \qquad \text{...(ii)}$$

Now we show that any solution of equation (i) is equivalent to QP^*. Suppose R satisfies equation (i), then it satisfies equation (ii). Let u be a string of length i in the set R. Then u belongs to the set $Q\,(\in + P + P^2 + \ldots\ldots\ldots\ldots + P^i) + RP^{i+1}$. As P does not have $\in$, RP^{i+1} has no string of length less than $i+1$ and so u is not in the set RP^{i+1}. This means u belongs to the set $Q(\in + P + P^2 + \ldots\ldots\ldots\ldots + P^i)$, and hence to QP^*.

Consider a string u in the set QP^*. Then u is in the set QP^k for some $k \geq 0$ and hence in $Q(\in + P + P^2 + \ldots\ldots\ldots\ldots + P^k)$. So u is on the R.H.S. of equation (ii). Therefore u is in R [L.H.S. of equation (ii)]. Thus R and QP^* represent the same set. This proves the uniqueness of the solution of equation (i).

Q.25. How can you test the equivalence of two regular expressions ?

Ans. Let we have two regular expressions P and Q. P and Q are equivalent if and only if they represent the same set. Also, they are equivalent if and only if the corresponding finite automata are equivalent.

The equivalence of P and Q can be proves by three ways –

(i) Prove that the sets P and Q are the same. (For non-equivalence, find a string in one set but not in the other).

(ii) Use the identities to prove the equivalence of P and Q

(iii) Construct the corresponding finite automata M and M' and prove that they are equivalent. (For non-equivalence, prove that M and M' are not equivalent)

One of the above three methods can be chosen according to the problem.

Q.26. Explain the comparison method to test the equivalence of two finite automata over Σ.

Ans. Two finite automata over Σ are equivalent, if they accept the same set of strings over Σ. When two finite automata are not equivalent, there is some string u over Σ satisfying the following –

One automaton reaches a final state on application of u, whereas the other reaches a nonfinal state.

Following is the method called ***comparison method***, to test the equivalence of two finite automata over Σ –

Comparison Method – Let M and M' be two finite automata over Σ. Construct comparison table consisting of $n + 1$ columns, where n is the number of input symbols. First column consists of pairs of vertices of the form (q, q') where $q \in M$ and $q' \in M'$. If (q, q') appears in some row of first column, then the corresponding entry in the a-column (where $a \in \Sigma$) is (q_a, q_a'), where q_a *and* q_a' are reachable from q and q', respectively, on application of a (i.e., by a-paths).

The comparison table is constructed by starting with the pair of initial vertices q_{in}, q_{in}' of M and M' in the first column. The first elements of the subsequent columns are (q_a, q_a'), where q_a and q_a' are reachable by a-paths from q_{in} and q_{in}'. Now repeat the construction by considering the pairs in the second and subsequent columns which are not in the first column.

The row-wise construction is repeated. There may be two cases –

(i) Case 1 – If we reach a pair (q, q') such that q is a final state of M, and q' is a non-final state of M', or vice-versa, we terminate the construction process and conclude that M and M' are not equivalent.

(ii) Case 2 – In this case, the construction is terminated when no new element appears in the second and subsequent columns those are not in the first column. In other words, when all the elements in the second and subsequent columns appear in the first column. Thus we conclude that M and M' are equivale

Q.27. Explain Myhill-Nerode theorem. **(R.G.P.V., June 2011)**

Or

State and prove Myhill-Nerode theorem. **(R.G.P.V., June 2003, Dec. 2007, 2012)**

Or

Write short note on Myhill-Nerode theorem. **(R.G.P.V., June 2010)**

Or

Write the Myhill-Nerode theorem. **(R.G.P.V., Dec. 2002)**

Or

State the Myhill-Nerode theorem and its applications. **(R.G.P.V., June 2015)**

Or

Write and explain Myhill-Nerode theorem. **(R.G.P.V., June 2016, Dec. 2017)**

Or

Explain Myhill-Nerode theorem with example. **(R.G.P.V., Dec. 2016)**

Ans. An equivalence relation R such that xRy implies $xzRyz$ is said to be ***right invarient*** (with respect to concatenation). We see that every finite automaton induces a right invarient equivalence relation, defined as R_m was defined on its set of input strings. This result is formalized in the following theorem.

Theorem. The following three statements are equivalent –

(i) The set $L \subseteq \Sigma^*$ is accepted by some finite automaton

(ii) L is the union of some of the equivalence classes of a right invarient equivalence relation of finite index

(iii) Let equivalence relation R_L be defined by – xR_Ly if and only if for all z in Σ^*, xz is in L exactly when yz is in L. Then R_L is of finite index.

Proof. (i) → (ii) Assume that L is accepted by some DFA $M = (Q, \Sigma, \delta, q_0, F)$. Let R_m be the equivalence relation xR_my if and only if $\delta(q_0, x) = \delta(q_0, y)$. R_m is right invarient since, for any z, if $\delta(q_0, x) = \delta(q_0, y)$, then $\delta(q_0, xz) = \delta(q_0, yz)$. The index of R_m is finite, since the index is at most the number of states in Q. Moreover, L is the union of those equivalence classes that include a string x such that $\delta(q_0, x)$ is in F, i.e., the equivalent classes corresponding to final states.

(ii) → (iii) We show that any equivalence relation E satisfying (ii) is a refinement of R_L, i.e., every equivalence class of E is entirely contained in some equivalence class of R_L. Thus, the index of R_L cannot be greater than the index of E and so is finite. Suppose that xEy. Then as E is the right invarient, for each z in Σ^*, $xzEyz$, and thus yz is in L if and only if xz is in L. Thus xR_Ly, and hence the equivalence class of x in R_L. Now, we conclude that each equivalence class of E is contained within some equivalence class of R_L.

(iii) → (i) First, we must show that R_L is right invarient. Suppose xR_Ly, and let u be in Σ^*. We have to prove that xuR_Lyu, i.e., for any z, xuz is in L exactly when yuz is in L. But since xR_Ly, by the definition of R_L we know that for any v, xv is in L exactly when yv is in L. Let $v = wz$ to prove that R_L right invarient.

Now let Q' be the finite set of equivalence classes of R_L and $[x]$ the element of Q' containing x. Define $\delta'([x], a) = [xa]$. The definition is consistent, since R_L is right invarient. Had we chosen y instead of x from the equivalence class $[x]$, we would have obtained $\delta'([x], a) = [ya]$. But xR_Ly, so xz is in L exactly, when yz is in L. Particularly, if $z = az'$, xaz' is in L exactly when yaz' is in L, so xaR_Lya, and $[xa] = [ya]$. Let $q_0' = [\in]$ and let $F' = \{[x] \mid x \text{ is in } L\}$. The finite automaton $M' = (Q', \Sigma, \delta', q_0', F')$ accepts L, since $\delta'(q_0, x) = [x]$, and thus x is in $L(M')$ if and only if $[x]$ is in F'.

Q.28. Give the minimization algorithm to minimize the finite automata with the help of Myhill-Nerode theorem.

Ans. The Myhill-Nerode theorem has the implication that there is an essentially unique minimum state DFA for every regular set.

Theorem – The minimum state automaton accepting a set L is unique upto an isomorphism (i.e., renaming of the states) and is given by M'.

Proof – In the proof of Myhill-Nerode theorem, we saw that any DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepting L defines an equivalence relation that is a refinement of R_L. Therefore, the number of states of M is greater than or equal to the number of states of M'. If equality holds, then each of the states of M can be identified with one of the states of M'. i.e., let q be a state of M. There must be some x in Σ^*, such that $\delta(q_0, x) = q$, otherwise q could be removed from Q, and a smaller automation found. Identify q with the state $\delta'(q_0', x)$, of M'. This identification will be consistent. If $\delta(q_0, x) = \delta(q_0, y) = q$, then, by the proof of Myhill-Nerode theorem, x and y are in the same equivalence class of R_L. Thus, $\delta'(q_0', x) = \delta'(q_0', y)$.

Minimization Algorithm – There is a simple method for finding the minimum state DFA M' equivalent to a given DFA $M = (Q, \Sigma, \delta, q_0, F)$. Let $\equiv$ be the equivalence relation on the states of M such that $p \equiv q$ if and only if for each input string x, $\delta(p, x)$ is an accepting state if and only if $\delta(q, x)$ is an accepting state. It is observed that there is an isomorphism between those equivalence classes of $\equiv$ that contain a state reachable from q_0 by some input string and the states of the minimum state FA M'. Thus the states of M' may be identified with these classes.

Rather than give a formal algorithm for computing the equivalence classes of $\equiv$ we first work through an example first some terminology is needed. If $p \equiv q$, we say p is equivalent to q. We say that p is distinguishable from q if there exists an x such that $\delta(p, x)$ is in F and $\delta(q, x)$ is not, or vice-versa.

Q.29. Explain Myhill-Nerode method of minimization.
(R.G.P.V., June 2009)

Ans. Refer to Q.28.

Q.30. State and prove the pumping lemma for regular sets.
(R.G.P.V., June 2010, 2017)

Or

State and prove the pumping lemma theory of regular language.
(R.G.P.V., June 2009)

Or

Write short note on pumping lemma for regular language.
(R.G.P.V., Dec. 2004)

Ans. Pumping lemma gives a method of pumping (or generating) many input strings from a given string. It also gives a necessary condition for an input string to belong to a regular set. Hence, it can be used to show that certain sets are not regular. It can be given as follows –

Pumping Lemma – Let $M(Q, \Sigma, \delta, q_0, F)$ be a finite automaton with n states. Let L be the regular set accepted by M. Let $u \in L$ and $|u| \geq m$. If $m \geq n$, then there exists x, y, z such that –

$$u = xyz, y \neq \in, \text{ and } xy^iz \in L, \quad \text{for each } i \geq 0.$$

Proof – Let

$$u = a_1a_2a_3 \ldots a_m, \qquad m \geq n$$

$$\delta(q_0, a_1a_2a_3 \ldots a_i) = q_i, \qquad \text{for } i = 1, 2, \ldots, m$$

$$Q_1 = \{q_0, q_1, q_2 \ldots, q_m\}$$

i.e., Q_1 is the sequence of states in the path with path value $u = a_1a_2a_3 \ldots a_m$. Since there are only n distinct states, at least two states in Q_1 must coincide. Among various repeated states pairs, take the first pair. Let us take them as q_j and q_k ($q_j = q_k$). Then j and k satisfy the condition $0 \leq j < k \leq n$.

The string u can be divided into three substrings $a_1a_2 \ldots, a_j, a_{j+1} \ldots a_k$ and $a_{k+1} \ldots a_m$. Let x, y, z, represent these strings respectively. Now, as $k \leq n$, $|xy| \leq n$ and $u = xyz$. The path with path value u in the transition diagram of M is shown in fig. 2.37.

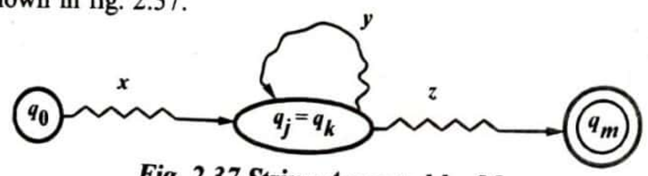


Fig. 2.37 String Accepted by M

The automaton M starts from the initial state, i.e., q_0. On applying the string x, it reaches q_j (which is equal to q_k). On applying the string y, it comes back to q_j. So after application of y^i for each $i \geq 0$, the automaton is in the same state, i.e., q_j. On applying z, it reaches q_m, i.e., a final state. Hence $xy^iz \in L$. Since every state in Q_1 is obtained by applying an input symbol, $y \neq \in$.

Note that the decomposition is valid only for strings of length greater than or equal to the number of states. For such a string $u = xyz$, we can iterate (or repeat) the substring y in xyz as many times as we want and get strings of the form xy^iz, which are longer than xyz and are in L. By considering the path from q_0 *to* q_k and then from q_k to q_m (without going through the loop), we obtain a path ending in a final state with path value xz. This corresponds to the case when $i = 0$.

Q.31. Explain applications of pumping lemma.
(R.G.P.V., Dec. 2002, May 2018)

Ans. Pumping lemma can be used to prove that certain sets are not regular. Following are the steps needed to prove it –

(i) Step 1 – Suppose L is regular. Let n be the number of states in the corresponding finite automaton.

(ii) Step 2 – Select a string u such that $|u| \geq n$. Using pumping lemma, write $u = xyz$, with $|xy| \leq n$ and $|y| > 0$.

(iii) Step 3 – Find an appropriate integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

Note that, the crucial part of it is to find i such that $xy^iz \notin L$. In some cases, we prove $xy^iz \notin L$ by considering $|xy^iz|$. While in some cases we may have to use the structure of strings in L.

Q.32. Explain the closure properties of regular sets.

Or

Prove that the regular sets are closed under intersection.
(R.G.P.V., June 2003, Dec. 2003, June 2004)

Or

Show that regular languages are closed under complement and subtraction operation. ***(R.G.P.V., Dec. 2002)***

Or

Write the closure property of regular expression.(R.G.P.V., June 2017)

Ans. If L_1 and L_2 are regular languages, then there exist regular expression $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ such that $L_1 = L(R_1)$ and $L_2 = L(R_2)$. By definition, $\boldsymbol{R_1} + \boldsymbol{R_2}$, $\boldsymbol{R_1R_2}$, and $\boldsymbol{R_1}^*$ are regular expressions denoting the languages $L_1 \cup L_2$, L_1L_2, and L_1^*, respectively. Thus, closure under union, concatenation, and star-closure is immediate.

Following theorems prove the other closure properties of regular languages –

Theorem 1. If L is regular then transpose L^T is also regular.

Proof. Every regular expression is recognized by transition system. A transition system M can be converted into finite accepter (FA), accepting the same set as M. Any set accepted by FA is represented by a regular expressic A subset L of Σ^* is regular set if and only if it is accepted by FA, so L is regul

We can construct a FA $M = (Q, \Sigma, \delta, q_0, F)$ such that

$$T(M) = L$$

We construct a transition system M' by starting with the state diagram of M and reversing the direction of the directed edges. The set of initial states of M' is defined as the set F, and q_0 is defined as the (only) final state of M', i.e., $M' = (Q, \Sigma, \delta', F, \{q_0\})$.

If $w \in T(M)$, we have a path from q_0, to some final state in F with path value w. By 'reversing the edges', we get a pair in M' from some final state in F to q_0. Its path value is w^T. So $w^T \in T(M')$. In a similar way, we can see that if $w_1 \in T(M')$, then $w_1^T \in T(M)$. Thus from the state diagram, it is easy to see that $T(M') = T(M)^T$. We can prove rigorously that $w \in T(M)$ if and only if $w^T \in T(M')$ by induction on $|w|$. So $T(M)^T = T(M')$ we know that, a subset L of Σ^* is a regular set if and only if it is recognized by a transition system. So $T(M')$ is regular, i.e., $T(M)^T$ is regular.

Theorem 2. If L is a regular set over Σ, then the complement of L, i.e., $\Sigma^* - L$ is also regular over Σ.

Proof. As L is regular if it is accepted by FA. We can construct a DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepting L, i.e., $L = T(M)$.

We construct another DFA $M' = (Q, \Sigma, \delta, q_0, F')$ by defining $F' = Q - F$, i.e., M and M' differ only in their final states. A final state of M' is a nonfinal state of M and vice versa. The state diagrams of M and M' are the same except for the final states.

$w \in T(M')$ if and only if $\delta(q_0, w) \in F' = Q - F$, i.e., if and only if $w \notin L$. This proves

$$T(M') = \Sigma^* - L.$$

Theorem 3. If L_1 and L_2 are regular sets over Σ_1 then $L_1 \cap L_2$ is also regular over Σ.

Proof. For closure under intersection, we start with DeMorgan's law

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

for any languages L_1 and L_2. Now, if L_1 and L_2 are regular, then by closure under complementation, so are $\overline{L_1}$ and $\overline{L_2}$. Using closure under union, we next get that $\overline{L_1} \cup \overline{L_2}$ is regular. Using closure under complementation once more, we see that $\overline{\overline{L_1} \cup \overline{L_2}}$ is regular, i.e., $L_1 \cap L_2$ is regular.

Theorem 4. The class of regular sets is closed under substitution.

Proof. Let $R \subseteq \Sigma^*$ be a regular set and for each a in Σ let $R_a \subseteq \Delta^*$ be a regular set. Let $f : \Sigma \to \Delta^*$ be the substitution defined by $f(a) = R_a$. Select regular expressions denoting R and each R_a. Replace each occurrence of the symbol a in the regular expression for R by the regular expression for R_a. To prove that the resulting regular expression denotes $f(R)$, observe that the substitution of a union, product, or closure is the union, product or closure of the substitution. A simple induction on the number of operators in the regular expression completes the proof.

Theorem 5. The class of regular sets is closed under homomorphisms and inverse homomorphisms.

Proof. Closure under homomorphisms follows immediately from closure under substitution, since every homomorphism is a substitution, in which $h(a)$ has one member.

To show closure under inverse homomorphism, let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA accepting L, and let h be a homomorphism from Δ to Σ^*. We construct a DFA M' that accepts $h^{-1}(L)$ by reading symbol a in Δ and simulating M on $h(a)$. Formally, let $M' = (Q, \Delta, \delta', q_0, F)$ and define $\delta'(q, a)$, for q in Q and a in Δ to be $\delta(q, h(a))$. Note that $h(a)$ may be a long string, or $\in$, but δ is defined on all strings by extension. It is easy to show by induction on $|x|$ that $\delta'(q_0, x) = \delta(q_0, h(x))$. Therefore M' accepts x if and only if M accepts $h(x)$. That is, $L(M') = h^{-1}(L(M))$.

Theorem 6. The class of regular sets is closed under quotient with arbitrary sets.

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton accepting some regular set R and let L be an arbitrary language. The quotient R/L is accepted by a finite automaton $M' = (Q, \Sigma, \delta, q_0, F')$, which behaves like M except that the final states of M' are all states q of M such that there exists y in L for which $\delta(q, y)$ is in F. Then $\delta(q_0, x)$ is in F' if and only if there exists y such that $\delta(q_0, xy)$ is in F. Thus, M' accepts R/L.

Q.33. What do you mean by closure properties of regular languages ? Define some important closer properties. ***(R.G.P.V., Nov. 2019)***

Ans. Refer to Q.32.

Q.34. Prove the following two properties –

(i) Regular set are closed over the intersection operation

(ii) Let A and B are context free language over the same alphabet Σ then, prove that $A \cup B$ (A union B) is also context free.

(R.G.P.V., Nov. 2019)

Ans. (i) Refer to Q.32, Theorem 3.

(ii) The proof is constructive – If we start with context-free grammars

$$G_1 = (V_1, \Sigma, S_1, P_1)$$

and

$$G_2 = (V_2, \Sigma, S_2, P_2)$$

Generating A and B respectively.

A grammar $G = (V, \Sigma, S, P)$ generating $A \cup B$. First we rename the elements of V_2 if necessary so that $V_1 \cap V_2 = \phi$, and then we define

$$V = V_1 \cup V_2 \cup \{S\}$$

where S is a new symbol not in V_1 or V_2.

Then we let

$$P = P_1 \cup P_2 \cup \{S \to S_1 | S_2\}$$

On the one hand, if $x \in A = L(G_1)$, then $S \to^* x$ in the grammar G because we can start with the production $S \to S_1$ and continue with the derivation of x in G_1, and similarly for $x \in B$. Therefore

$$A \cup B \subseteq L(G)$$

On the other hand, if x is derivable from S in G, the first step in any derivation must be

$$S \Rightarrow S_1 \text{ or } S \Rightarrow S_2$$

In the first case, all subsequent productions used must be productions in G_1 because no variables in V_2 are involved and thus $x \in A$, in the second case $x \in B$.

Therefore,

$$L(G) \subseteq A \cup B$$

Q.35. Define two-way deterministic finite automaton (2DFA).

Or

Write short note on two-way finite automata. ***(R.G.P.V., Dec. 2002, June 2010)***

Or

Define two-way finite automata. ***(R.G.P.V., Dec. 2016)***

Ans. The finite automaton as a control unit that read a tape, moving one square right at each move. We added non-determinism to the model, which allow many "copies" of the control unit to exist and scan the tape simultaneously. Next we added $\in$-transitions, which allowed change of state without reading the input symbol or moving the tape head. Another extension is to allow the tape head the ability to move left as well as right. Such a finite automaton is called a ***two-way finite automaton or 2-FA.*** It accepts an input string if it moves the tape head off the right end of the tape, at the same time entering an accepting state. We shall see that even this generalization does not increase the power of the finite automaton; two-way FA accepts only regular sets. We give a proof only for a special case of a two-way FA, that is deterministic and whose tape head must move left or right, i.e., not stationary, at each move.

A two-way deterministic finite automaton or 2DFA is a quintuple

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

where, Q = A finite set of internal state

Σ = A finite set of symbols called the input alphabets

q_0 = The initial state

F = A set of final state as before and

δ = Transition function which maps $Q \times \Sigma$ to $Q \times \{L, R\}$.

If $\delta(q, a) = (p, L)$ then in state q, scanning input symbol a, the 2DFA enters state p and moves its head left one square. If $\delta(q, a) = (p, R)$, the 2DFA enters state p and moves its head right one square.

Q.36. Define two-way finite automata with example. (R.G.P.V., Dec. 2017)

Ans. **Two-way Finite Automata** – Refer to Q.35.

Example – Let M be the two-way deterministic finite automata, which behaves as follows, its starting state is q_0. Now according to M, it repeat a cycle of moves where the tape head moves right until two 1's have been encountered or the left until encountering 0, at which the state is q_0, it is reentered and the cycle repeated M, which has three states, all of which are final, and the δ is given in the table.

Q \ Σ	0	1
q_0	q_0, R	q_1, R
q_1	q_1, R	q_2, L
q_2	q_0, R	q_2, L

and input string is **101001**

The initial state is q_0, the first ID is

$$q_0\ 101001$$

The second 'ID' is obtained through the symbol to the immediate 'right' of the state q_0 in the first ID is a 1 and $\delta(q_0, 1)$ is (q_1, R) thus the second ID is q_1, **01001** continue in this fashion we get the result.

$$q_0\ 101001$$

$$\xrightarrow{q_1,\ 1,\ R}$$

$$\vdash 1\ q_1\ 01001$$

$$\xrightarrow{q_1,\ R}$$

$$\vdash 10\ q_1\ 1001 \quad \text{move till we get one 'zero'}$$

$$\xrightarrow{q_2,\ L}$$

$$\vdash 1\ q_2\ 01001$$

$$\xrightarrow{q_0,\ R}$$

$$10\ q_0\ 1001$$

$$\xrightarrow{q_1,\ R}$$

$$\vdash 101\ q_1\ 001$$

$$\xrightarrow{q_1,\ R}$$

$$1010\ q_1\ 01$$

$$\xrightarrow{q_1,\ R}$$

$$\vdash 10100\ q_1\ 1$$

$$\xleftarrow{q_2,\ L}$$

$$\vdash 1010\ q_2\ 01$$

$$\xrightarrow{q_0,\ R}$$

$$\vdash 10100\ q_0\ 1$$

$$\xrightarrow{q_1,\ R}$$

$$\vdash 101001\ q_1$$

NUMERICAL PROBLEMS

Prob.16. Write regular expression for the following language –
The set of strings over alphabet {a, b, c} containing at least one a and at least one b. ***(R.G.P.V., Dec. 2008)***

Sol. The regular expression for the set of all strings over $\{a, b, c\}$ containing at least one "a" and at least one "b" is given by

$$\{(a+b+c)^*a(a+b+c)^*b(a+b+c)^*\} + \{(a+b+c)^*b(a+b+c)^*a(a+b+c)^*\}$$

Prob.17. Obtain the regular expression for the following set –
(i) Set of all strings over $\Sigma = (a, b)$ with exactly two a.
(ii) $L = \{a^x \mid x$ is divisible by 3 or 5$\}$
(R.G.P.V., Nov. 2019)

Sol. (i) The regular expression for the set of over $\Sigma = (a, b)$ with exactly two "a" is given by

b* ab* ab*

(ii) (aaa)* + (aaaaa)*

Prob.18. Give english description of the language of the following regular expression –

(0* 1*)* 000 (0 + 1)* ***(R.G.P.V., Dec. 2008)***

Sol. The given regular expression is

(0* 1*)* 000 (0 + 1)*

By using the identity $(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$ for regular expression, the above regular expression can be written as follows –

(0 + 1)* 000 (0 + 1)*

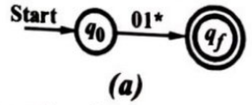
In english, the expression (0 + 1)* 000 (0 + 1)* denotes all strings of 0's and 1's with at least three consecutive 0's.

Prob.19. Convert the following regular expression to NFA* with ∈ transition –

01* ***(R.G.P.V., Dec. 2008)***

Sol. The given regular expression is 01*.

We can express it with the initial state q_0 and final q_f.



(a)

We eliminate the concatenation in the regular expression with the help of state q_1.

(b) ***(c)***

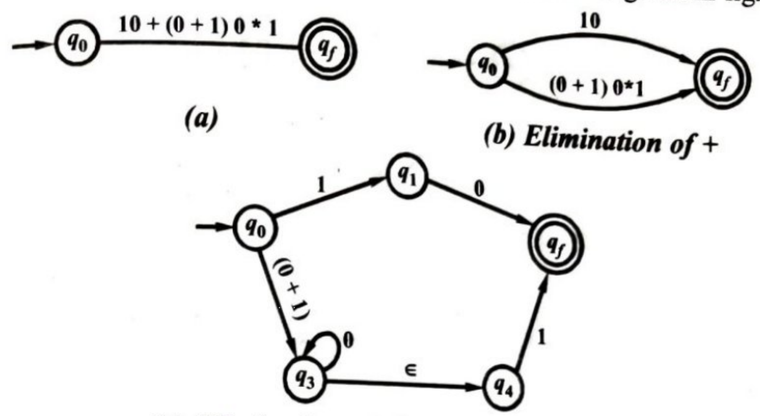
Fig. 2.38

This is the required NFA with ∈ move.

Prob.20. Construct an NFA for the following regular expression –

10 + (0 + 1) 0 * 1 ***(R.G.P.V., Dec. 2007)***

Sol. The NFA is constructed by eliminating the operation +, concatenation and * in successive steps. The step-by-step construction is given in fig. 2.39.



(a) ***(b) Elimination of +***

(c) Elimination of Concatenation and *

Fig. 2.39

Now, we eliminate ∈-move in fig. 2.39 (c) and get fig. 2.40 which gives the NFA equivalent to the given regular expression.

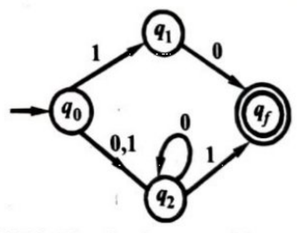


Fig. 2.40 NFA Equivalent to Given Expression

Prob.21. Construct an NFA for the regular expression
(0*(10 + 01)*11)* ***(R.G.P.V., June 2017)***

Sol. The resultant NFA for the regular expression (0*(10 + 01)*11)* is shown in fig. 2.41.

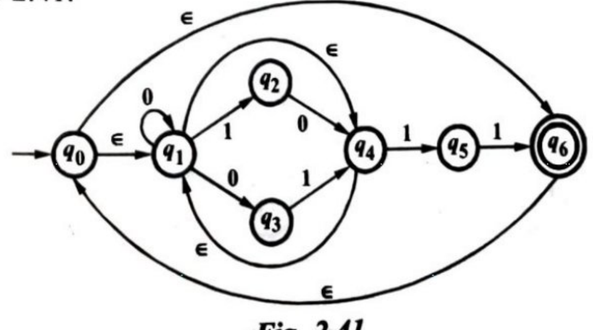
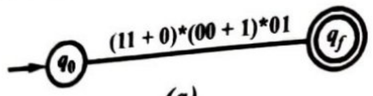


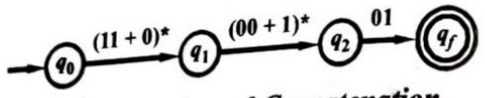
Fig. 2.41

Prob.22. Construct finite automata equivalent to the following regression expression ***(11 + 0)*(00 + 1)*01*** ***(R.G.P.V., Dec. 2006)***

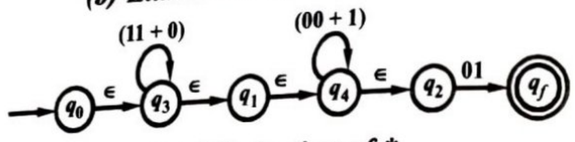
Sol. The finite automata is constructed by eliminating the concatenation, *, and + operations in successive steps. The step-by-step construction is given in fig. 2.42.



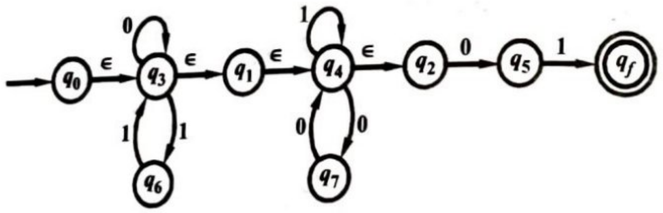
(a)



(b) Elimination of Concatenation



(c) Elimination of *



(d) Equivalent Finite Automata

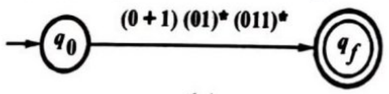
Fig. 2.42

Prob.23. For each of these regular expressions over {0, 1}, draw an NFA with ∈-move recognizing the corresponding language –

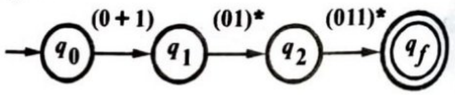
(i) (0 + 1) (01)* (011)* (ii) 010* + 0 (01 + 10)*11 .

(R.G.P.V., Dec. 2005)

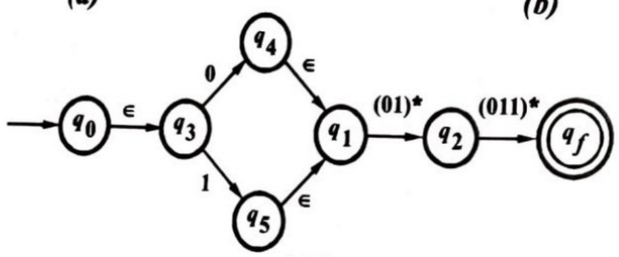
Sol. (i) (0 + 1) (01)* (011)* – The NFA with ∈-moves is constructed by eliminating the operation +, concatenation and * in successive steps. The step-by-step construction is given in fig. 2.43.



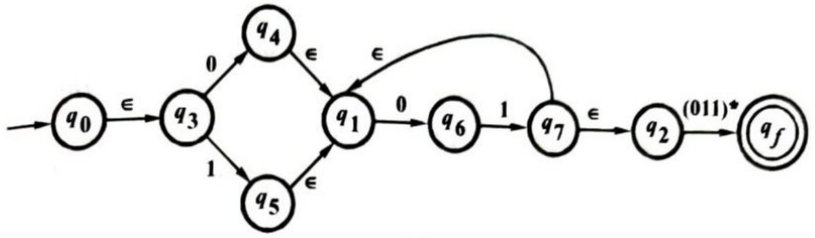
(a)



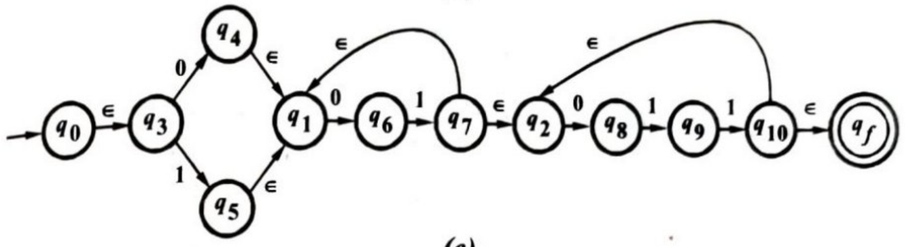
(b)



(c)



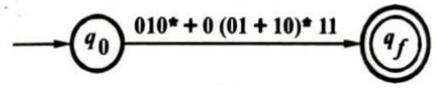
(d)



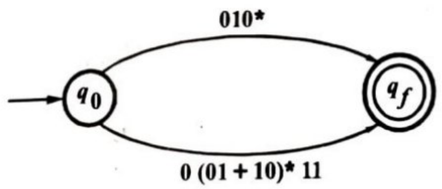
(e)

Fig. 2.43 Construction of NFA with ∈-moves

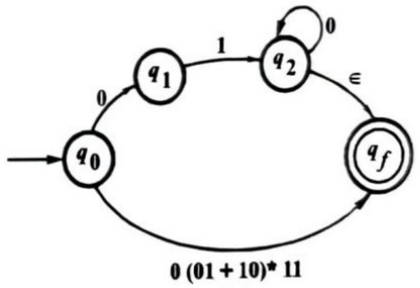
(ii) 010* + 0 (01 + 10)* 11 – The NFA with ∈-moves is constructed by eliminating the operation +, concatenation and * in successive steps. The step-by-step construction is given in fig. 2.44.



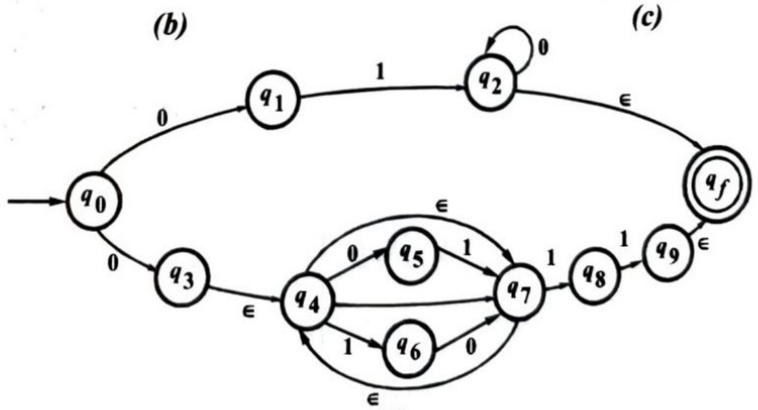
(a)



(b)



(c)



(d)

Fig. 2.44 Construction of NFA with ∈-moves

Prob.24. Construct DFA for the following regular expression –

(i) 0 1 (0 + 1)* 10 (ii) ab* (a + b) a.

(R.G.P.V., June 2007)

Sol. (i) The given regular expression is 01 (0 + 1)*10. we can construct the transition system with ∈-moves. Eliminating ∈-moves we get the NFA accepting the given set of strings. The transition system is given in fig. 2.45.

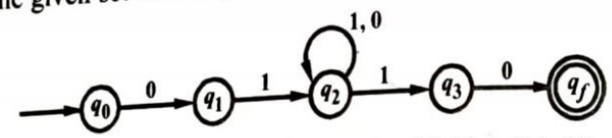


Fig. 2.45 Transition System for 01 (0 + 1)* 10

The successor table is constructed for DFA is given in table 2.20.

Table 2.20

State	*0*	*1*
$[q_0]$	$[q_1]$	ϕ
$[q_1]$	ϕ	$[q_2]$
$[q_2]$	$[q_2]$	$[q_2q_3]$
$[q_2q_3]$	$[q_2q_f]$	$[q_2q_3]$
$[q_2q_f]$	$[q_2]$	$[q_2q_3]$

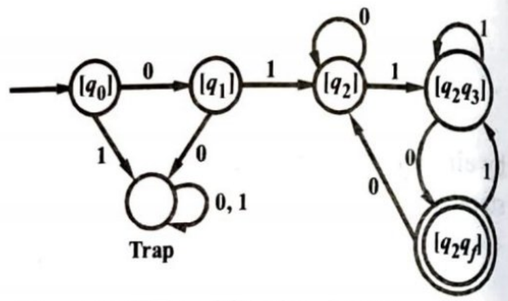


Fig. 2.46 DFA of 01 (0 + 1)*10

The state diagram for the successor table is the required DFA as indicated in fig. 2.46. As q_f is the only final state of NFA, $[q_2q_f]$ is only final state of DFA.

(ii) ab* (a + b) a –

(a) Construction of transition graph eliminating ∈-moves shown in fig. 2.47.

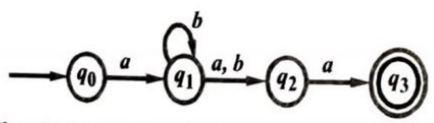


Fig. 2.47 FA Equivalent to ab* (a + b) a

(b) Construction of DFA –

Table 2.21 Transition Table for DFA

State	*a*	*b*
$[q_0]$	$[q_1]$	ϕ
$[q_1]$	$[q_2]$	$[q_1,q_2]$
$[q_2]$	$[q_3]$	ϕ
$[q_3]$	ϕ	ϕ
$[q_1,q_2]$	$[q_2,q_3]$	$[q_1,q_2]$
$[q_2,q_3]$	$[q_3]$	ϕ

As q_3 is the final state of NFA, so $[q_2, q_3]$ and $[q_3]$ are the final states of DFA.

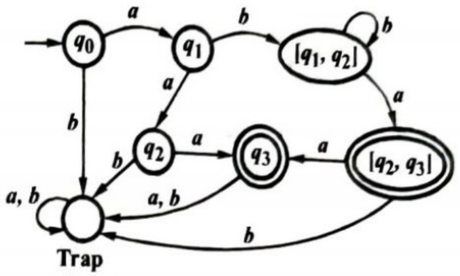


Fig. 2.48 DFA for ab*(a + b) a

Prob.25. Describe the NFA and DFA in brief. Also explain the construction of DFA for the following regular expression –

r = (0 + 1) * (00 + 11) (0 + 1)* ***(R.G.P.V., Nov. 2019)***

Sol. **NFA and DFA** – Refer to Q.1 and Q.2.

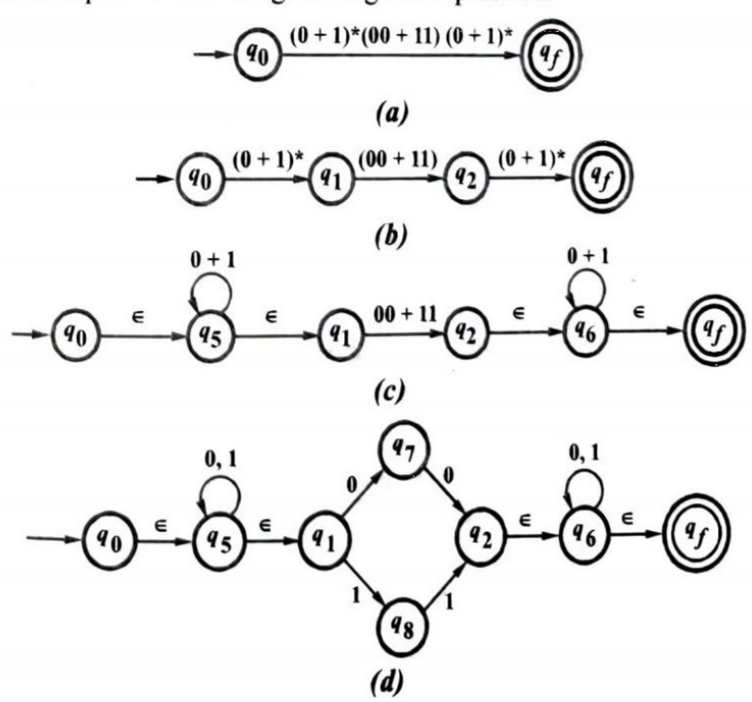
Construction of Transition Graph – The given regular expression is (0 + 1) * (00 + 11) (0 + 1)*. We can construct the transition graph with ∈-moves. Eliminating ∈-moves we get NFA accepting the given set of strings.

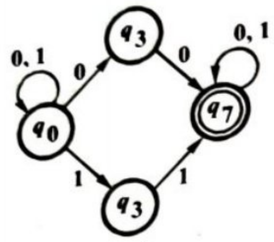
We start with fig. 2.49 (a). We eliminate the concatenations in the given R.E. by introducing new vertices q_1, q_2 and get fig. 2.49 (b).

We eliminate *-operations in fig. 2.49 (b) by introducing two new vertices q_5 and q_6 and ∈-moves as shown in fig. 2.49 (c).

We eliminate concatenations and + in fig. 2.49 (c) and get fig. 2.49 (d).

We eliminate ∈-moves in fig. 2.49 (d) and get fig. 2.49 (e) which gives the NDFA equivalent to the given regular expression.





(e)

*Fig. 2.49 Transition Graph for (0 + 1) * (00 + 11) (0 + 1)**

Construction of DFA – We construct the transition table as given in table 2.22 corresponding to fig. 2.49.

Table 2.22 Transition Table

State	Input	
	0	1
$\rightarrow q_0$	q_0, q_3	q_0, q_4
q_3	q_f	–
q_4	–	q_f
q_f	q_f	q_f

Table 2.23

Q	Q_0	Q_1
$\rightarrow [q_0]$	$[q_0, q_3]$	$[q_0, q_4]$
$[q_0, q_3]$	$[q_0, q_3, q_f]$	$[q_0, q_4]$
$[q_0, q_4]$	$[q_0, q_3]$	$[q_0, q_4, q_f]$
$[q_0, q_3, q_f]$	$[q_0, q_3, q_f]$	$[q_0, q_4, q_f]$
$[q_0, q_4, q_f]$	$[q_0, q_3, q_f]$	$[q_0, q_4, q_f]$

The successor table is constructed for DFA is given in table 2.23.

The state diagram for the successor table is the required DFA as shown in fig. 2.50. As q_f is the only final state of NDFA, hence $[q_0, q_3, q_f)$ and (q_0, q_4, q_f) are the final states of DFA.

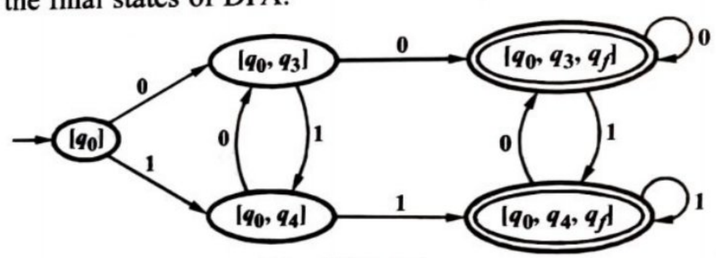
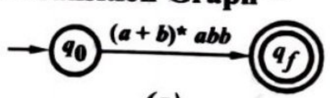


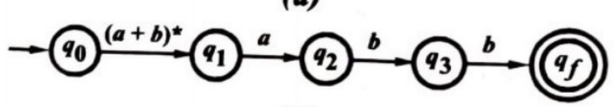
Fig. 2.50 FA

Prob.26. Construct an NFA with ∈-move corresponding to the R.E. (a + b)*abb, convert it to a DFA and minimize it. (R.G.P.V., June 2010)

***Sol.* Construction of Transition Graph –**



(a)



(b)

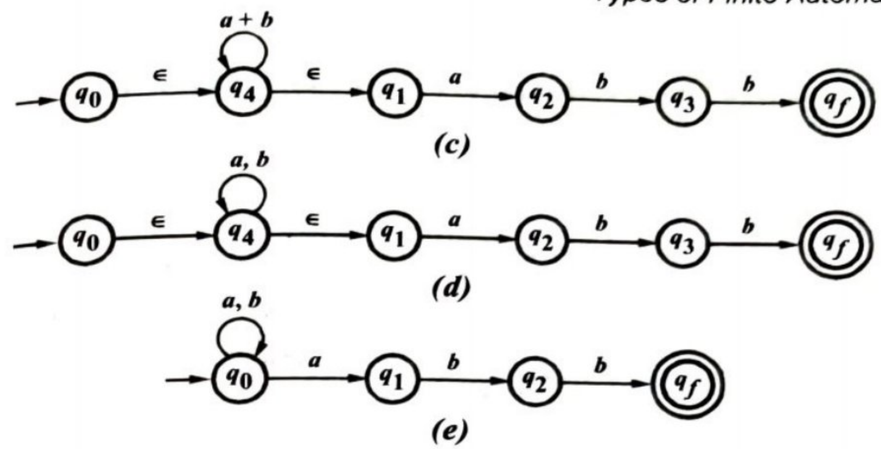


Fig. 2.51 Construction of NDFA

First of all we construct the transition graph with ∈-moves. Then we eliminate ∈-moves. We start with fig. 2.51 (a). We eliminate the concatenation in the given R.E. by introducing new vertices q_1, q_2 and q_3 and get fig. 2.51 (b).

We eliminate *-operation in fig. 2.51 (b) by introducing a new vertex q_4 and ∈-moves as shown in fig. 2.51 (c).

We eliminate + in fig. 2.51 (c) and get fig. 2.51 (d).

We eliminate ∈-moves in fig. 2.51 (d) and get fig. 2.51 (e) which gives the NDFA equivalent to the given R.E.

Construction of DFA – We construct the transition table as given in table 2.24 corresponding to fig. 2.51.

Table 2.24 Transition Table

State	Input	
	a	b
$\rightarrow q_0$	q_0, q_1	q_0
q_1	–	q_2
q_2	–	q_f
q_f	–	–

Table 2.25

Q	Q_a	Q_b
$\rightarrow [q_0]$	$[q_0 q_1]$	$[q_0]$
$[q_0 q_1]$	$[q_0 q_1]$	$[q_0 q_2]$
$[q_0 q_2]$	$[q_0 q_1]$	$[q_0 q_f]$
$[q_0 q_f]$	$[q_0 q_1]$	$[q_0]$

The successor table is constructed for DFA and given in table 2.25.

The state diagram for the successor table is the required DFA as shown in fig. 2.52. As q_f is the only final state of NDFA, hence $[q_0 q_f]$ is the final state of the DFA.

The state diagram for the equivalent automaton is given in fig. 2.52.

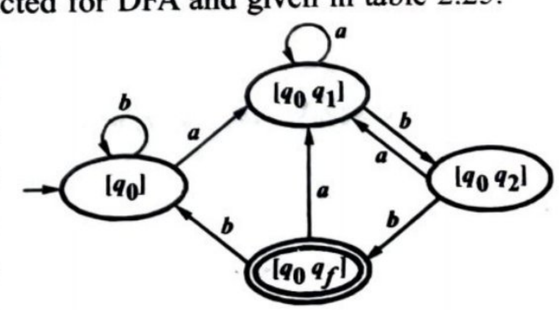
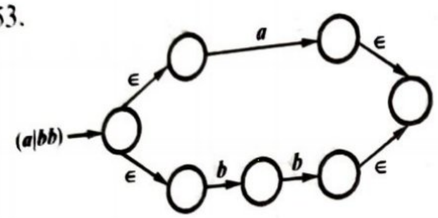


Fig. 2.52 FA

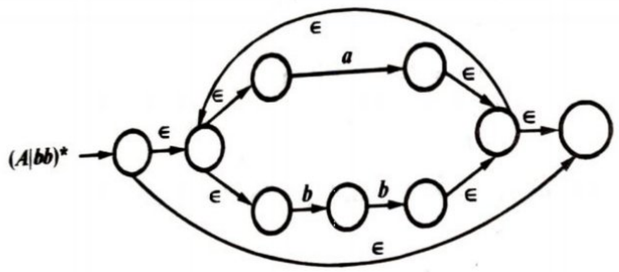
Prob.27. Construct an NFA with ∈ moves, corresponding to the regular expression a*b (a | bb)*, convert it to a DFA and minimize it.

(R.G.P.V., Dec. 2009)

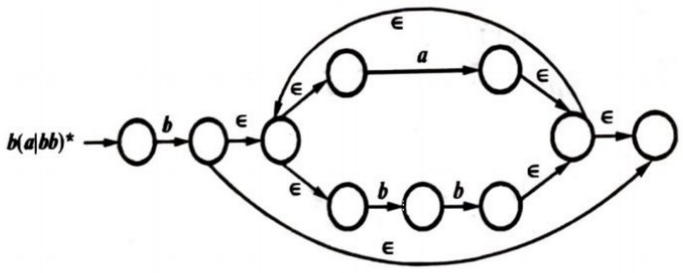
Sol. The NFA for the given regular expression is given below step-by-step in fig. 2.53.



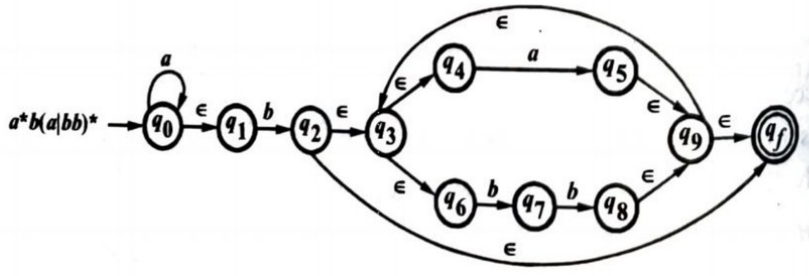
(a)



(b)



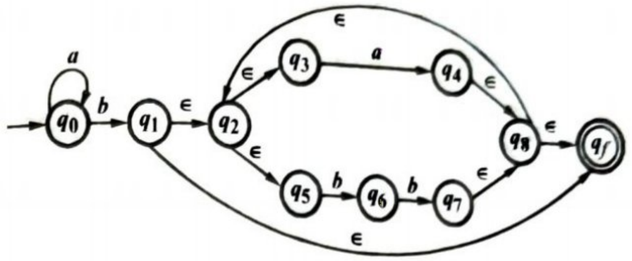
(c)



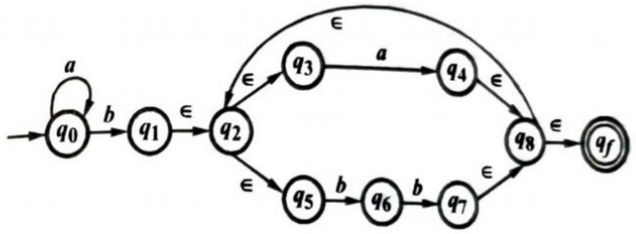
(d)

Fig. 2.53 NDFA with ∈ Moves

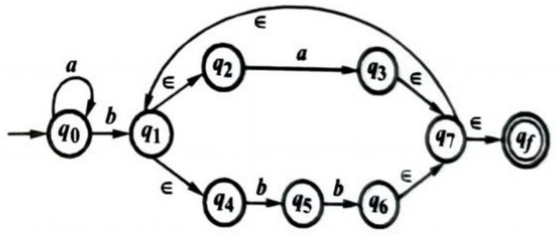
The conversion of NDFA to DFA is given step-by-step in fig. 2.54.



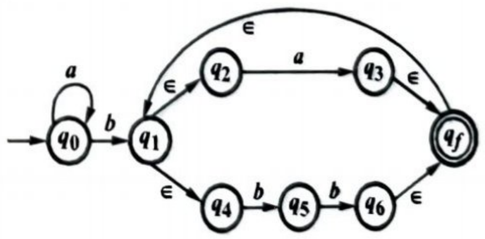
(a)



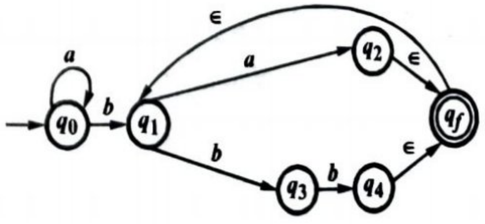
(b)



(c)



(d)



(e)

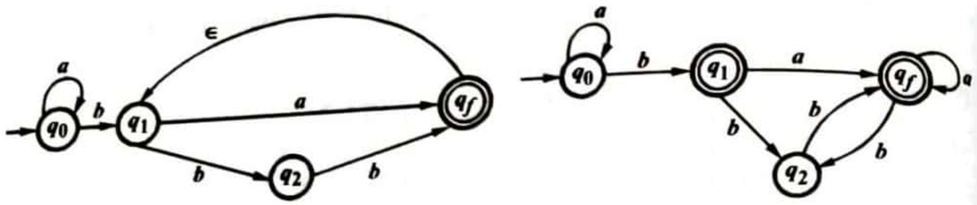


Fig. 2.54 DFA

To minimize the DFA it will be easier if we construct the transition table given in table 2.26.

The successor table is same as table 2.26.

Table 2.26 Transition Table

State	INPUT	
	a	b
→ q_0	q_0	q_1
q_1	q_3	q_2
q_2	–	q_3
q_3	q_3	q_2

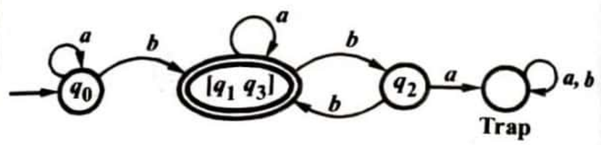


Fig. 2.55 Reduced DFA

The state diagram for the successor table is the required DFA.

Finally, we try to reduce the number of states. As the rows corresponding to q_1 and q_3 are identical, we identify them. The state diagram for the equivalent automaton, where the number of states is reduced, is given in fig. 2.55.

Prob.28. Regular expression (aa*)/(bb*) is given. Construct NFA for the expression and convert this NFA to DFA. ***(R.G.P.V., Dec. 2010)***

Sol. The NFA for the given regular expression is given below –

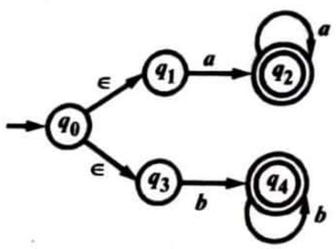


Fig. 2.56 NFA

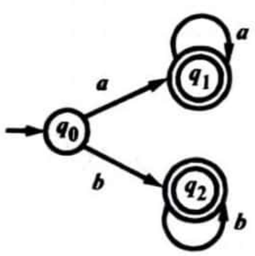


Fig. 2.57 NFA without ∈

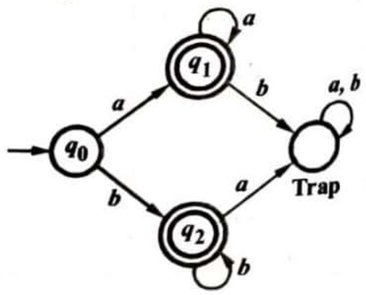


Fig. 2.58 DFA of aa/bb**

Now, eliminating ∈ in fig. 2.56, we get

Now to convert it into DFA, make all possible input transitions to every state.

The required DFA is shown in fig. 2.58.

Prob.29. Convert the following automata to regular expression –

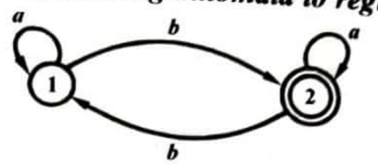


Fig. 2.59 ***(R.G.P.V., Dec. 2007)***

Sol. We can directly apply the algebraic method using Arden's theorem. Since the graph does not contain any ∈-moves and there is only one initial state.

The two equations for q_1 and q_2 can be written as –

$$q_1 = q_1a + q_2b$$
$$q_2 = q_2a + q_1b$$

By applying Arden's theorem, we get,

$$q_1 = a^*(q_2b) = (q_2b)a^*$$

Substituting q_1 in q_2, we get,

$$q_2 = q_2a + (q_2ba^*)b = q_2a + q_2ba^*b$$
$$= q_2(a + ba^*b) = (a + ba^*b)^*$$

As q_2 is the only final state, the regular expression corresponding to the given diagram is $(a + ba^*b)^*$.

Prob.30. Find out the regular expression for given machine and derive the theorem used.

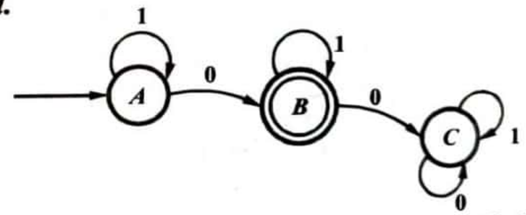


Fig. 2.60 ***(R.G.P.V., June 2006)***

Sol. There is only one initial state. Also, there are no ∈-moves. The equations are

$$A = A1 + \in$$
$$B = A0 + B1$$
$$C = B0 + C(0 + 1)$$

Since $A = A1 + \in$

By Arden's theorem, we get

$$A = \in 1^* = 1^*$$

So, $B = A0 + B1$

$$= 1^*0 + B1$$

Therefore, by Arden's theorem, we get

$$B = (1^*0)\,1^*$$

Since B is the only final state, we need not solve for C, The regular expression corresponding to the given diagram is $(1^*0)1^*$.

Prob.31. State Arden's theorem. Determine regular expression equivalent to finite automata given in fig. 2.61.

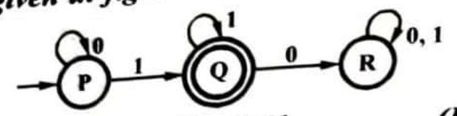


Fig. 2.61 ***(R.G.P.V., Nov. 2018)***

Sol. **Arden's Theorem** – Refer to Q.24.

Since, there is only one initial state and no $\in$ moves. Therefore, the equations for P, Q and R are as follows –

$$P = P0 + \in$$
$$Q = P1 + Q1$$
$$R = Q0 + R(0 + 1)$$

Since $P = P0 + \in$,

Therefore, by Arden's theorem, we get

$$P = \in 0^* = 0^*$$

Similarly, $Q = P1 + Q1 = 0^*1 + Q1$

$$Q = (0^*1)1^*$$

Since Q is the only final state, so we need not to solve for R. Hence regular expression to the given fig. 2.61 is $(0^*1)0^*$.

Prob.32. Find out the regular expression from given DFA.

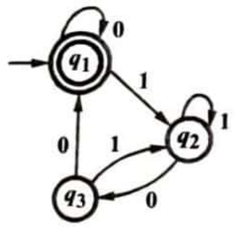


Fig. 2.62 ***(R.G.P.V., Dec. 2014)***

Sol. There is no Λ-moves and only one initial state. We get the following equations for q_1, q_2 and q_3 –

$$q_1 = q_1 0 + q_3 0 + \in$$
$$q_2 = q_2 1 + q_1 1 + q_3 1$$
$$q_3 = q_2 0$$

By substituting q_3 in q_2, we get

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

Applying Arden's theorem on q_2, we get

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

By substituting q_3 in q_1, we get

$$q_1 = q_1 0 + q_2 00 + \in$$

Now, substituting q_2 in q_1, we get

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

Once again by applying Arden's theorem, we get

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

Since q_1 is the only final state, so the regular expression corresponding to fig. 2.62 is $(0 + 1\,(1 + 01)^*\,00)^*$

Prob.33. Convert the following DFA into regular expression

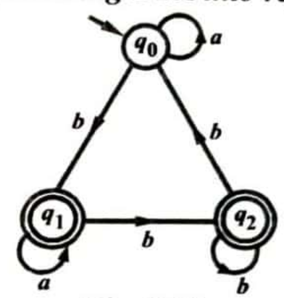


Fig. 2.63 ***(R.G.P.V., June 2007)***

Sol. There is no $\in$-moves and only one initial state. We get the following equations for q_0, q_1 and q_2 –

$$q_0 = q_0 a + q_2 b + \in$$
$$q_1 = q_0 b + q_1 a$$
$$q_2 = q_1 b + q_2 b$$

Applying Arden's theorem on q_1 and q_2, we get

$$q_1 = q_0 b(a)^*$$
$$q_2 = q_1 b(b)^*$$

Substituting resulting q_1 in q_2, we get

$$q_2 = q_0\ ba^*bb^*$$

Substituting resulting q_2 in q_0,

$$q_0 = q_0 a + q_0 ba^*bb^*b + \in$$
$$= q_0(a + ba^*bb^*b) + \in$$
$$= \in(a + ba^*bb^*b)^*$$
$$= (a + ba^*\ bb^*\ b)^*$$

By substituting q_0 in q_1, we get

$$q_1 = (a + ba^*bb^*b)^*ba^*$$

Similarly, $q_2 = (a + ba^*bb^*b)^*\ ba^*bb^*$

As there are two final states q_1 and q_2, thus the required regular expression is obtained by taking union of q_1 and q_2, Therefore,

$$q_1 + q_2 = \{(a + ba^*bb^*b)^*ba^*\} + \{(a + ba^*bb^*b)^*\ ba^*bb^*\}$$
$$= (a + ba^*bb^*b)^*ba^*(\in + bb^*)$$
$$= \mathbf{(a + ba^*bb^*b)^*ba^*b^*}$$

Prob.34. Find a regular expression corresponding to a finite automata.

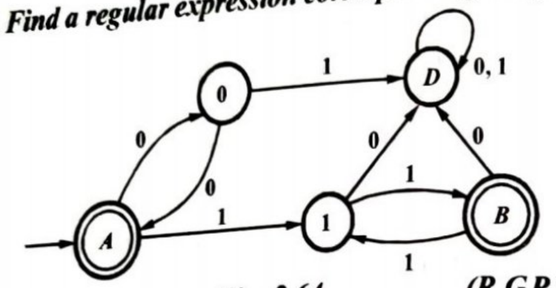


Fig. 2.64 ***(R.G.P.V., Dec. 2005)***

Sol. The state labeled A is both the initial state and an accepting state; this tells us that $\in \in L$. It is easy to see from the diagram that the other strings x for which $\delta^*(A, x) = A$ are even-length strings of 0's, corresponding to the regular expression $(00)^*$.

No matter what input we get from here on, the result will never be in L. We denote this case by state D. Once the FA reaches D, it stays in that state; a string x for which $\delta^*(A, x) = D$ cannot be a prefix of any element of L.

The easiest way to reach the other accepting state B from A is with the string 11. Once the FA is state B, any even-length string of 1's returns it to B, and these are the only strings that do this. Therefore, if x is a string that causes the FA to go from A to B without revisiting A, x must be of the form $11\,(11)^k$ for some $k \geq 0$. Since such a string could be preceded by $(00)^j$, the language

$$\{x \mid \delta^*(A,x) = B\}$$

corresponds to the regular expression $(00)^*\,11(11)^*$. The language L, therefore, corresponds to the regular expression $(00)^* + (00)^*11(11)^*$, which can be simplified to

$$(00)^*\,(11)^*.$$

Prob.35. Convert the following DFA into regular expression.

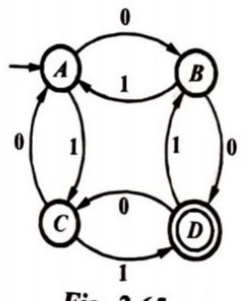


Fig. 2.65 ***(R.G.P.V., June 2007)***

Sol. As the graph does not contain $\in$-moves and there is only one initial state, we get the following equations for A, B, C and D –

$$A = B1 + C0 + \in$$
$$B = A0 + D1$$

$$C = A1 + D1$$
$$D = B0 + C0$$

By substituting B and C in A equation,

$$\begin{aligned} A &= A01 + D11 + A10 + D10 + \in \\ &= (D11 + D10 + \in) + A\,(01 + 10) \\ &= (D\,(11 + 10) + \in)\,(01 + 10)^* \\ &= D\,(11 + 10)\,(01 + 10)^* + (01 + 10)^* \end{aligned}$$

By substituting resulting A in B,

$$\begin{aligned} B &= (D\,(11 + 10)\,(01 + 10)^* + (01 + 10)^*)\,0 + D1 \\ &= D\,(11 + 10)\,(01 + 10)^*\,0 + (01 + 10)^*\,0 + D1 \\ &= D\,((11 + 10)\,(01 + 10)^*\,0 + 1) + (01 + 10)^*0 \end{aligned}$$

By substituting resulting A in C,

$$\begin{aligned} C &= (D\,(11 + 10)\,(01 + 10)^* + (01 + 10)^*)1 + D1 \\ &= D\,(11 + 10)\,(01 + 10)^*1 + (01 + 10)^*1 + D1 \\ &= D\,((11 + 10)\,(01 + 10)^*1 + 1) + (01 + 10)^*1 \end{aligned}$$

Putting resulting B and C in D, we have

$$\begin{aligned} D &= \{D\,((11 + 10)\,(01 + 10)^*\,0 + 1) + (01 + 10)^*0\}\,0 \\ &\quad + \{D\,((11 + 10)\,(01 + 10)^*\,1 + 1) + (01 + 10)^*\,1\}0 \\ &= D\,((11 + 10)\,(01 + 10)^*\,0 + 1)0 + (01 + 10)^*00 \\ &\quad + D\,((11 + 10)\,(01 + 10)^*\,1 + 1)\,0 + (01 + 10)^*\,10 \\ &= D\,\{((11 + 10)\,(01 + 10)^*\,0 + 1)\,0 + (01 + 10)^* \\ &\quad 00 + ((11 + 10)\,(01 + 10)^*\,1 + 1)0 \\ &\quad + (01 + 10)^*\,10\} \end{aligned}$$

Finally by applying Arden's theorem, we get

$$\begin{aligned} D = {} & (((11 + 10)\,(01 + 10)^*\,0 + 1)0 + (01 + 10)^*\,00 \\ & + ((11 + 10)\,(01 + 10)^*\,1 + 1)0 + (01 + 10)^*10)^* \end{aligned}$$

Prob.36. Find out whether the following grammars generate the same language –

G_1 –
$A \to 0B \mid 1E$
$B \to 0A \mid 1F \mid \in$
$C \to 0C \mid 1A$
$D \to 0A \mid 1D \mid \in$
$E \to 0C \mid 1A$
$F \to 0A \mid 1B \mid \in$

G_2 –
$X \to 0Y \mid 0 \mid 1Z$
$Y \to 0X \mid 1Y \mid 1$
$Z \to 0Z \mid 1X$

(R.G.P.V., June 2009)

Sol. Since the grammars G_1 and G_2 are the regular grammars, $L(G_1) = L(G_2)$, if the minimal state automata accepting $L(G_1)$, and the minimal state automata accepting $L(G_2)$ are identical.

The transition diagram of the automata accepting $L(G_1)$ is given in fig. 2.66.

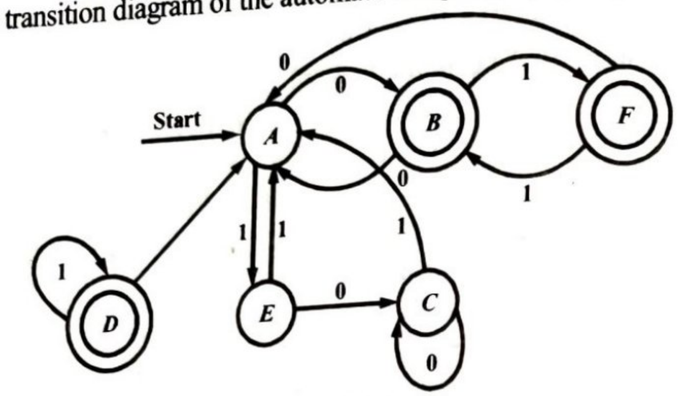


Fig. 2.66

The automata is deterministic, hence to minimize it we proceed as follows –

Since state D is unreachable state, first eliminate state D, hence after eliminating state D we get –

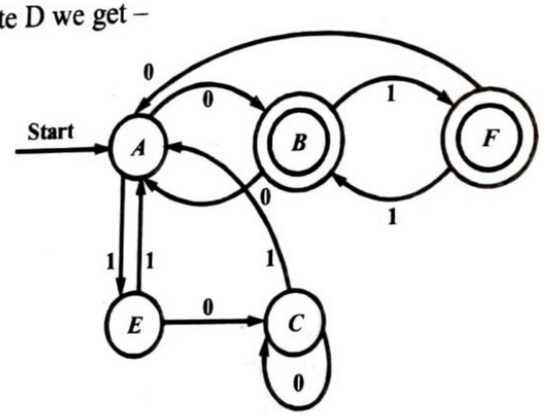


Fig. 2.67

We then identify the non-distinguishable states if any in the above automata as follows –

Initially we have two groups as follows –

A, E, C	B, F
Group I	Group II

Since, $\delta(A, 0) = B$

$\delta(E, 0) = C$

$\delta(C, 0) = C$

State B is distinguishable from rest of the members of group I. Hence we divide group I into two groups – one containing A, and other containing E, C as shown below –

A	E, C	B, F
Group I	Group II	Group III

Since, $\delta(E, 0) = C$

$\delta(C, 0) = C$

Partitioning of group II is not possible because the transitions from all the members of group II goes to group II only for input 0. Similarly,

$\delta(E, 1) = A$

$\delta(C, 1) = A$

Partitioning of group II is not possible, because the transitions from all the members of group II goes to group I only for input 1 –

Since, $\delta(B, 0) = A$

$\delta(F, 0) = A$

Partitioning of group III is not possible, because the transitions from all the members of group III goes to group I only for input 0. Similarly,

$\delta(B, 1) = F$

$\delta(F, 1) = B$

Partitioning of group III is not possible, because the transitions from all the members of group III goes to group III only for input 1.

Hence states E and F are non-distinguishable states, as well as states B and F are also non-distinguishable states. Therefore, if we merge E and C to form a state E_1, B and F to from B_1, we get fig. 2.68.

Since there exists no dead state in the automata shown above, it is a minimal state automata accepting $L(G_1)$.

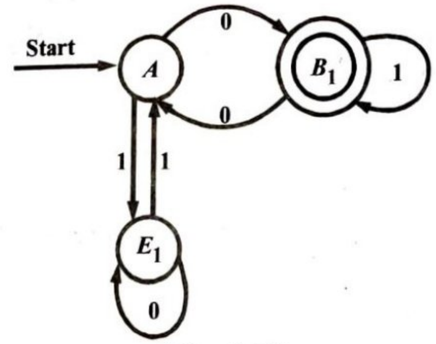


Fig. 2.68

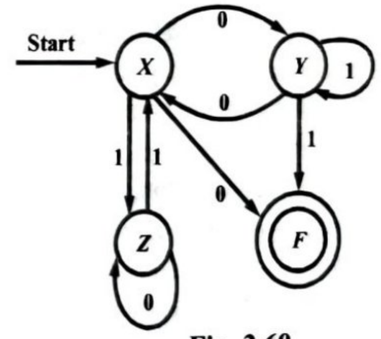


Fig. 2.69

The transition diagram of the automata accepting $L(G_2)$ is fig. 2.69.

This is a non-deterministic automata, its equivalent deterministic automata is –

	0	1
$\{X\}$	$\{Y, F\}$	$\{Z\}$
$*\{Y, F\}$	$\{X\}$	$\{Y, F\}$
$\{Z\}$	$\{Z\}$	$\{X\}$

Therefore, the transition diagram of the deterministic automata is shown in fig. 2.70.

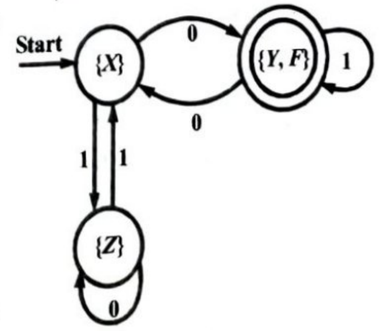


Fig. 2.70

This automata does not contain, unreachable, non-distinguishable, as well as dead state, hence it is a minimal state automata accepting $L(G_2)$, and it is identical to the minimal state automata accepting $L(G_1)$, therefore, $L(G_1) = L(G_2)$ and thus G_1 and G_2 generate the same language.

Prob.37. Construct minimum state DFA for the ahead machine using minimization algorithm.

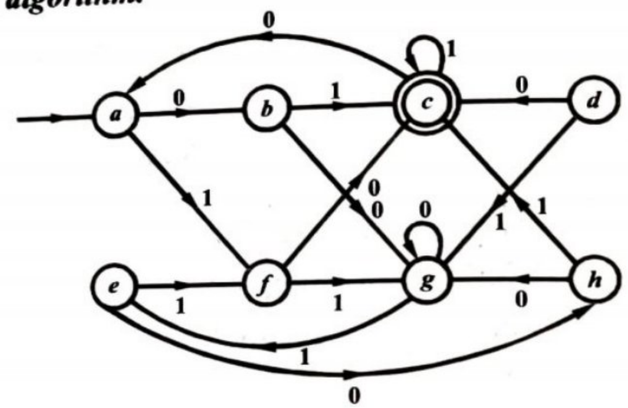


Fig. 2.71

(R.G.P.V., Dec. 2006, June 2007)

Sol. Let M be the finite automaton of fig. 2.71. In fig. 2.72 we have constructed a table with an entry for each pair of states. An × is placed in the table each time we discover a pair of states that cannot be equivalent. Initially an × is placed in each entry corresponding to one final state and one nonfinal state. We place an × in the entries (a, c), (b, c), (c, d), (c, e), (c, f), (c, g), and (c, h).

Next for each pair of states p and q that are not already known to be distinguishable we consider the pairs of states $r = \delta(p, a)$ and $s = \delta(q, a)$ for each input symbol a. If states r and s have been shown to be distinguishable by some string x, then p and q are distinguishable by string ax. Thus if the entry (r, s) in the table has an ×, and × is also placed at the entry (p, q). If the entry (r, s) does not yet have an ×, then the pair (p, q) is placed on a list associated with the (r, s)-entry. At some future time, if the (r, s) entry receives an ×, then each pair on the list associated with the (r, s)-entry also receives an ×.

Now, we place an × in the entry (a, b), since the entry $(\delta(b, 1), \delta(a, 1)) = (c, f)$ already has an ×. Similarly, the (a, d)-entry receives an × since the entry $(\delta(a, 0), \delta(d, 0)) = (b, c)$ has an ×. Consideration of the (q, e)-entry on input 0 results in the pair (a, e) being placed on the list associated with (b, h). Observe that on input 1, both a and e go to the same state f and hence no string starting with a 1 can distinguish a from e. Because of the 0-input, the pair (a, g) is placed on the list associated with (b, g). When the (b, g)-entry is considered, it receives an × on account of a 1-input, and hence the pair (a, g) receives an × since it was on the list for (b, g). The string 01 distinguishes a from g.

On completion of the table in fig. 2.72, we conclude that the equivalent states are $a \equiv e$, $b \equiv h$, and $d \equiv f$. The minimum-state finite automaton is given in fig. 2.73.

	a	b	c	d	e	f	g
b	×						
c	×	×					
d	×	×	×				
e		×	×	×			
f	×	×	×		×		
g	×	×	×	×	×	×	
h	×		×	×	×	×	×

Fig. 2.72 Calculation of Equivalent States

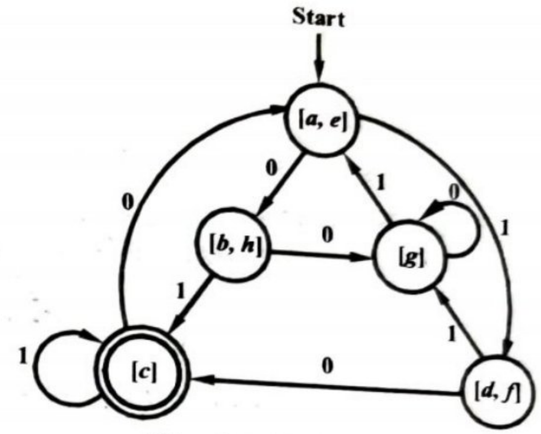


Fig. 2.73 Minimum State Finite Automaton

Prob.38. Show that the set $L = \left\{b^{i^2} \mid i \geq 1\right\}$ is not regular.

(R.G.P.V., June 2011)

Or

State pumping lemma for regular sets and apply it to prove that $L = \left\{a^{j^2} \mid j \geq 1\right\}$ is not regular.

(R.G.P.V., Nov. 2018)

Sol. **Pumping Lemma for Regular Sets** – Refer to Q.30.

(i) Step 1 – Suppose L is regular and we get a contradiction. Let n be the number of states in FA accepting L.

(ii) Step 2 – Let $u = b^{n^2}$. Then $|u| = n^2 > n$. By pumping lemma, we can write $u = xyz$ with $|xy| \leq n$ and $|y| > 0$.

(iii) Step 3 – Consider xy^2z. $|xy^2z| = |x| + 2|y| + |z| > |x| + |y| + |z|$ as $|y| > 0$. It means $n^2 = |xyz| = |x| + |y| + |z| < |xy^2z|$. As $|xy| \leq n, |y| \leq n$. Thus, $|xy^2z| = |x| + 2|y| + |z| \leq n^2 + n$, i.e.,

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

Hence, $|xy^2z|$ strictly occurs between n^2 and $(n + 1)^2$, but is not equal to any one of them. Thus, $|xy^2z|$ is not a perfect square. So $xy^2z \notin L$. But by pumping lemma $xy^2z \in L$. This is a contradiction. ***Thus, L is not regular.***

Prob.39. Prove $L = \{a^p \mid p$ is a prime$\}$ is not regular using pumping lemma. **(R.G.P.V., Dec. 2014)**

Sol. (i) Step 1 – We suppose L is regular and get a contradiction. Let n be the number of states in the finite automaton accepting L.

(ii) Step 2 – Let p be a prime number greater than n, Let $u = a^p$. By pumping lemma, u can be written as $u = xyz$, with $|xy| \le n$ and $|y| > 0$. x, y, z are simply strings of a's. So, $y = a^m$ for some $m \ge 1$ (and $\le n$).

(iii) Step 3 – Let $i = p+1$. Then $|xy^iz| = |xyz| + |y^{i-1}| = p + (i-1)m = p + pm$. By pumping lemma, $xy^iz \in L$. But $|xy^iz| = p + pm = p(1+m)$ and $p(1+m)$ is not a prime. So $xy^iz \notin L$. This is a contradiction. ***Thus, L is not regular.***

Prob.40. Use pumping lemma to prove that the following sets are not regular.

(i) $L = \{a^n b^{2n} \mid n \ge 0\}$ (ii) $L\ \{a^n \mid n$ is a prime number$\}$. **(R.G.P.V., June 2007)**

Sol. (i) $L = \{a^n b^{2n} \mid n \ge 0\}$ – We prove L is not regular by contradiction. If L is regular, we can apply pumping lemma. Let n be the number of states. Let $w = a^n b^{2n}$. By pumping lemma, $w = xyz$ with $|xy| \le n$, $|y| > 0$ and $xz \in L$. As $|xy| \le n$, $xy = a^m$ and $y = a^l$ where $0 < l \le n$. So $xz = a^{n-l}b^{2n} \in L$, a contradiction since $n - l \ne n$. Thus L is not regular.

(ii) $L\ \{a^n \mid n$ is a prime number$\}$ – Refer sol. of Prob.39.

Prob.41. Using pumping lemma show that the following set are not regular –

(i) $\{a^n b^{2n} \mid n > 0\}$ (ii) $\{a^n b^m \mid 0 < n < m\}$. **(R.G.P.V., Dec. 2009)**

Sol. (i) Let i be constant of pumping lemma.

Select $Z = a^i b^{2i}$, this ensures that Z is in L and $|Z| \ge i$.

If we write $Z = uvw$, then the possible choices of v satisfying the conditions –

$1 \le |v| \le i$ and $|uv| \le i$ are –

$v = a^p$, where $1 \le p \le i$, i.e., v can be chosen to be string of minimum one a and maximum i a's

For this choice of v, uv^0w will be $a^{i-p}b^i$, and since $1 \le p \le i$, uv^0w will have $(i-p)$ a's and $(2i)$ b's, hence it cannot belong to L. Therefore, we get contradiction to pumping lemma.

v cannot contain b's, because if v is chosen to contain b's, then $|uv|$ will not be less than or equal to i. Thereby proving that L is not regular.

(ii) Let n be constant of pumping lemma.

Select $Z = a^n b^m$, this ensures that Z is in L, and $|Z| \ge n$ (because $n < m$).

If we write $Z = uvw$, then the possible choices of v satisfying the conditions –

$1 \le |v| \le n$ and $|uv| \le n$ are –

$v = a^p$, where $1 \le p \le n$, i.e., v can be chosen to be string of minimum one a and maximum n a's.

For this choice of v, uv^2w will be $a^{n+p}b^n$, and since $1 \le p \le n$, uv^2w will have more number of a's than b's, hence it cannot belong to L. Therefore, we get contradiction to pumping lemma.

v cannot contain b's, because if v is chosen to contain b's, then $|uv|$ will not be less than or equal to n. Thereby proving that L is not regular.

Prob.42. Show that $L = \{0^i 1^i / i \ge 1)$ is not regular.

Sol. (i) Step 1 – Suppose L is regular and we get a contradiction. Let n be the number of states in finite automaton accepting L.

(ii) Step 2 – Let $u = 0^n 1^n$. Then $|u| = 2n > n$. By pumping lemma we write $u = xyz$ with $|xy| \le n$ and $|y| \ne 0$.

(iii) Step 3 – We want to find i so that $xy^iz \notin L$ for getting a contradiction. The string y can be in any of the following forms –

(a) Case 1 – y has only 0's, i.e., $y = 0^k$ for some $k \ge 1$.

(b) Case 2 – y has only 1's, i.e., $y = 1^l$ for some $l \ge 1$.

(c) Case 3 – y has both 0's and 1's, i.e., $y = 0^k 1^j$ for some $k, j \ge 1$.

In case 1, we can take $i = 0$. As $xyz = 0^n 1^n$, $xz = 0^{n-k}1^n$. As $k \ge 1$, $n - k \ne n$. So $xz \notin L$.

In case 2, we take $i = 0$. As before, xz is $0^n 1^{n-l}$ and $n \ne n - l$. So $xz \notin L$.

In case 3, we take $i = 2$. As $xyz = 0^{n-k} 0^k 1^j 1^{n-j}$, $xy^2z = 0^{n-k}0^k 1^j 0^k 1^j 1^{n-j}$. As xy^2z is not of the form $0^i 1^i$, $xy^2z \notin L$.

Thus in all the cases we get a contradiction. ***Therefore, L is not regular.***

Prob.43. What do you mean by closure properties of regular languages ? State these properties. State pumping Lemma and show that $L = \{a^i b^i | i >= 1\}$ is not a regular language. **(R.G.P.V., Dec. 2015)**

Sol. Refer to Q.32, Q.30 and Prob.42.

Prob.44. Prove the following is not regular language –

$\{0^n 1^n | n \ge 1\}$

This language consisting of a string of 0's followed by an equal length string of 1's, is the language L_{01}. **(R.G.P.V., Dec. 2008)**

Sol. Refer to Prob.42.

Prob.45. Apply pumping lemma to prove that $L = \{a^n b^{2n} | n \ge 1\}$ is not regular set. **(R.G.P.V., Dec. 2003)**

Sol. (i) Step 1 – Suppose L is regular and we get a contradiction. Let m be the number of states in finite automaton M accepting L.

(ii) Step 2 – Let $u = a^m b^{2m}$. Then $|u| = 3m > m$. By pumping lemma we write $u = xyz$ with $|xy| \le n$ and $|y| \ne 0$.

(iii) Step 3 – We want to find i so that $xy^iz \notin L$ for getting a contradiction. The string y can be in any of the following forms –

Case 1 – y has only a's, i.e., $y = a^k$ for some $k \ge 1$.

Case 2 – y has only b's, i.e., $y = b^l$ for some $l \ge 1$.

Case 3 – y has both a's and b's, i.e., $y = a^k b^j$ for some $k, j \ge 1$.

In case 1, we can take $i = 0$. As $xyz = a^m b^{2m}$, $xz = a^{m-k} b^{2m}$. As $k \ge 1$, $m - k \ne m$. So $xz \notin L$.

In case 2, we take $i = 0$. As before xz is $a^m b^{2m-l}$ and $2m - l \ne 2m$ as $l \ge 1$. So $xz \notin L$.

In case 3, we take $i = 2$. As $xyz = a^{m-k} a^k b^j b^{2m-j}$, $xy^2z = a^{m-k} a^k b^j a^k b^j b^{2m-j}$. As xy^2z is not of the form $a^n b^{2n}$, $xy^2z \notin L$.

Thus, in all cases we get a contradiction. ***Therefore, L is not regular.***

Prob.46. Prove whether the sets is regular or not –

(i) $\{0^{2n} \mid n \ge 1\}$ (ii) $\{0^n 1^n \mid n \ge 0\}$. (R.G.P.V., Dec. 2006)

Sol. (i) $\{0^{2n} \mid n \ge 1\}$ – We can write 0^{2n} as $0\,(0^2)^i 0$, where $i \ge 0$. Now $\{(0^2)^i \mid \ge 0\}$ is simply $\{0^2\}^*$, so L is represented by the regular expression $0\,(p)^*0$, where p represents $\{0^2\}$. The corresponding finite automata is given in fig. 2.74.

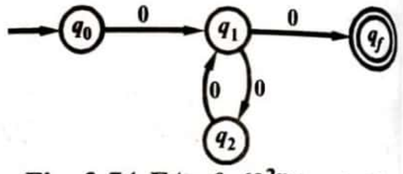


Fig. 2.74 FA of $\{0^{2n} \mid n \ge 1\}$

(ii) $\{0^n 1^n \mid n \ge 0\}$ – The proof is by contradiction. Let $L = \{0^n 1^n \mid n \ge 0\}$. $\{\delta(q_0, 0^n) \mid n \ge 0\}$ is a subset of Q and hence finite, so $\delta(q_0, 0^n) = \delta(q_0, 0^m)$ for some m and n, $m \ne n$. So $\delta(q_0, 0^m 1^n) = \delta(\delta(q_0, 0^m), 1^n) = \delta(\delta(q_0, 0^n), 1^n) = \delta(q_0, 0^n 1^n)$. As $0^n 1^n \in L$, $\delta(q^0, 0^n 1^n)$ is a final state and so is $\delta(q_0, 0^m 1^n)$ This means $0^m 1^n \in L$ with $m \ne n$, which is a contradiction. Hence L is not regular.

Prob.47. Explain pumping lemma for regular sets. Also prove following set is not regular.

$L = \{0^m 1^n 0^{m+n} \mid m \ge 1 \text{ and } n \ge 1\}$ (R.G.P.V., Dec. 2012)

***Sol.* Pumping Lemma** – Refer to Q.30.

Given set, L is $\{0^m 1^n 0^{m+n} \mid m \ge 1 \text{ and } n \ge 1\}$

$L = \{0100, 011000, 001000, 00110000, ...\}$

Case I – Let, $y = 1$ ($\because y \ne \in$)

$x = 0$

$z = 00$

$u = 0(1)^i 00$

When $i = 1$, $u = 0100$ which is in 'L'

When $i = 2$, $u = 01100$ which is not in 'L'

Therefore, $L = \{0^m 1^n 0^{m+n} \mid m \ge 1, n \ge 1\}$ is not regular.

Case 2 – Let, $y = 0$

$x = 01$

$z = 0$

$u = 01(0)^i 0$

When $i = 1$, $u = 0100$ which is in 'L'

When $i = 2$, $u = 01000$ which is not in 'L'

Therefore, $L = \{0^m 1^n 0^{m+n} \mid m \ge 1, n \ge 1\}$ is not regular.

Case 3 – Let, $y = 10$

$x = 0$

$z = 0$

$u = 0(10)^i 0$

When $i = 1$, $u = 0100$ which is in 'L'

When $i = 2$, $u = 010100$ which is not in 'L'

Therefore, $L = \{0^m 1^n 0^{m+n} \mid m \ge 1, n \ge 1\}$ is not regular.

Prob.48. Prove that $a^n \subset b^n | n \ge 1$ is not regular. (R.G.P.V., Dec. 2011)

Sol. Let u be any string such that

$u \in a^n \subset b^n$ and $|u| \ge 3$

Now, let us assume that

$u = xyz$ where $|y| \ne 0$ and $|xy| \le n$

and according to pumping lemma

If u is regular the xy^iz is also regular and vise versa.

Here, 6 cases arises –

Case-1 – Let $y = c$

$x = a$

$z = b$

$u = xy^iz = a(c)^i b$

When $i = 1$, $u = acb$, which is in L.

When $i = 2$, $u = accb$, which is not in L.

Hence the language is not regular.

Case-2 – Let $y = a$

$x = \in$

$z = cb$

$u = xy^iz = (a)^i cb$

When $i = 1$, $u = acb$, which is in L.

When $i = 2$, $u = aacb$, which is not in L.

Hence the language is not regular.

Case-3 – Let $y = b$

$x = ac$

$z = \in$

$u = ac(b)^i$

When $i = 1$, $u = acb$, which is in L.

When $i = 2$, $u = acbb$, which is not in L.

Case-4 – Let $y = ac$
$x = \epsilon$
$z = b$
$u = (ac)^i b$

When $i = 1$, $u = acb$, which is in L.

When $i = 2$, $u = acacb$, which is not in L.

Hence, given language is not regular.

Similarly for case-5 as $y = cb$, $z = \epsilon$, $x = a$ the given language is not regular for $i = 2$.

Case-6 – Let $y = acb$
$x = \epsilon$ & $z = \epsilon$
$u = (acb)^i$

When $i = 1$, $u = acb$, which is in L.

When $i = 2$, $u = acbacb$, which is not in L.

Hence, the given language is not regular.

It is clear from all the above cases that the language $L = \{a^n c b^n / n \geq 1\}$ is not regular.

Prob.49. Prove by induction –

$$\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}.$$

(R.G.P.V., Dec. 2003, June 2010)

***Sol.* Proof** – Assume

$$S(n) = \sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$$

Basis – For $n = 1$,

$$S(1) = \sum_{i=1}^{1} \frac{1}{i(i+1)} = \frac{1}{1(2)} = \frac{1}{1+1} \Rightarrow S(1) \text{ is true}$$

Inductive Hypothesis –

$$S(k) = \sum_{i=1}^{k} \frac{1}{i(i+1)} = \frac{k}{k+1} \text{ is true.}$$

Inductive Step –

$$S(k+1) = \sum_{i=1}^{k+1} \frac{1}{i(i+1)} = \sum_{i=1}^{k} \frac{1}{i(i+1)} + \frac{1}{(k+1)(k+2)}$$

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

Therefore, $S(k) \Rightarrow S(k+1)$. Hence, the result follows by mathematical induction.

UNIT 3

GRAMMARS

TYPES OF GRAMMAR, CONTEXT SENSITIVE GRAMMAR, AND CONTEXT FREE GRAMMAR, REGULAR GRAMMAR, DERIVATION TREES, AMBIGUITY IN GRAMMAR, SIMPLIFICATION OF CONTEXT FREE GRAMMAR, CONVERSION OF GRAMMAR TO AUTOMATA MACHINE AND VICE VERSA, CHOMSKY HIERARCHY OF GRAMMAR, KILLING NULL AND UNIT PRODUCTIONS, CHOMSKY NORMAL FORM AND GREIBACH NORMAL FORM

Q.1. How many types of grammar are there ? Explain each of them.
(R.G.P.V., Dec. 2011)

Or

Write short note on Chomsky classification of grammar.
(R.G.P.V., June 2003, 2010)

Or

Write short note on Chomsky hierarchy. ***(R.G.P.V., June 2006)***

Ans. Chomsky classified the grammars into four types in terms of productions (types 0-3).

A type 0 grammar is any phrase structure grammar without any restrictions.

A production without any restrictions is called a type 0 production.

A production of the form $\phi A \psi \rightarrow \phi \alpha \psi$ is called a type 1 production if $\alpha \neq \in$. In type 1 productions erasing of A is not permitted. Here A is a variable, ϕ is called the left context ψ, the right context, and $\phi \alpha \psi$ the replacement string.

A grammar is called type 1 or context-sensitive or context dependent if all its productions are type 1 productions. The production $S \rightarrow \in$ is also allowed in a type 1 grammar, but in this case S does not appear on the right-hand side of any production. The language generated by a type 1 grammar is called a type 1 or context-sensitive language.

A type 2 production is a production of the form $A \rightarrow \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup \Sigma)^*$. That is, the L.H.S. has no left context or right context. For example, $S \rightarrow Aa$, $A \rightarrow a$, $B \rightarrow abc$, $A \rightarrow \in$ are type 2 productions.

A grammar is called a type 2 grammar if it contains only type 2 productions. It is also called a context-free grammar. A language generated by a context-free grammar is called type 2 language or a context-free language.

A production of the form $A \to a$ or $A \to aB$, where $A, B \in V_N$ and $a \in \Sigma$ is called a type 3 production.

A grammar is called a type 3 or regular grammar if all its productions are type 3 productions. A productions $S \to \in$ is allowed in type 3 grammar, but in this case S does not appear on the right-hand side of any production.

Q.2. Describe context free and context sensitive grammar. ***(R.G.P.V., June 2015)***

Ans. Refer to Q.1.

Q.3. What do you understand by a context-free grammar ? Give some examples.

Or

Define context-free grammar. ***(R.G.P.V., Dec. 2007)***

Or

Define CFG ***(R.G.P.V., Dec. 2016)***

Or

Explain context-free grammars. ***(R.G.P.V., May 2018)***

Ans. A grammar $G = (V, T, P, S)$ is said to be context-free, if every production in P is of the form –

$$A \to x$$

where, $A \in V$ and $x \in (V \cup T)^*$.

here, V = Non terminal variables

T = Set of terminal values

S = Start symbol

P = Production rules.

A language is said to be context-free if there is a context-free grammar G such that $L = L(G)$.

Every regular grammar is context-free, so a regular language is also context-free. But as we know that there are non-regular languages.

We can say that, the family of regular languages is a proper subset of the family of context-free languages.

For example –

The grammar $G = (\{S\},\{a,b\},P,S)$

having productions

$$S \to aSa$$
$$S \to bSb$$
$$S \to \lambda$$

is context-free. A typical derivation in this grammar is as follows –

$$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aabbaa$$

Q.4. What do you mean by context free ? Explain the types of grammar. ***(R.G.P.V., June 2011, Dec. 2017)***

Ans. Refer to Q.3 and Q.1.

Q.5. Why CFG is not considered adequate for describing natural language ? Explain with suitable example. ***(R.G.P.V., Dec. 2015)***

Ans. For a number of reasons, context free grammars are not in general regarded as adequate for the description of natural languages like English. For example if we extend the productions

< sentence > → < noun phrase > < verb phrase >

< noun phrase > → < adjective > < noun phrase >

< noun phrase > → < noun >

< noun > → boy

< adjective > → little

to encompass all of English, we would be able to derive "rock" as a noun phrase and "runs" as a verb phrase. Thus "rock runs" would be a sentence, which is nonsense. Clearly some semantic information is necessary to rule out meaningless strings that are made to associate the meaning of the sentence with its derivation.

Q.6. What do you understand by a regular grammar ?

Or

Explain the left linear grammar. ***(R.G.P.V., Dec. 2007)***

Ans. A ***regular grammar*** is one that is either left-linear or right-linear.

A grammar $G = (V, T, P, S)$ is said to be ***left-linear*** if all productions are of the form –

$$A \to Bx \quad \text{or} \quad A \to x$$

where $A, B, \in V$, and $x \in T^*$

A grammar is said to be ***right-linear*** if all productions are of the form –

$$A \to xB$$
$$A \to x$$

Note that, in a regular grammar at most one variable appears on the right side of any production. Moreover, that variable must consistently be either the leftmost or rightmost symbol of the right side of any production.

Q.7. How can we construct regular grammar from regular expression ? ***(R.G.P.V., June 2015)***

Ans. We can construct regular grammar from regular expression by using following method –

(i) Construct a NFA with $\in$ from given regular expression.

(ii) Eliminate $\in$ transitions and convert it to equivalent DFA.

(iii) From constructed DFA, the corresponding states become non terminal symbols and transitions made are equivalent to production rules.

Q.8. Prove that a language L is regular if and only if there exists a left-linear grammar G such that L = L(G).

***Ans.* Proof** – Here we outline the main idea only.

Given any left-linear grammar with productions of the form –

$$A \to Bv$$
$$A \to v$$

or

Construct from it, a right-linear grammar G' by replacing every such productions on G with

$$A \to v^R B$$
$$A \to v^R$$

or

respectively.

Few examples will make it clear quickly that $L(G) = (L(G'))^R$. We know that the transpose or reverse of any regular language is also regular. Since G' is right-linear, $L(G')$ is regular. But then so are $(L(G'))^R$ and $L(G)$.

Q.9. Define derivation tree or parse tree with an example.

Or

(R.G.P.V., May 2018)

Explain derivation trees.

Ans. A derivation tree is an ordered tree in which nodes are labelled with the left sides of productions of any grammar, also in which, the children of a node represent its corresponding right sides.

For example, let $G = (\{S, A\}, \{a, b\}, P, S)$, where P consists of

$$S \to aAS \mid a \mid SS, A \to SbA \mid ba.$$

Fig. 3.1 shows a derivation tree representing

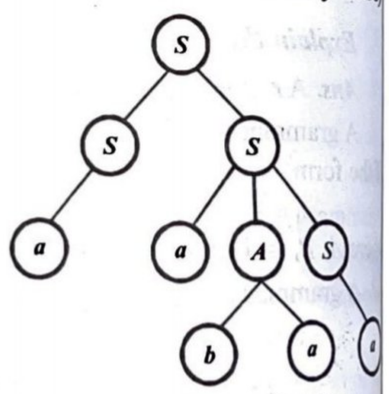


Fig 3.1 Derivation Tree

$$S \Rightarrow SS \Rightarrow aS \Rightarrow aaAS \Rightarrow aabaS \Rightarrow aabaa$$

Q.10. What are leftmost and rightmost derivations ? Explain with suitable example. ***(R.G.P.V., Dec. 2015)***

Or

Explain LMD and RMD with example. ***(R.G.P.V., June 2011)***

Ans. A derivation is said to be ***leftmost,*** if in each step the leftmost variable in the sentential form is replaced. And if in each step the rightmost variable is replaced, the derivation is called ***rightmost.***

For example, given a grammar having productions

$$S \to aAB,$$
$$A \to bBb,$$
$$B \to A \mid \in.$$

Then,

$$S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abAbB \Rightarrow abbBbbB \Rightarrow abbbbB \Leftarrow abbbb$$

is a leftmost derivation of the string *abbbb*. The rightmost derivation of the same string *abbbb* is

$$S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abAb \Rightarrow abbBbb \Rightarrow abbbb.$$

Q.11. Explain leftmost derivation, rightmost derivation and parse with suitable example. ***(R.G.P.V., June 2010)***

Ans. Refer to Q.10 and Q.9.

Q.12. Explain ambiguity in context-free grammars.

Or

Write short note on ambiguity in grammar. ***(R.G.P.V., June 2006)***

Or

Explain ambiguous grammar problem. ***(R.G.P.V., June 2016)***

Or

Define ambiguity. ***(R.G.P.V., Dec. 2016)***

Or

Give a detailed description of ambiguity in context free grammar. ***(R.G.P.V., May 2018)***

Ans. Sometimes there may occur some ambiguous sentences in the language we are using. Let us take an example of the following english sentence – "In books selected information is given". The word 'selected' may refer to books or information. So the sentence may be parsed in two different ways. The same situation may arise in context-free languages.

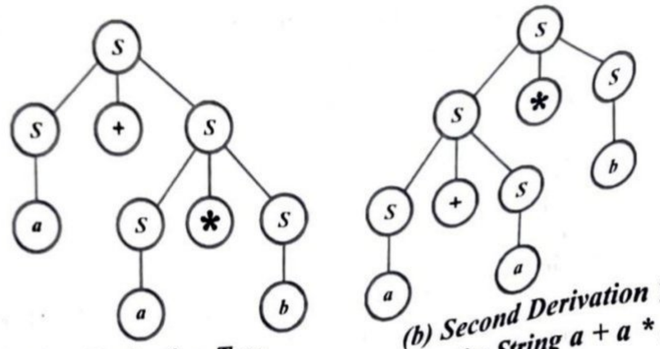
The same terminal string may be the yield of two different derivation trees. So there may be two different leftmost derivations for a string. This leads to the definition of ambiguous sentences in a context-free language, as follows –

A terminal string $u \in L(G)$ is ambiguous, if there exist two or more derivation trees for u (or there exist two or more leftmost derivations of u).

For example, $G = (\{S\}, \{a, b, +, *\}, P, S)$, where P consists of

$$S \to S + S \mid S * S \mid a \mid b$$

We have two derivation trees for string $a + a * b$, given in fig. 3.2.



(a) First Derivation Tree for String a + a * b

(b) Second Derivation Tree for String a + a * b

Fig. 3.2 Two Different Derivation Trees for One String a + a * b

The leftmost derivations of $a + a * b$ induced by the two derivation trees are as follows –

(i) $S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * b$

(ii) $S \Rightarrow S * S \Rightarrow S + S * S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * b$

Therefore, string $a + a * b$ is ambiguous.

So, we can define that a context-free grammer G is ambiguous if there exists some $u \in L(G)$, which is ambiguous.

Q.13. Prove that the CFG G1 with productions –

$$S1 \rightarrow S1 + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (S1) \mid a$$

(R.G.P.V., Dec. 2005)

is unambiguous.

***Ans.* Proof –** We wish to show that every string x in $L(G1)$ has only one leftmost derivation from $S1$. The proof will be by mathematical induction on $|x|$, and it will actually be easier to prove something apparently stronger – For any x derivable from one of the variables $S1$, T, or F, x has only one leftmost derivation from that variable.

For the basis step, we observe that a can be derived from any of the three variables, and that in each case there is only one derivation.

In the induction step, we assume that $k \geq 1$ and that for every y derivable from $S1$, T, or F for which $|y| \leq k$, y has only one leftmost derivation from that variable. We wish to show the same result for a string x with $|x| = k + 1$.

Consider first the case in which x contains at least one + not within parentheses. Because the only +'s in strings derivable from T or F are within parentheses, x can be derived only from $S1$, and any derivation of x must begin $S1 \Rightarrow S1 + T$, where this + is the last + in x that is not within parentheses. Therefore, any leftmost derivation of x from $S1$ has the from

$$S1 \Rightarrow S1 + T \Rightarrow * y + T \Rightarrow * y + z$$

where the last two steps represent leftmost derivation of x from $S1$ and z, from T, respectively, and the + is still the last one not within parentheses. The induction hypothesis tells us that y has only one leftmost derivation from $S1$ and z has only one from T. Therefore, x has only one leftmost derivation from $S1$.

Next consider the case in which x contains no + outside parentheses but at least one * outside parentheses. This time x can be derived only from $S1$ or T; any derivation from $S1$ must begin $S1 \Rightarrow T \Rightarrow T * F$, and any derivation from T must begin $T \Rightarrow T * F$. In either case, the * must be the last one in x that is not within parentheses. As in the first case, the subsequent steps of any leftmost derivation must be

$$T * F \Rightarrow * y * F \Rightarrow * y * z$$

consisting first of a leftmost derivation of y from T and then of a leftmost derivation of z from F. Again, the induction hypothesis tell us that there is only one possible way for these derivations to proceed, and therefore there is only one leftmost derivation of x from $S1$ or T.

Finally, suppose x contains no +'s or *'s outside parentheses. Then x can be derived from any of the variables, however, the only derivation from S1 begins $S1 \Rightarrow T \Rightarrow F \Rightarrow (S1)$, and the only derivation from T or F begins the same way, with the first one or two steps omitted. Therefore, $x = (y)$, where $S1 \Rightarrow * y$. By the induction hypothesis, y has only one leftmost derivation from $S1$, and it follows that x has only one from each of the three variables. This completes the proof.

Q.14. Explain rule for simplification of CFG (R.G.P.V., June 2010)

Ans. In a context-free grammar G, consists of (V, T, P, S), it may not be necessary to use all the symbols in $V \cup T$, or all the production in P for deriving sentences. So when we study a context-free language $L(G)$, we try to eliminate symbols and productions in G which are not useful for derivation of sentences. For example, consider

$$G = (\{S, A, B, C, E\}, \{a, b, c\}, P, S)$$

where $P = \{S \rightarrow AB, A \rightarrow a, B \rightarrow b, B \rightarrow C, E \rightarrow c \mid \in\}$

It is easy to observe that $L(G) = \{ab\}$. Let

$$G' = (\{S, A, B\}, \{a, b\}, P', S),$$

where $P' = \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\}$ and $L(G) = L(G')$

We have eliminated the symbols C, E and c and the productions $B \rightarrow C$, $E \rightarrow c \mid \in$. The following points regarding the symbols and productions can be eliminated –

(i) C does not derive any terminal string

(ii) E and c do not appear in any sentential form

(iii) $E \rightarrow \in$ is a null production

(iv) $B \rightarrow C$ simply replaces B by C.

In this way one can restrict the format of productions without reducing the generative power of context-free grammars.

Q.15. Explain the following –
(i) Parse tree (ii) Useless symbols. ***(R.G.P.V., Dec. 2007)***

Ans. (i) Parse Tree – Refer to Q.9.

(ii) Useless Symbols – A symbol that is not useful is useless symbol. Clearly, if a variable is either not live or not reachable, then it is useless.

Let $G = (V, T, P, S)$ be a grammar. A symbol X is useful if there is a derivation $S \overset{*}{\Rightarrow} \alpha \times \beta \overset{*}{\Rightarrow} w$ for some α, β and w, where w is in T^*. Otherwise X is useless symbol. There are two aspects to usefulness.

(a) Some terminal string must be derivable from X and

(b) X must occur in some string derivable from S.

These two conditions are not, sufficient to guarantee that X is useful, since X may occur only in sentential forms that contain a variable from which no terminal string can be derived.

Q.16. How can we remove null production from context-free grammar ?

Ans. A production of the form '$A \to \wedge$', where A is a variable ($A \in V$) is called a null production. These productions are just used to erase. A context-free grammar may have these type of productions.

Example – Consider a CFG 'G' whose productions are

$$S \to aS \mid aA \mid \wedge$$
$$A \to \wedge$$

So we have two null productions viz. $S \to \wedge$ and $A \to \wedge$. We can delete $A \to \wedge$ providing erase A whenever it occurs in the course of derivation of a terminal string.

So, we can replace $S \to aA$ by $S \to a$

Now if G_1 is the grammar whose productions are

$$S \to aS \mid a \mid \wedge$$

then $$L(G_1) = L(G) = \{a^n / n \geq 0\}$$

Q.17. What are unit productions ? How can we remove these type of productions from a CFG ?

Ans. The productions in the form of

$$A \to B, \text{ where } A, B \in V_N$$

are called unit productions.

A context-free grammar may also have these type of productions.

Example – Consider a CFG whose productions are

$$S \to A$$

$$A \to B$$
$$B \to C$$
$$C \to a$$

It is easy to see that $L(G) = \{a\}$

Now we can replace all non-terminals on right hand side of a unit production by a terminal value i.e., we have

$$C \to a$$

then $$B \to a$$

then $$A \to a$$

and $$S \to a$$

i.e. if $S \to a$ is G_1, then

$$L(G_1) = \{a\}$$

i.e. $$L(G_1) = L(G) = \{a\}$$

Q.18. Define the normal forms for context-free grammars.

Or

Define CNF and GNF.

Or ***(R.G.P.V., June 2010)***

Define the following –
(i) Chomsky normal form (ii) Greibach normal form.
(R.G.P.V., Dec. 2002, 2011)

Or

Explain with example Chomsky Normal Form and GNF forms ?
(R.G.P.V., June 2017)

Ans. In a context-free grammar, the R.H.S. of a production can be any string of variables and terminals. When the productions in G satisfy certain restrictions, then G is said to be in a ***'normal form'***. Among several 'normal forms', we explain two 'normal forms', i.e., Chomsky normal form and Greibach normal form.

(i) Chomsky Normal Form (CNF) – In the Chomsky normal form (CNF), we have restriction on the length of R.H.S. and the nature of symbols in the R.H.S. of productions.

A context-free grammar is in ***Chomsky normal form*** if all productions are of the form –

$$A \to BC \quad \text{or} \quad A \to a$$

where A, B, C, are in V and a is in T.

For example – the grammar

$$S \to AS \mid a$$
$$A \to SA \mid b$$

is in Chomsky normal form. While the grammar

$$S \to AS \mid AAS$$
$$A \to SA \mid aa$$

is not in Chomsky normal form because both productions $S \to AAS$ and $A \to aa$ violate the conditions of the definition, given above.

We can also say that –

"Any context-free grammar G with $\in \notin L(G)$ has an equivalent grammar G' in Chomsky normal form."

(ii) Greibach Normal Form (GNF) – Greibach normal form (GNF) is another normal form quite useful in some proofs and constructions. A context-free grammar generating the set accepted by a pushdown automaton is in Greibach normal form.

Here, restrictions are put, not on the length of the right sides of a production, but on the positions of the terminals and variables. Arguments which justify Greibach normal form are little complicated and not very transparent. Also, constructing a grammar in Greibach normal form equivalent to any context-free grammar is tedious. Nevertheless, Greibach normal form has many theoretical and practical consequences.

A context-free grammar is said to be in Greibach normal form if all of its productions have the following form –

$$A \to aA_1$$

where $a \in T$ and $A_1 \in V^*$.

For example, the following grammar –

$$S \to AB$$
$$A \to aA|bB|b,$$
$$B \to b$$

is not in Greibach normal form. But, using the substitution, we immediately get the equivalent grammar as follows –

$$S \to aAB|bBB|bB.$$
$$A \to aA \mid bB \mid b.$$
$$B \to b$$

This is in Greibach normal form and we can also say that –

"For every context-free grammar G, with $\in \notin L(G)$, there exists an equivalent grammar G' in Greibach normal form".

Q.19. Write short note on CNF. ***(R.G.P.V., Dec. 2004)***

Or

Write short note on Chomsky normal form. ***(R.G.P.V., June 2003)***

Ans. Refer to Q.18 (i).

Q.20. Write short note on Greibach normal form.(R.G.P.V., Dec. 2017)

Ans. Refer to Q.18 (ii).

Q.21. What are steps to convert the context-free grammar to Chomsky normal form ? Explain each step with suitable examples. (R.G.P.V., June 2004)

Ans. As we know for every context-free grammar, there is an equivalent grammar in Chomsky normal form. Following procedure gives the method of constructing a grammar in CNF equivalent to a context-free grammar –

(i) Elimination of Null Productions and Unit Productions – We apply theorem to eliminate null productions. We then apply theorem to the resulting grammar to eliminate chain productions. Let the grammar thus obtained be $G = (V, T, P, S)$.

(ii) Elimination of Terminals on R.H.S. – We define $G_1 = (V_1, T, P_1, S)$, where P_1 and V_1 are constructed as follows –

(a) All the productions in P of the form $A \to a$ or $A \to BC$ are included in P_1. All the variables in V are included in V_1.

(b) Consider $A \to X_1X_2.....X_n$, with some terminal on R.H.S. If X_i is a terminal, say a_i, add a new variable C_{ai} to V' and $C_{ai} \to a_i$ to P_1. In production $A \to X_1X_2 .....X_n$ every terminal on R.H.S. is replaced by the corresponding new variable and the variables on the R.H.S. are retained. The resulting production is added to P_1. Thus we get $G_1 = (V_1, T, P_1, S)$.

(iii) Restricting the Number of Variables on R.H.S. – For any production in P_1, the R.H.S. consists of either a single terminal (or Λ in $S \to \Lambda$) or two or more variables. We define $G_2 = (V_2, T, P_2, S)$ as follows –

(a) All productions in P_1 are added to P_2 if they are in the required form. All the variables in V_1 are added to V_2.

(b) Consider $A \to A_1A_2.....A_m$, where $m \geq 3$. We introduce new productions $A \to A_1C_1$, $C_1 \to A_2C_2$,....., $C_{m-2} \to A_{m-1}\, A_m$, and new variables C_1, C_2, ..., C_{m-2}. These are added to P_2 and V_2, respectively.

Thus we get G_2 in Chomsky normal form.

Q.22. Write an algorithm to convert a grammar to Greibach normal form (GNF). ***(R.G.P.V., June 2004)***

Or

Explain GNF conversion steps. ***(R.G.P.V., June 2016)***

Ans. Following procedure gives the method of constructing a grammar in GNF equivalent to a context-free grammar.

Construction of G – (i) We eliminate null productions and then construct a grammar G in Chomsky normal form. We renames the variables as A_1, A_2, ..., A_n with $S = A_1$. We write G as $(\{A_1, A_2, ..., A_n\}, T, P, A_1)$.

(ii) To get the productions in the form $A_i \to a\gamma$ or $A_i \to A_j\gamma$, where $j > i$, convert the A_i-productions ($i = 1, 2, \ldots\ldots, n-1$) to the form $A_i \to A_j\gamma$, such that $j > i$. By the principle of induction, the construction can be carried out for $i = 1, 2, \ldots\ldots, n$. Thus for $i = 1, 2, \ldots\ldots, n-1$, any A_i-production is of the form $A_i \to A_j\gamma$, where $j > i$ or $A_i \to a\gamma'$. Any A_n-production is of the form $A_n \to A_n\gamma$ or $A_n \to a\gamma'$.

(iii) Convert A_n-productions to the form $A_n \to a\gamma$. Here, productions of the form $A_n \to A_n\gamma$ are eliminated using lemma 2. The resulting A_n-productions are of the form $A_n \to a\gamma$.

(iv) Modify A_i-productions to the form $A_i \to a\gamma$ for $i = 1, 2, \ldots, n-1$. At the of step (iii), the A_n-productions are of the form $A_n \to a\gamma$. The A_{n-1} productions are of the form $A_{n-1} \to a\gamma'$ or $A_{n-1} \to A_n\gamma$. By applying lemma 1 we eliminate productions of the form $A_{n-1} \to A_n\gamma$. The resulting A_{n-1}-productions are in the required form. We repeat the construction by considering $A_{n-2}, A_{n-3}, \ldots, A_1$.

(v) Modify Z_i-productions. Every time we apply lemma 2, we get a new variable (we take it as Z_i when we apply the lemma for A_i-productions.) The Z_i-productions are of the form $Z_i \to \alpha Z_i$ or $Z_i \to \alpha$ (where α is obtained from $A_i \to A_i\alpha$), and hence of the form $Z_i \to a\gamma$ or $Z_i \to A_k\gamma$ for some k. At the end of step (iv), the R.H.S. of any A_k-production starts with a terminal. So we can apply lemma 1 to eliminate $Z_i \to A_k\gamma$. Thus at the end of step (v), we get an equivalent grammar G_1 in GNF.

NUMERICAL PROBLEMS

Prob.1. Construct a CFG generating all integers (with sign). **(R.G.P.V., June 2010)**

Sol. $G = (V, T, P, S)$

where, $V = \{S, \langle\text{sign}\rangle, \langle\text{digit}\rangle, \langle\text{integer}\rangle\}$

$T = \{0, 1, 2, 3, \ldots\ldots, 9, +, -\}$

P consists of $S \to \langle\text{sign}\rangle\ \langle\text{integer}\rangle$, $\langle\text{sign}\rangle \to + \mid -$

$\langle\text{integer}\rangle \to \langle\text{digit}\rangle\ \langle\text{integer}\rangle \mid \langle\text{digit}\rangle$

$\langle\text{digit}\rangle \to 0|1|2|\ldots\ldots|9$

$L(G)$ = the set of all integers. For example, the derivation of -15 can be obtained as follows –

$$S \Rightarrow \langle\text{sign}\rangle\ \langle\text{integer}\rangle \Rightarrow -\ \langle\text{integer}\rangle$$
$$\Rightarrow -\langle\text{digit}\rangle\ \langle\text{integer}\rangle \Rightarrow -1\ \langle\text{integer}\rangle \Rightarrow -1\ \langle\text{digit}\rangle$$
$$\Rightarrow -15.$$

Prob.2. Design context free grammars for the following languages –
The set $\{0^n 1^n | n \geq 1\}$ i.e., the set of all strings of one or more 0's followed by an equal number of 1's. **(R.G.P.V., Dec. 2008)**

Write the CFG for the following language – ***Or***
$L = \{0^n 1^n / n \geq 1\}$ **(R.G.P.V., Dec. 2013)**

Sol. Required CFG can be defined as –

$$L(G) = \{(S), (0, 1), P, S\}$$

where p consists of

$$S \to 0S1/01$$

Prob.3. Write CFG for set of all words consisting of an equal number of a's and b's.

For example : a a b b, a b a b, a b b b a a. **(R.G.P.V., Dec. 2012)**

Sol. Required CFG can be defined as follows –

$$L(G) = \{(S), (a, b), P, S\}$$

where P consists of,

$$S \to aSbS/bSaS/\in$$

Prob.4. Find context-free grammar for the following language (with $n \geq 0$, $m \geq 0$, $k \geq 0$) –

$$L = \{a^n b^m c^k : n = m \text{ or } m \leq k\}$$

Sol. We construct G as follows – **(R.G.P.V., June 2005, 2008)**

$$G = (\{S, A, B\}, \{a, b, c\}, P, S)$$

where P consists of

$$S \to aAbC/DbBc/bBc/aAb/abC/ab/\in$$
$$A \to aAb/ab$$
$$B \to bBc/cC/c$$
$$C \to cC/c$$
$$D \to aD/a$$

Prob.5. Construct CFG for the following set – **Ans.**

(i) $\{a^{2n} bc \mid n \geq 1\}$

(ii) $\{a^n b^m c^m d^n \mid m, n \geq 1\}$.

Sol. (i) $\{a^{2n} bc \mid n \geq 1\}$ **(R.G.P.V., June 2007)**

We construct G as follows –

$$G = (\{S, A\}, \{a, b, c\}, P, S\}$$

where, P consists of

$$S \to aaAbc/aabc$$
$$A \to aaA/aa$$

(ii) $\{a^n b^m c^m d^n \mid m, n \geq 1\}$

We construct G as follows –

$$G = \{(S, A), (a, b, c, d), P, S\}$$

The required productions are as follows –

$$S \rightarrow aSd/aAd$$
$$A \rightarrow bAc/bc$$

Prob.6. Write CFG for the following sets –

(i) $L = \{a^n b^m c^k : n = m \text{ or } m \leq k, n \geq 0, m \geq 0, k \geq 0\}$

(ii) $L = \{a^{2n} bc \mid n \geq 1\}$. ***(R.G.P.V., Dec. 2009)***

Sol. (i) Refer to Prob.4.

(ii) Refer to Prob.5 (i).

Prob.7. Write CFG for the following sets – ***(R.G.P.V., Dec. 2006)***

$$L = \{a^n b^m c^m a^n \mid n, m \geq 1\}$$

Sol. Required CFG can be defined as –

$$L(G) = \{(S, A), (a, b, c), P, S\}$$

where, P consists of,

$$S \rightarrow aSa/aAa$$
$$A \rightarrow bAc/bc$$

Prob.8. Construct CFG for generating set –

$$L = \{a^n b^{2n} c^m d^{3m} \mid n, m \geq 1\}$$ ***(R.G.P.V., June 2008)***

Sol. The required CFG is defined as

$$G = (\{S, A, B\}, \{a, b, c, d\}, P, S)$$

where P consist of

$$S \rightarrow AB$$
$$A \rightarrow aAbb/abb$$
$$B \rightarrow cBddd/cddd$$

Ans.

Prob.9. Write a CFG to generate the language –

$$L = \{0^m 1^n 0^{m+n} \mid m, n \geq 1\}.$$ ***(R.G.P.V., June 2009)***

Sol. The productions of the grammar are –

$$S = 0A0$$
$$A \rightarrow 0A0|1B0|10$$
$$B \rightarrow 1B0|10$$

Because we want to get block of m 0's, followed by a block of n 1's, followed by a block of $m + n$ 0's generated, and it is required that m and n both are greater than or equal to 1. This requires that for every 0 in the first block, a corresponding 0 must be generated in the last block, as well as for every 1 in the second block, a corresponding 0 must be generated in the last block.

Therefore, the grammar G is –

$$G = (\{S, A, B\}, \{0, 1\}, P, S),$$ where the members of P are given above.

Prob.10. Construct a context free grammar for the languages –

$L(G_1) = \{a^i b^{2i}, i > 0\}$ and $L(G_2) = (a^n b a^n, n > 0\}$ ***(R.G.P.V., Dec. 2010)***

Sol. (i) Let $G_1 = (\{S\}, \{a, b\}, P, S)$

where, P consist of

$$S \rightarrow aSbb/abb$$

This is the required grammar.

(ii) Let, $G_2 = (\{S, A\}, (a, b), P, S)$, where, P consists of $S \rightarrow aAa$, $A \rightarrow aAa|b$. This is the required grammar.

Prob.11. Construct the context-free grammar (CFG) equivalent to a regular expression –

(011 + 1)* (01)*

Or ***(R.G.P.V., Dec. 2005)***

Give CFG for R.E. (011 + 1)* (01)*. ***(R.G.P.V., June 2016)***

Sol. Starting with grammars for the languages {0} and {1}; however, we may simplify the process by taking some obvious shortcuts. The productions

$$A \rightarrow 011 \mid 1$$

generate the language {011, 1}. By grammar G^* generating L^*, we can use the productions

$$B \rightarrow AB|\in$$
$$A \rightarrow 011 \mid 1$$

to generate the language {011, 1}*, using the start symbol B. Similarly, we can use

$$C \rightarrow DC \mid \in$$
$$D \rightarrow 01$$

to derive {01}* from the start symbol C. Finally, we generate the concatenation of the two languages by adding the production $S \rightarrow BC$. The final grammar has start symbol S, auxiliary variables A, B, C, and D and productions

$$S \rightarrow BC$$
$$B \rightarrow AB \mid \in$$
$$A \rightarrow 011 \mid 1$$
$$C \rightarrow DC \mid \in$$
$$D \rightarrow 01$$

Prob.12. Find the CFG for the regular expression $(110 + 11)^*(10)^*$. **(R.G.P.V., Dec. 2014)**

Sol. Starting with grammars for the languages {0} and {1}; however, we may simplify the process by taking some obvious shortcuts. The productions

$$A \to 110 \mid 11$$

generate the language {110, 11}. By grammar G^* generating L^*, we can use the productions

$$B \to AB \mid \in$$
$$A \to 110 \mid 11$$

to generate the language $\{110, 11\}^*$, using the start symbol B. Similarly, we can use

$$C \to DC \mid \in$$
$$D \to 10$$

to derive $\{10\}^*$ from the start symbol C. Finally, we generate the concatenation of the two languages by adding the production $S \to BC$. The final grammar has start symbol S, auxiliary variables A, B, C, and D and productions

$$S \to BC$$
$$B \to AB \mid \in$$
$$A \to 110 \mid 11$$
$$C \to DC \mid \in$$
$$D \to 10$$

Prob.13. Find L(G) for the grammar $S \to aCa$, $C \to aCa/b$. **(R.G.P.V., Dec. 2016)**

Sol. Required $L(G)$ of

$$G = \{(S, C), (a, b), P, S\}$$

where, P consist of

$$S \to aCa$$
$$C \to aCa/b \text{ is defined as}$$

$$L(G) = \{a^n b a^n / n \geq 1\}$$ Ans.

Prob.14. Consider the grammar with productions –

$$S \to aaB,\ A \to bBb \mid \in,\ B \to Aa$$

Show that the string aabbabba is not in the language generated by this grammar. **(R.G.P.V., June 2005)**

Sol. $S \Rightarrow aaB \Rightarrow aaAa \Rightarrow aabBba$

$\Rightarrow aabAaba \Rightarrow aabbBbaba$

$\Rightarrow aabbAababa \Rightarrow aabbababa$

Since, the given string is *aabbabba*.

Hence, the string *aabbabba* is not in the language.

Prob.15. Find regular grammars for the following languages on (a, b).

(i) $L = \{w \mid n_a(w)$ and $n_b(w)$ are both even$\}$

(ii) $L = \{w \mid (n_a(w) - n_b(w)\}$ mod3 $= 1\}$

Sol. (i) $L = \{w \mid n_a(w)$ and $n_b(w)$ are both even$\}$ **(R.G.P.V., Dec. 2013)**

Required grammar is

$$G = (V, T, P, A)$$

where, $V = \{A, B, C, D\}$

$T = \{a, b\}$

and P is defined as

$$A \to aB/bD/\in$$
$$B \to aA/bC$$
$$C \to aD/bB$$
$$D \to bA/aC$$

(ii) $L = \{w \mid (n_a(w) - n_b(w))$ mod3 $= 1\}$

Required grammar is

$$G = (V, T, P, A)$$

where, $V = \{A, B, C, D, E\}$

$T = \{a, b\}$

and P is defined as

$$A \to aB/bC/a$$
$$B \to aC/bA$$
$$C \to aA/bB/b$$

Prob.16. Find the regular grammar for the language –

(i) $L = \{w \in \{a, b\}^* : w$ does not contain the substring aa$\}$

(ii) $L = \{a^n b^n : n \geq 0\}$

(R.G.P.V., Dec. 2010)

Sol. (i) Observe that whenever a a occurs it must be followed immediately by a b. Such a substring may be preceded and followed by an arbitrary number of b's. This suggests that the answer involves the repetition of strings of the form $b...bab....b$, that is, the language denoted by the regular expression $(b^*abb^*)^*$. However, the answer is still incomplete, since the strings ending in a or consisting of all b's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (b^*abb^*)^* (a + \in) + b^* (a + \in)$$

If we see L as the repetition of the strings b and ab, the shorter expression will be –

$$r = (b + ab)^* (a + \in)$$

The DFA for this regular expression is as follows –

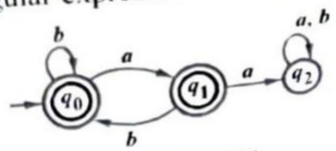


Fig. 3.3 DFA

Let $G = q(\{q_0, q_1, q_2\}, \{a, b\}, P, q_0)$, where P is given by

$$\left.\begin{array}{l} q_0 \to aq_1 \\ q_0 \to a \\ q_0 \to bq_0 \\ q_0 \to b \end{array}\right\}$$

$$\left.\begin{array}{l} q_1 \to aq_2 \\ q_1 \to bq_0 \\ q_1 \to b \end{array}\right\}$$

$$\left.\begin{array}{l} q_2 \to aq_2 \\ q_2 \to bq_2 \end{array}\right\}$$

G is the required grammar.

(ii) $L = \{a^n b^n : n \geq 0\}$ (Assume a as 0 and b as 1)

Refer to Prob.46 (ii) (Unit-II).

Since the language $L = \{a^n b^n : n \geq 0\}$ is not regular, so we can't generate its regular grammar.

Prob.17. Prove that the following is not regular language $\{0^n/n$ is perfect square$\}$. **(R.G.P.V., Dec. 2017)**

Sol. The given language is

$\{0^n/n$ is perfect square$\}$

where $n \geq 0$

$\because$ 0 to power n is 0, for all values of n.

$\therefore$ The number of strings generated by the language is

$\{0, 00, 000.....\}$

Hence, it is not an regular language.

Prob.18. Given a grammar, make non-deterministic finite automata and convert it to deterministic form –

$$S \to 0S \mid 1A \mid 1$$
$$A \to 0 \mid 0A \mid 1S$$

(R.G.P.V., June 2004)

Sol. Let $M = \{\{q_0, q_1, q_f\}, \{0, 1\}, \delta, q_0, \{q_f\}\}$, where q_0 and q_1 are corresponding to S and A, respectively, and q_f is the final state.

$S \to 0S$ production induces a transition from q_0 to q_0 with label 0. Similarly, $S \to 1A$, $A \to 0A$, and $A \to 1S$ induce transitions from q_0 to q_1 with label 1, from q_1 to q_1 with label 0, and from q_1 to q_0 with label 1, respectively. $S \to 1$ and $A \to 0$ induce transitions from q_0 to q_f with label 1 and from q_1 to q_f with label 0, respectively. Transition graph for this solution is given in fig. 3.4.

Transition table correspond to fig. 3.4 is given in table 3.1.

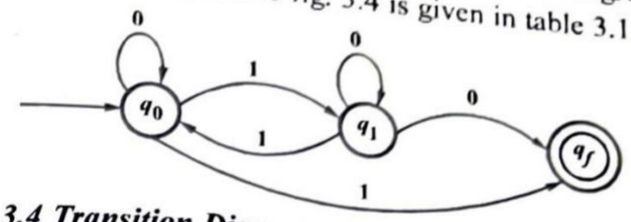


Fig. 3.4 Transition Diagram of NDFA for Given Example

Table 3.1 Transition Table of NDFA

State	Input	
	0	1
$\to q_0$	q_0	q_1, q_f
q_1	q_1, q_f	q_0
q_f	–	–

Table 3.2 Transition Table of DFA

State	Input	
	0	1
$[q_0]$	$[q_0]$	$[q_1, q_f]$
$[q_1, q_f]$	$[q_1, q_f]$	q_0

Transition table of DFA can be constructed as given in table 3.2.

Transition diagram correspond to table 3.2 is drawn in fig. 3.5.

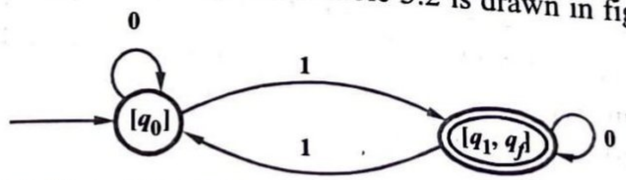


Fig. 3.5 Transition Diagram of DFA for Given Example

Prob.19. Consider the following productions –

$$S \to aAS \mid a$$
$$A \to SbA \mid SS \mid ba$$

Find out LMD, RMD, Parse tree for string aabbaa.

(R.G.P.V., June 2006)

Sol. **LMD (Leftmost Derivation) –**

$$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS$$
$$\Rightarrow aabbaS \Rightarrow aabbaa$$

RMD (Rightmost Derivation) –

$$S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa$$
$$\Rightarrow aSbbaa \Rightarrow aabbaa$$

Parse Tree –

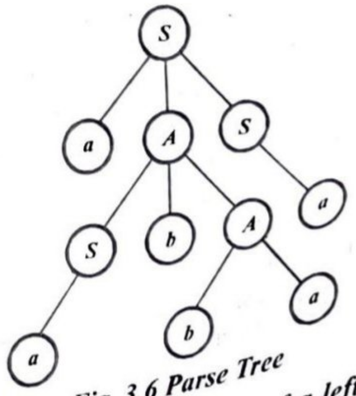


Fig. 3.6 Parse Tree

Prob.20. Derive the string "aabbabba" for leftmost derivation and rightmost derivation using a CFG given by –

$$S \to aB \mid bA$$
$$A \to a \mid aS \mid bAA$$
$$B \to b \mid bS \mid aBB$$

(R.G.P.V., Dec. 2011)

And also draw the derivation tree.

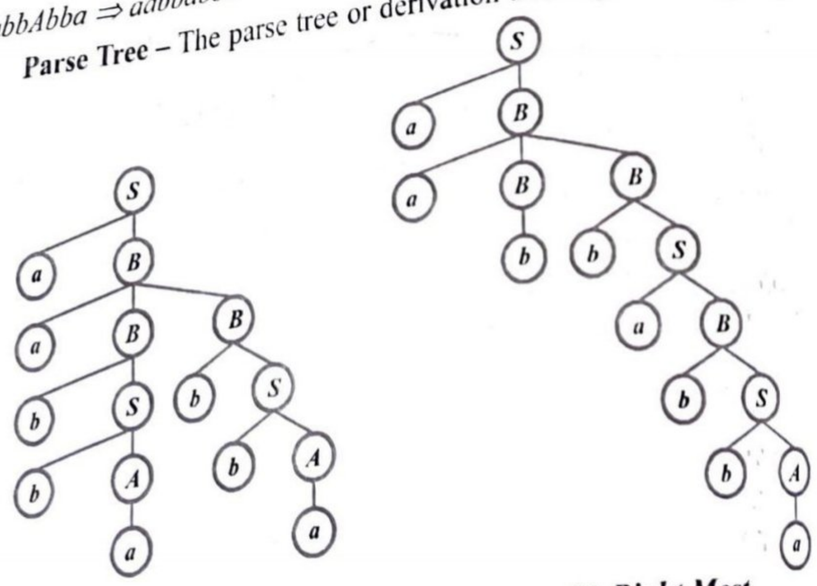
Sol. Leftmost Derivation –

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aabB \Rightarrow aabbS \Rightarrow aabbaB \Rightarrow aabbabS \Rightarrow aabbabbA \Rightarrow aabbabba$

Rightmost Derivation –

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaBbS \Rightarrow aaBbbA \Rightarrow aaBbba \Rightarrow aabSbba \Rightarrow aabbAbba \Rightarrow aabbabba$

Parse Tree – The parse tree or derivation tree is given in fig. 3.7.



(a) Left Most ***(b) Right Most***

Fig. 3.7 Derivation Tree

Prob.21. Consider the grammar –

$$S \to aS|aSbS|\in$$

This grammar is ambiguous. Show in particular that the string aab has two –

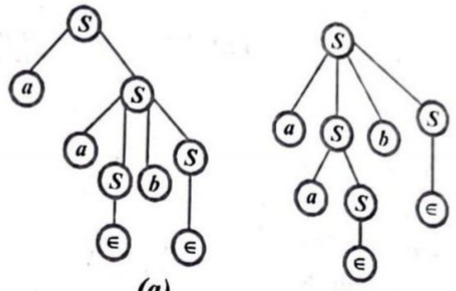
(i) Parse trees (ii) Leftmost derivations
(iii) Rightmost derivations.

(R.G.P.V., Dec. 2008)

Sol. (i) Parse Tree – Given that

$$S \to aS|aSbS|\in$$

This grammar is ambiguous. So that string "*aab*" can be generated from this grammar in two distinct ways, as shown in the following derivation trees or parse trees in fig. 3.8.



(a) ***(b)***

Fig. 3.8 Parse Trees

$S \to aS$	$S \to aSbS$
$S \to aaSbS$	$S \to aaSbS$
$S \to aab$	$S \to aab$

(ii) Leftmost Derivations – "*aab*" has two distinct leftmost derivations –

$S \Rightarrow aS \Rightarrow aaSbS \Rightarrow aab$

$S \Rightarrow aSbS \Rightarrow aaSbS \Rightarrow aab$

(iii) Rightmost Derivations – Also "*aab*" has two distinct rightmost derivations –

$S \Rightarrow aS \Rightarrow aaSbS \Rightarrow aab$

$S \to aSbS \Rightarrow aSb \Rightarrow aaSb \Rightarrow aab$

Prob.22. Let G be the grammar –

$$S \to aB \mid bA$$
$$A \to a \mid aS \mid bAA$$
$$B \to b \mid bS \mid aBB$$

For the string aaabbabbba find –

(i) Leftmost derivation ***(ii) Rightmost derivation***
(iii) Parse tree ***(iv) Is the grammar unambiguous ?***

(R.G.P.V., June 2004)

Or

Let G be the grammar.

$$S \rightarrow aB \mid bA$$
$$A \rightarrow a \mid aS \mid bAA$$
$$B \rightarrow b \mid bS \mid aBB$$

For the string aaabbabbba find a

(i) Leftmost derivation (ii) rightmost Derivation (iii) Parse tree.

(R.G.P.V., Dec. 2012)

Or

Let G be the grammar –

$$S \rightarrow aB \mid bA$$
$$A \rightarrow a \mid aS \mid bAA$$
$$B \rightarrow b \mid bS \mid aBB$$

For the string aaabbabbba, find leftmost and rightmost derivations.

(R.G.P.V., Dec. 2007)

Or

Consider the following CFG –

$$S \rightarrow aB \mid bA, \quad A \rightarrow a|aS|bAA, \quad B \rightarrow b|bS|aBB$$

For string "aaabbabbba", find –

(i) Leftmost derivation (ii) Rightmost derivation (iii) Parse tree.

(R.G.P.V., June 2007)

Sol. (i) Leftmost Derivation –

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaaBBB \Rightarrow aaabSBB \Rightarrow aaabbABB \Rightarrow aaabbaBB$
$\Rightarrow aaabbabB \Rightarrow aaabbabbS \Rightarrow aaabbabbbA \Rightarrow aaabbabbba$

(ii) Rightmost Derivation –

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaBbS \Rightarrow aaBbbA \Rightarrow aaBbba \Rightarrow aaaBBbba$
$\Rightarrow aaaBbbba \Rightarrow aaabSbbba \Rightarrow aaabbAbbba \Rightarrow aaabbabbba$

(iii) Parse Tree – The parse tree or derivation tree is given in fig. 3.9.

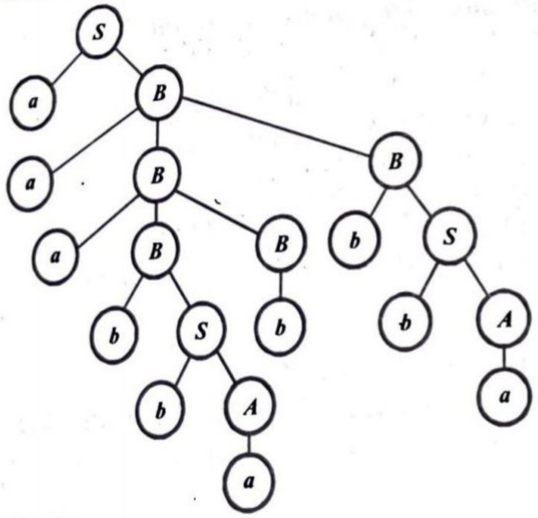
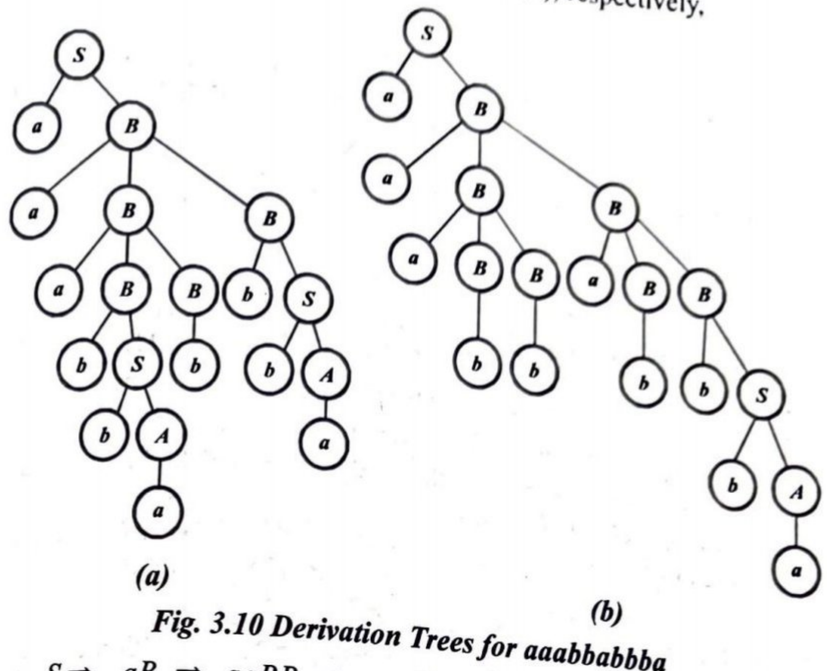


Fig. 3.9 Derivation Tree

(iv) Ambiguity – Here, we have more than one parse tree for the given string *aaabbabbba* as shown in fig. 3.10. The string has two different derivations as shown in fig. 3.10 (a) and 3.10 (b), respectively.



(a) ***(b)***

Fig. 3.10 Derivation Trees for aaabbabbba

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaaBBB \Rightarrow aaabSBB \Rightarrow aaabbABB$
$\Rightarrow aaabbaBB \Rightarrow aaabbabB \Rightarrow aaabbabbS \Rightarrow aaabbabbbA$
$\Rightarrow aaabbabbba$

and $S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaaBBB \Rightarrow aaabBB \Rightarrow aaabbB \Rightarrow aaabbaBB$
$\Rightarrow aaabbabB \Rightarrow aaabbabbS \Rightarrow aaabbabbbA \Rightarrow aaabbabbba$

So, the grammar is ambiguous.

Prob.23. Consider the following grammar –

$$S \rightarrow abSb \mid aAb \mid a$$
$$A \rightarrow bS \mid aAAb$$

For the string abababb, illustrate –

(i) LMD (ii) RMD (iii) Derivation tree

Also show that the given grammar is ambiguous.

(R.G.P.V., Nov. 2018)

Sol. (i) LMD (Leftmost Derivation) –

S ⇒ abSb ⇒ abaAbb ⇒ ababSbb ⇒ abababb

(ii) RMD (Rightmost Derivation) –

S ⇒ aAb ⇒ abSb ⇒ ababSbb ⇒ abababb

(iii) ***Derivation Tree*** –

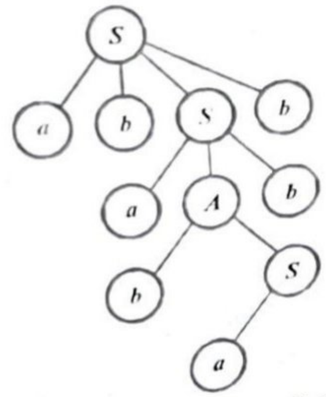
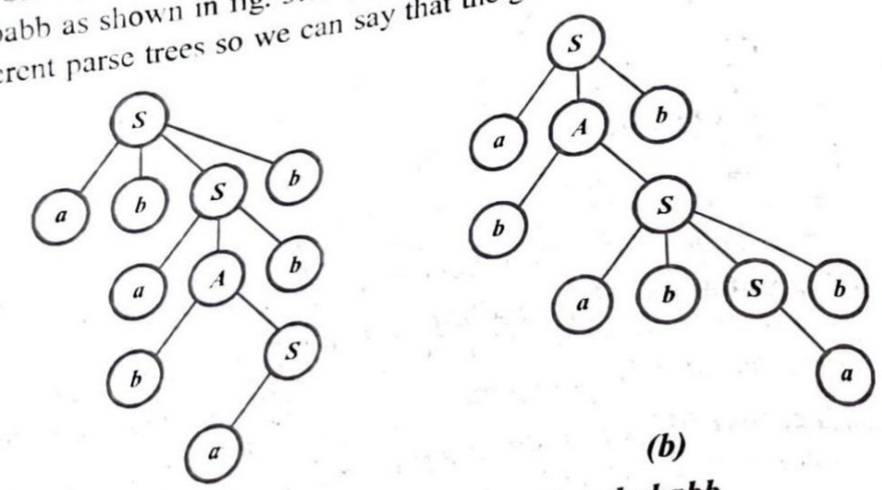


Fig. 3.11 Derivation Tree

Since, these are more than one parse tree are possible for the given string abababb as shown in fig. 3.12 (a) and fig. 3.12 (b). Since the string has two different parse trees so we can say that the given grammar is ambiguous.



(a) ***(b)***

Fig. 3.12 Derivation Trees for abababb

Prob.24. Prove that the given grammar is ambiguous.

(i) $S \rightarrow aSb/SS/\in$

(ii) $S \rightarrow AB/aaB,\ A \rightarrow a/Aa,\ B \rightarrow b$ ***(R.G.P.V., Dec. 2016)***

Sol. (i) $S \rightarrow aSb/SS/\in$ –

To prove that the grammar G is ambiguous, we have to find a string $u \in L(G)$, which is ambiguous. Consider $u = aabb \in L(G)$. Then we get two derivation trees for u, as shown in fig. 3.13. Thus G is ambiguous.

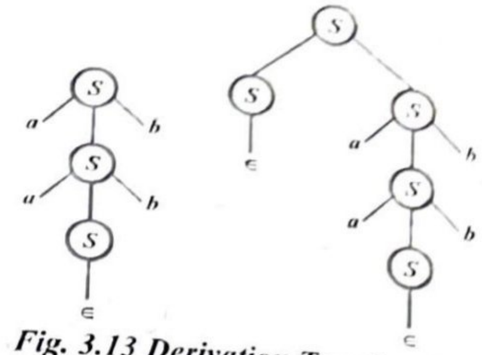
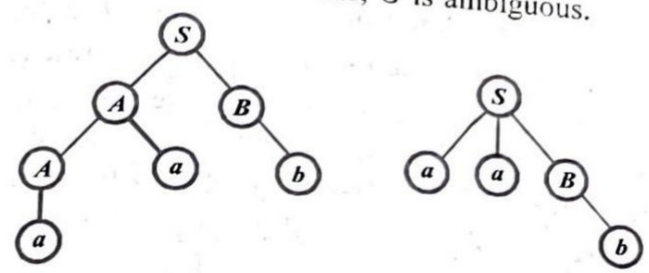


Fig. 3.13 Derivation Tree for abab

(ii) $S \rightarrow AB/aaB,\ A \rightarrow a/Aa,\ B \rightarrow b$ –

To prove that grammar G is ambiguous, we have to find a string $u \in L(G)$, which is ambiguous. Consider $u = aab \in L(G)$.Then we get two derivation trees for u, as shown in fig. 3.14. Thus, G is ambiguous.

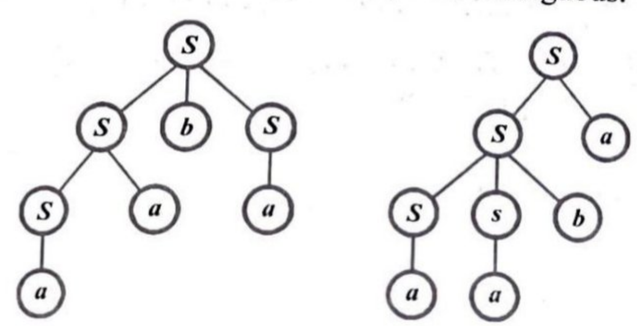


(a) ***(b)***

Fig. 3.14 Derivation Trees for aab

Prob.25. Show that the grammar $S \rightarrow a/Sa/bSS/SSb/SbS$ ***is ambiguous.***
(R.G.P.V., June 2011, 2015)

Sol. To prove that grammar G is ambiguous we have to find a string $u \in L(G)$, which is ambiguous. Consider $u = aaba \in L(G)$. Then we get two derivation trees for u (see fig. 3.15). Thus, G is ambiguous.



(a) ***(b)***

Fig. 3.15 Derivation Trees for aaba

Prob.26. Show that the grammar $S \to a|abSb|aAb$, $A \to bS \mid aAAb$ is ambiguous. **(R.G.P.V., Dec. 2014)**

Sol. To prove that grammar G is ambiguous we have to find a string $u \in L(G)$, which is ambiguous. Consider $u = abab \in L(G)$. Then we get two derivation trees for u as shown in fig. 3.16. Thus, G is ambiguous.

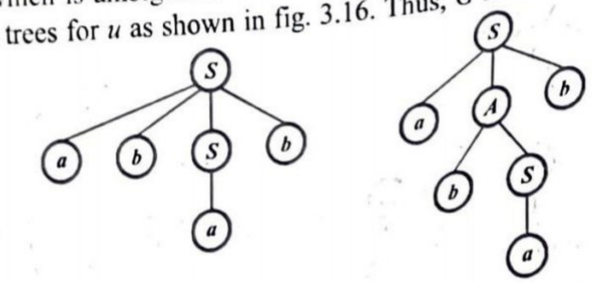


Fig. 3.16

Prob.27. Show that the following grammar is ambiguous **(R.G.P.V., Dec. 2015)**

$$S \to aSbS|bSaS|\epsilon$$

Sol. To prove that grammar G is ambiguous, we have to find a string $u \in L(G)$, which is ambiguous. Consider $u = abab \in L(G)$. Then, we get two derivation trees for u, as shown in fig. 3.17. Thus, G is ambiguous.

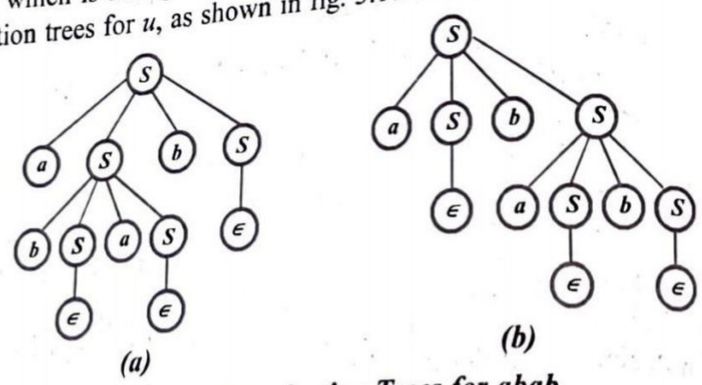


Fig. 3.17 Derivation Trees for abab

Prob.28. Show that the following grammar is ambiguous –

$$E \to E + E|E*E|2|3|4$$

(R.G.P.V., June 2010)

Sol. Now, consider the grammar

$$G = (V, T, E, P) \text{ with}$$
$$V = \{E\},\ T = \{2, 3, 4, +, *\},$$

and productions

$$E \to E + E$$
$$E \to E * E$$
$$E \to 2|3|4.$$

The strings $(2 + 3) * 4$ and $2 * 3 + 4$ are in $L(G)$. It is easy to see that this grammar generates a restricted subset of arithmetic expressions for FORTRAN and Pascal-like programming languages. The grammar is ambiguous. For instance, the string $2 + 3 * 4$ has two different derivation trees, as shown in fig. 3.18.

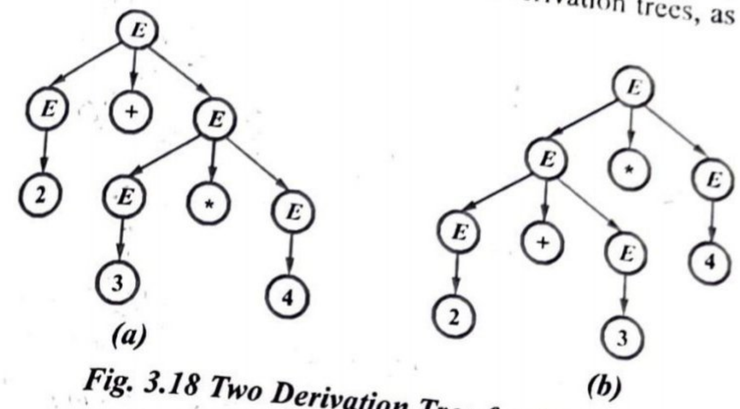


Fig. 3.18 Two Derivation Tree for 2 + 3 * 4

One way to resolve the ambiguity, as is done in programming manuals, to associate precedence rules with the operators + and *. Since * has higher precedence over + we would take fig. 3.18 (a) as the correct parsing as it indicates that 3*4 is a subexpression to be evaluated before performing the addition. It is better to rewrite the grammar so that only parsing is possible.

Prob.29. What do you mean by ambiguity ? Show that the grammar

$$S \to aB \mid ab$$
$$A \to a$$
$$B \to C \mid b$$
$$C \to D$$
$$D \to E$$
$$E \to a$$

is ambiguous.

Sol. **Ambiguity** – Refer to Q.12. **(R.G.P.V., Dec. 2010)**

Problem – The given grammar G is –

$$G = (\{S, A, B, C, D, E\}, \{a, b\}, P, S)$$

where, P consists of

$$S \to aB \mid ab$$
$$A \to a$$
$$B \to C \mid b$$
$$C \to D$$
$$D \to E$$
$$E \to a$$

We have two derivation trees for ab given in fig. 3.19. Therefore, the grammar is ambiguous.

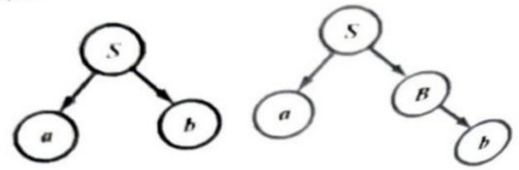


Fig. 3.19 Derivation Trees for ab

Prob.30. Check whether the given grammar is ambiguous or not –

$S \to iC + S$
$S \to iC + ScS$
$S \to a$
$C \to b$

(R.G.P.V., Dec. 2011)

Sol. To check whether the given grammar G is ambiguous, we have to find a string $u \in L(G)$, which is ambiguous. Consider $u = ib + ib + aca$. Then, we get two derivation trees for u (see fig. 3.20). Thus, G is ambiguous.

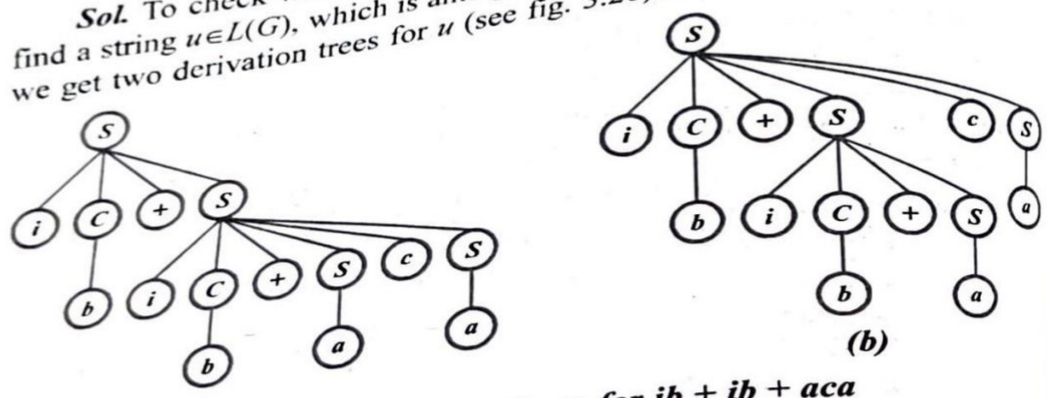


Fig. 3.20 Derivation Trees for ib + ib + aca

Prob.31. Remove the $\in$ production from the following CFG –

$S \to XYX$
$X \to 0X/\in$
$Y \to 1Y/\in$

(R.G.P.V., June 2011)

Sol. We find that the nullable variables are A, B, C. Then, we construct the grammar in the following way –

$S \to XYX \mid YX \mid XX \mid XY \mid X \mid Y$
$X \to 0X \mid 0$
$Y \to 1Y \mid 1$

Prob.32. Eliminate $\in$-productions from a given CFG –

$S \to ABCBCDA$
$A \to CD$

$B \to Cb$
$C \to a \mid \in$
$D \to bD \mid \in$

(R.G.P.V., Dec. 2005)

Sol. First remove the $\in$-production from C and D.

After removing the $\in$-production from C and D, the resultant CFG is

$S \to ABCBCDA/ABBCDA/ABCBDA/ABBDA/ABCBCA/ABBCA/ABCBA/ABBA$

$A \to CD/C/D/\in$
$B \to Cb/b$
$C \to a$
$D \to bD/b$

Now, remove $\in$-production from A

$S \to ABCBCDA/ABBCDA/ABCBDA/ABBDA/ABCBCA/ABBCA/ABCBA/ABBA/BCBCDA/ABCBCD/BCBCD/BBCDA/ABBCD/BBCD/BCBDA/ABCBD/BCBD/BBDA/ABBD/BBD/BCBCA/ABCBC/BCBC/BBCA/ABBC/BBC/BCBA/ABCB/BCB/BBA/ABB/BB$

$A \to CD/C/D$
$B \to Cb/b$
$C \to a$
$D \to bD/b$

This is the required grammar without $\in$-production.

Prob.33. Remove all unit-productions from –

$S \to Aa|B,\ B \to A|bb,\ A \to a|bc|B$

Sol. The dependency graph for the unit productions is given as in fig. 3.21.

Fig. 3.21

We observe from it that $S \overset{*}{\Rightarrow} A$, $S \overset{*}{\Rightarrow} B$, $B \overset{*}{\Rightarrow} A$, and $A \overset{*}{\Rightarrow} B$. Hence, we add to the original non-unit productions –

$S \to Aa,$
$A \to a|bc,$
$B \to bb,$

the new rules –

$S \to a|bc|bb,\quad A \to bb,\quad B \to a|bc,$

to get the original grammar as follows –

$S \to a|bc|bb|Aa,\quad A \to a|bb|bc,\ B \to a|bb|bc.$

This is the required grammar without unit productions.

Prob.34. What do you mean by useless production ? Consider the grammar
$G = (V, T, P, S)$ where V, T, P, S are given as –
$V = \{S, A, B, C, E\}$
$T = \{a, b, c\}$
$S = \{S\}$ and
P consists of
$S \to AB$
$A \to a$
$B \to b$
$B \to C$
$E \to c$
Eliminate useless symbols and productions from the above grammar.
(R.G.P.V., Dec. 2015)

Sol. A production rule not affecting the language is called useless production. One invariably wants to remove productions from a grammar that can never take part in any derivation. For example, in the grammar whose entire production set is

$$S \to aSb \mid \in \mid A$$
$$A \to aA$$

the production $S \to A$ clearly plays no role, as a cannot be transformed into a terminal string. While A can occur in a string derived from S, this can never lead to a sentence. Removing this production leaves the language unaffected and is a simplification by any definition.

Identify the production that lead to terminal string in the given grammar

i.e.,
$$S \to AB$$
$$A \to a$$
$$B \to b$$
$$E \to c$$

$B \to C$ does not lead to a terminal string, so remove this production.

Eliminate the variables that cannot be reachable from the start symbols. Hence the variable E can be removed as it is not possible to reach E from S. Thus, the resulting productions take the form

$$S \to AB$$
$$A \to a$$
$$B \to b$$

Prob.35. Find a CFG with no useless symbols equivalent to –
$S \to AB|CA$, $B \to BC|AB$
$A \to a$, $C \to aB|b$
(R.G.P.V., June 2003)

Sol. Since, all the productions of grammar are reachable so skip the step-1 and continue to step-2.

Check whether the production forms any infinite loop –

Here, production $B \to BC/AB$ forms an infinite loop i.e. the production will never terminate.

So, remove the production $B \to BC/AB$ and all the productions which contains B.

The resultant grammar is

$$S \to CA$$
$$A \to a$$
$$C \to b$$

This is the required grammar with no useless symbols.

Prob.36. Eliminate unit, useless and $\in$-productions from the grammar
$S \to aA/aBB$, $A \to aaA/\in$, $B \to bB/bbC$, $C \to B$.
(R.G.P.V., Dec. 2016)

Sol. The given grammar is

$$S \to aA/aBB$$
$$A \to aaA/\in$$
$$B \to bB/bbC$$
$$C \to B$$

(i) Elimination of unit production

Since $C \to B$ is unit production

So replace B with bB/bbC in $C \to B$ i.e.

$$S \to aA/aBB$$
$$A \to aaA/\in$$
$$B \to bB/bbC$$
$$C \to bB/bbC$$

(ii) Elimination of useless production

Here, $B \to bB/bbC$ and

$C \to bB/bbC$ both are making an infinite loop (i.e. these equations will never ever end).

So after removing these equations and equation that contains B & C variable the reduced grammar will be

$$S \to aA$$
$$A \to aaA/\in$$

(iii) Elimination of $\in$-production –

Placing $\in$ in place of A in RHS of every equation which contains A

$S \to aA/a$

$A \to aaA/aa$

Hence, after elimination of unit, useless and $\in$-production, the reduced grammar is –

$S \to aA/a$

$A \to aaA/aa$

Prob.37. What do we eliminate during simplification of a context-free grammar (CFG) ? Simplify the following CFG –

$S \to aA \mid aBB$

$A \to aAA \mid \in$

$B \to bB \mid bbC$

$C \to B$

(R.G.P.V., Nov. 2018)

Sol. We eliminate symbols and productions in G which are not useful for derivation of sentences during the simplification of a context-free grammar. Now we shall simplify the given grammar –

$S \to aA \mid aBB$

$A \to aAA \mid \in$

$B \to bB \mid bbC$

$C \to B$

(i) Elimination of Unit Production –

Since $C \to B$ is a production

So we can replace $B \to$ with $bB \mid bbC$ in $C \to B$ i.e.

$S \to aA \mid aBB$

$A \to aAA \mid \in$

$B \to bB \mid bbC$

$C \to bB \mid bbC$

(ii) Elimination of Useless Production –

Here, $B \to bB \mid bbC$ and

$C \to bB \mid bbC$

both are making an infinite loop, so after removing these productions and productions that contain B and C variable, the reduced grammar will be

$S \to aA$

$A \to aAA \mid \in$

(iii) Elimination of Null Production –

Now placing $\in$ in place of A in R.H.S. of every production, the reduced grammar is –

$S \to aA \mid a$

$A \to aAA \mid a$

Hence, after elimination the simplified grammar is –

$S \to aA \mid a$

$A \to aAA \mid a$

Prob.38. Simplify the given grammar –

$S \to aSB \mid aA \mid bB$

$A \to aA \mid \in$

$B \to bB \mid \in$

Sol. The nullable productions are –

$A \to \in$ and $B \to \in$

(R.G.P.V., June 2009)

To eliminate $A \to \in$ from the above grammar, we replace A on the right side of the productions.

$S \to aA$

$A \to aA$

Replace each occurrence of A by $\in$ in each of these productions to obtain the non $\in$-productions, to be added to the grammar, the list of these productions is –

$S \to a$

$A \to a$

Add these productions to the grammar and eliminate $A \to \in$ from the grammar to get the following grammar –

$S \to aSB \mid aA \mid bB \mid a$

$A \to aA \mid a$

$B \to bB \mid \in$

To eliminate $B \to \in$ from the grammar, the non $\in$-productions to be added are obtained as follows –

List of the productions containing B on right side is –

$S \to aSB \mid bB$

$B \to bB$

Replace each occurrence of B by $\in$ is each of these productions to obtain the non $\in$-productions, to be added to the grammar, the list of these productions is –

$S \to aS \mid b$

$B \to b$

Add these productions to the grammar and eliminate $B \to \in$ from the grammar to obtain the following grammar –

$S \to aSB \mid aA \mid bB \mid a \mid aS \mid b$

$A \to aA \mid a$

$B \to bB \mid b$

Prob.39. Show that the following two grammars are equivalent –

Grammar-1	***Grammar-2***
$S \to abAB \mid ba$	$S \to abAaA \mid abAbb \mid ba$
$A \to aaa$	$A \to aaa$
$B \to aA \mid bb$	

(R.G.P.V., Dec. 2009)

Sol. Two grammars are equivalent if the language generated by them are equivalent.

So, language generated by Grammar-1 –

$L(G_1)$ = set of all possible strings generated by grammar 1.

$S \Rightarrow abAB \Rightarrow abaaaB \Rightarrow abaaaaA \Rightarrow abaaaaaaa$

$S \Rightarrow abAB \Rightarrow abaaaB \Rightarrow abaaabb$

$S \Rightarrow ba$

So, $L(G_1) = \{abaaaaaaa, abaaabb, ba\}$

Now for $L(G_2)$

$S \Rightarrow abAaA \Rightarrow abaaaaaaa$

$S \Rightarrow abAbb \Rightarrow abaaabb$

$S \Rightarrow ba$

So, $L(G_2) = \{abaaaaaaa, abaaabb, ba\}$

Since $L(G_1)$ and $L(G_2)$ are equivalent.

Hence, their corresponding grammar i.e. Grammar-1 and Grammar-2 are equivalent. **Hence showed**

Prob.40. Construct the reduced grammar equivalent to grammar –

$S \to aAa$

$A \to Sb \mid bCC \mid DaA$

$C \to abb \mid DD$

$E \to aC$

$D \to aDA$

(R.G.P.V., Dec. 2014)

Sol. Since there is no $\in$-production in grammar, so start reducing the grammar from second step.

Remove the Useless Production –

(i) Since in the grammar $E \to aC$ can't be reachable. So remove this production.

(ii) The production $D \to aDA$ forms an infinite loop of production, so remove this production also.

After removing the production of $E \to aC$ and $D \to aDA$ the grammar is-

$S \to aAa$

$A \to Sb/bCC$

$C \to abb$

This is required reduced grammar.

Prob.41. Convert the following CFG into an equivalent CFG in Chomsky normal form –

$A \to BAB \mid B \mid \in$

$B \to 00 \mid \in$

(R.G.P.V., Dec. 2007)

Sol. Given CFG is –

$A \to BAB \mid B \mid \in$

$B \to 00 \mid \in$

Let us go through each step of conversion to Chomsky normal form as –

Step 1 – Elimination of $\in$-production.

$A \to BAB \mid B \mid AB \mid BB \mid BA$

$B \to 00$

Step 2 – Eliminating unit productions. Here we may simply add the production $A \to 00$ and delete $A \to B$.

Step 3 – Restricting the right sides of productions to single terminals or string of two or more variables.

$A \to ABA \mid AB \mid BB \mid BA \mid X_0X_0$

$B \to X_0X_0$

$X_0 \to 0$

The final Chomsky normal form is as

$A \to AT_1 \qquad T_1 \to BA$

$A \to AB \mid BB \mid BA \mid X_0X_0$

$B \to X_0X_0$

$X_0 \to 0$

Prob.42. Convert the following CFG to CNF

$S \to ABA$

$A \to aA \mid \in$

$B \to bB \mid \in$

(R.G.P.V., Dec. 2012, 2014, June 2015)

Or

Find the CNF (Chomsky Normal Form) for the following CFG –

$S \to ABA$

$A \to aA \mid \in$

$B \to bB \mid \in$

(R.G.P.V., Nov. 2019)

Sol. **Step 1** – Write all the productions separately

$S \to ABA$

$A \to aA$

$A \to \in$

$B \to bB$

$B \to \in$

Step 2 – Elimination of $\in$ productions

$S \to ABA/BA/AA/AB/A/B$

$A \to aA/a$

$B \to bB/b$

Step 3 – Elimination of unit productions

$$S \to ABA/BA/AA/AB/aA/bB/a/b$$
$$A \to aA/a$$
$$B \to bB/b$$

Step 4 – Restricting the right sides of productions to single terminals or string of two or more variables.

$$S \to ABA/BA/AA/AB/C_aA/C_bB/a/b$$
$$A \to C_aA/a$$
$$B \to C_bB/b$$
$$C_a \to a$$
$$C_b \to b$$

The final Chomsky normal form is as

$$S \to AT_1/AA/AB/BA/C_aA/C_bB/a/b$$
$$T_1 \to BA$$
$$A \to C_aA/a$$
$$B \to C_bB/b$$
$$C_a \to a$$
$$C_b \to b$$

Prob.43. Convert the following CFG into CNF –

$$S \to \sim S \mid [S \supset S] \mid p/q$$

(R.G.P.V., June 2010)

Sol. Since the given grammar has no unit or null productions, we omit step 1 and proceed to step 2.

Step 2 – Let $G_1 = (V_N, T, P_1, S)$, where P_1 and V_N are constructed as follows –

(i) $S \to p|q$ are added to P_1.

(ii) $S \to \sim S$ induces $S \to AS$ and $A \to \sim$

(iii) $S \to [S \supset S]$ induces $S \to BSCSD$, $B \to [$, $C \to \supset$, $D \to]$

$$V_N = \{S, A, B, C, D\}$$

Step 3 – P_1 consists of $S \to p|q$, $S \to AS$, $A \to \sim$, $B \to [$, $C \to \supset$, $D \to]$, $S \to BSCSD$

So,

$S \to BSCSD$ is replaced by $S \to BC_1$, $C_1 \to SC_2$, $C_2 \to CC_3$, $C_3 \to SD$.

Let

$$G_2 = (\{S, A, B, C, D, C_1, C_2, C_3\}, T, P_2, S)$$

where P_2 consists of $S \to p|q|AS|BC_1$, $A \to \sim$, $B \to [$, $C \to \supset$, $D \to]$, $C_1 \to SC_2$, $C_2 \to CC_3$, $C_3 \to SD$. G_2 is in CNF and equivalent to the given grammar.

Prob.44. What do you mean by normal forms ? Reduce the grammar G with following productions to CNF.

$$S \to ASA|bA$$
$$A \to B|S$$
$$B \to c$$

Sol. **Normal Forms** – Refer to Q.18. ***(R.G.P.V., Dec. 2015)***

Write all the productions separately

$$S \to ASA$$
$$S \to bA$$
$$A \to B$$
$$A \to S$$
$$B \to c$$

The only production $B \to c$ in Chomsky normal form

There is no null production.

Eliminating unit production.

$$S \to ASA \mid bA$$
$$A \to c \mid ASA \mid bA$$
$$B \to c$$

Eliminating useless production

$$S \to ASA/bA$$
$$A \to c/ASA/bA$$

Restricting the right sides of productions to single terminals or strings of two or more variables. Thus, replace terminal by only right by new variable $B_b \to b$

$$S \to ASA \mid B_bA$$
$$A \to c \mid ASA \mid B_bA$$
$$B_b \to b$$

Replace a combination of two variables on R.H.S. by a single variable unit to get the R.H.S. of the productions in two variables.

$$S \to AD_1/B_b\,A$$
$$A \to c/AD_1/B_b\,A$$
$$B_b \to b$$
$$D_1 \to SA$$

The grammar is in CNF.

Prob.45. Transform the grammar with productions –

$$S \to abAB,\ A \to bAB \mid \lambda,\ B \to BAa \mid A \mid \lambda$$

into Chomsky normal form.

(R.G.P.V., June 2005)

Or

Transform the grammar with productions –

$$S \to abAB,\ A \to bAB|\in,\ B \to BAa|A|\in$$

into Chomsky normal form.

(R.G.P.V., June 2008)

Sol. **Step 1** – Write all the production separately

$$S \to abAB$$
$$A \to bAB$$
$$A \to \in$$
$$B \to BAa$$
$$B \to A$$
$$B \to \in$$

Step 2 – Eliminating $\in$-productions

$$S \to abAB|abB|abA|ab$$
$$A \to bAB|bB|bA|b$$
$$B \to BAa|Ba|Aa|a$$

In this grammar, there is no unit production, thus the elimination of unit production is not performed.

Step 3 – Restricting the right sides of productions to single terminals or strings of two or more variables. Thus, replace terminal only right by new variables $C_a \to a$ and $C_b \to b$.

$$S \to C_aC_bAB|C_aC_bB|C_aC_bA|C_aC_b$$
$$A \to C_bAB|C_bB|C_bA|b$$
$$B \to BAC_a|BC_a|AC_a|a$$
$$C_a \to a$$
$$C_b \to b$$

Step 4 – Replace a combination of two variables on R.H.S. by a single variable, until the R.H.S. of the productions have only two variables.

$$S \to C_aE_1/C_aE_3/C_aE_4/C_aC_b$$
$$A \to C_bE_2 \mid C_bB \mid C_bA|b$$
$$B \to BE_5/BC_a|AC_a|a$$
$$C_a \to a, \quad E_1 \to C_bE_2$$
$$C_b \to b, \quad E_2 \to AB$$
$$E_3 \to C_bB \quad E_4 \to C_bA$$
$$E_5 \to AC_a$$

Prob.46. Convert the following CFG to CNF

$$S \to ASB \mid \in$$
$$A \to aAS \mid a$$
$$B \to SbS \mid A \mid bb$$

(R.G.P.V., June 2016)

Sol. **Step 1** – Write all the productions separately

$$S \to ASB$$
$$S \to \in$$
$$A \to aAS$$

$$A \to a$$
$$B \to SbS$$
$$B \to A$$
$$B \to bb$$

The only production $A \to a$ in Chomsky normal form.

Step 2 – Eliminating $\in$-productions –

$$S \to ASB \mid AB$$
$$A \to aAS \mid aA \mid a$$
$$B \to SbS \mid bS \mid Sb \mid b \mid A \mid bb$$

Step 3 – Eliminating unit productions –

$$S \to ASB \mid AB$$
$$A \to aAS \mid aA \mid a$$
$$B \to SbS \mid bS \mid Sb \mid b \mid aAS \mid aA \mid a \mid bb$$

Step 4 – Restricting the right sides of productions to single terminals or strings of two or more variables. Thus, replace terminal only right by new variable

$$B_a \to a \text{ and } B_b \to b$$
$$S \to ASB \mid AB$$
$$A \to B_aAS \mid B_aA \mid a$$
$$B \to SB_bS \mid B_bS \mid SB_b \mid b \mid B_aAS \mid B_aA \mid a \mid B_bB_b$$

Step 5 – Replace a combination of two variables on R.H.S. by a single variable unit to get the R.H.S. of the productions in two variables

$$S \to AA_1/\text{AB}$$
$$A \to B_aA_2/B_aA/a$$
$$A_1 \to SB$$
$$A_2 \to AS$$
$$B \to SA_3/B_bS/SB_b/b/B_aA_2/B_aA/a/B_aB_b$$
$$A_3 \to B_bS$$

Prob.47. Convert the following CFG to Chomsky normal form –

$$S \to AACD$$
$$A \to aAb \mid \in$$
$$C \to aC \mid a$$
$$D \to aDa \mid bDb \mid \in$$

(R.G.P.V., Dec. 2005)

Sol. **Step 1** – Write all the productions separately

$$S \to AACD$$
$$A \to aAb$$
$$A \to \in$$
$$C \to aC$$
$$C \to a$$
$$D \to aDa$$

$D \to bDb$

$D \to \in$

The only production $C \to a$ in Chomsky normal form.

Step 2 – Eliminating $\in$-productions –

$S \to AACD|ACD|AAC|CD|AC|C$

$A \to aAb|ab$

$C \to aC|a$

$D \to aDa|bDb|aa|bb$

Step 3 – Eliminating unit productions –

Thus, the grammar

$S \to AACD|ACD|AAC|CD|AC|aC|a$

$A \to aAb|ab$

$C \to aC|a$

$D \to aDa|bDb|aa|bb$

Step 4 – Restricting the right sides of productions to single terminals or strings of two or more variables. Thus, replace terminal only right by new variables $B_a \to a$ and $B_b \to b$

$S \to AACD|ACD|AAC|CD|AC|B_aC|a$

$A \to B_aAB_b|B_aB_b$

$C \to B_aC|a$

$D \to B_aDB_a|B_bDB_b|B_aB_a|B_bB_b$

$B_a \to a$

$B_b \to b$

Step 5 – Replace a combination of two variables on R.H.S. by a single variable unit to get the R.H.S. of the productions in two variables

$S \to AE_1/AE_2/AE_3/CD/AC/B_aC/a$

$A \to B_aE_4/B_aB_b$

$B_a \to a$

$B_b \to b$

$C \to B_aC/a$

$D \to B_aE_5/B_bE_6/B_aB_a/B_bB_b$

$E_1 \to AE_2,\ E_2 \to CD,\ E_3 \to AC,$

$E_4 \to AB_b,\ E_5 \to DB_a,\ E_6 \to DB_b$

Prob.48. Convert the following grammar G into Chomsky normal form.

$S \to ABAC$

$A \to aA|\in$

$B \to bB|\in$

$C \to c$

(R.G.P.V., Dec. 2013)

Sol. **Step 1 – Elimination of Null Productions and Unit Productions –**

The following grammar is obtained after removing null production.

$S \to ABAC|BAC|ABC|BC|AAC|AC|C$

$A \to aA|a$

$B \to bB|b$

$C \to c$

Now, unit productions are removed.

$S \to ABAC|BAC|ABC|BC|AAC|AC|c$

$A \to aA|a$

$B \to bB|b$

$C \to c$

Step 2 – Elimination of Terminals on R.H.S. –

$S \to ABAC|BAC|ABC|BC|AAC|AC|c$

$A \to C_aA|a$

$C_a \to a$

$B \to B_bB|b$

$B_b \to b$

$C \to c$

Step 3 – Restricting the Number of Variables on R.H.S. –

$S \to AE_1|BE_2|AE_3|BC|AE_2|AC|c$

$A \to C_aA|a, \quad E_1 \to BE_2$

$B \to B_bB|b, \quad E_2 \to AC$

$C \to c, \quad E_3 \to BC$

$C_a \to a, \quad B_b \to b$

The grammar is in CNF.

Prob.49. Convert the following grammar to CNF –

$\boldsymbol{S \to bA \mid aB}$

$\boldsymbol{A \to bAA \mid aS \mid a}$

$\boldsymbol{B \to aBB \mid bSbb}$

(R.G.P.V., Dec. 2011, 2017)

Sol. **Step 1** – Write all the productions separately

$S \to bA$

$S \to aB$

$A \to bAA$

$A \to aS$

$A \to a$

$B \to aBB$

$B \to bSbb$

The only production already in CNF are $A \to a$. There are no null and unit productions.

Step 2 – Choose the productions which are not in CNF, they are
$S \to bA$, $S \to aB$, $A \to bAA$, $A \to aS$, $B \to aBB$, $B \to bSbb$

Step 3 – Replace terminal only right by new variables $C_a \to a$ and $C_b \to b$

(i) $S \to bA$ is replaced by $S \to C_bA$ such that $C_b \to b$
(ii) $S \to aB$ is replaced by $S \to C_aB$ such that $C_a \to a$
(iii) $A \to bAA$ is replaced by $A \to C_bAA$ such that $C_b \to b$
(iv) $A \to aS$ is replaced by $A \to C_aS$ such that $C_a \to a$
(v) $B \to aBB$ is replaced by $B \to C_aBB$ such that $C_a \to a$
(vi) $B \to bSbb$ is replaced by $B \to C_bSC_bC_b$ such that $C_b \to b$

Step 4 – Replaced a combination of two variables on R.H.S. by a single variable.

(i) $A \to C_bAA$ is replaced by $A \to C_bD_1$ such that $D_1 \to AA$
(ii) $B \to C_aBB$ is replaced by $B \to C_aD_2$ such that $D_2 \to BB$
(iii) $B \to C_bSC_bC_b$ is replaced by $B \to C_bSD_3$ such that $D_3 \to C_bC_b$

Step 5 – Again replaced a combination of two variables on R.H.S. by a single variable.

$B \to C_bSD_3$ is replaced by $B \to C_bE_1$ such that $E_1 \to SD_3$

Therefore the grammar G' is

$G' = (\{S, A, B, C_a, C_b, D_1, D_2, D_3, E_1\}, \{a, b\}, P_1, S)$

The production P_1 which are in CNF are

$S \to C_bA \mid C_aB$
$A \to C_bD_1 \mid C_aS \mid a$
$B \to C_aD_2 \mid C_bE_1$
$C_a \to a$
$C_b \to b$
$D_1 \to AA$
$D_2 \to BB$
$D_3 \to C_bC_b$
$E_1 \to SD_3$

Prob.50. Find a grammar in Chomsky normal form equivalent to $S \to aAD$; $A \to aB \mid bAB$; $B \to b$, $D \to d$. **(R.G.P.V., May 2018)**

Sol. **Step 1** – Write all the productions separately

$S \to aAD$
$A \to aB$
$A \to bAB$

The only production already in CNF are $B \to$ b and $D \to d$.

Step 2 – Choose the productions which are not in CNF, there are
$S \to aAD$, $A \to aB$, $A \to bAB$

Step 3 – Replace terminal only right by new variables $C_a \to a$ and $C_b \to b$

(i) $S \to aAD$ is replaced by $S \to C_aAD$ such that $C_a \to a$
(ii) $A \to aB$ is replaced by $A \to C_aB$ such that $C_a \to a$
(iii) $A \to bAB$ is replaced by $A \to C_bAB$ such that $C_b \to b$.

Step 4 – Replaced a combination of two variable on R.H.S. by a single variable

(i) $S \to C_aAD$ is replaced by $S \to C_aE_1$ such that $E_1 \to AD$
(ii) $A \to C_bAB$ is replaced by $A \to C_bE_2$ such that $E_2 \to AB$.

$\therefore$ The grammar is –

$G' = (\{S, A, C_a, C_b, E_1, E_2\}, \{a, b\}, P, S)$

The production P_1 which are in CNF are

$S \to C_aE_1$
$A \to C_aB \mid C_bE_2$
$B \to b$
$D \to d$
$E_1 \to AD$
$E_2 \to AB$.

Prob.51. Reduce the grammar into CNF and GNF –
$G = (\{S, A, B\}, \{0, 1\}, \{S \to 1A \mid 0B, A \to 0S \mid 0, B \to 1S \mid 1\}, S)$. **(R.G.P.V., Dec. 2011)**

Sol. Given grammar is

$S \to 1A|0B$
$A \to 0S|0$
$B \to 1S|1$

(i) Conversion into CNF

Let $A_1 \to 0$ and $A_2 \to 1$

Then the above grammar in Chomsky normal form is

$S \to A_2A|A_1B$, $A_1 \to 0$
$A \to A_1S/0$, $A_2 \to 1$
$B \to A_2S/1$

(ii) Conversion into GNF

The given grammar is already in *GNF* i.e.

$S \to 1A/0B$
$A \to 0S/0$
$B \to 1S/1$

Prob.52. Construct a grammar in GNF equivalent to grammar –

$$S \to AA|a$$
$$A \to SS|b$$

(R.G.P.V., June 2006, May 2018)

Or

Find a grammar in GNF equivalent to the following grammar G

$$S \to AA|a$$
$$A \to SS|b$$

(R.G.P.V., June 2017)

***Sol.* Step 1** – The given grammar is in CNF. S and A are renamed as A_1 and A_2, respectively. So the productions are $A_1 \to A_2A_2 \mid a$ and $A_2 \to A_1A_1|b$. As the given grammar has no null productions and is in CNF we need not carry out step 1. So we proceed to step 2.

Step 2 – (i) A_1-productions are in the required form. They are $A_1 \to A_2A_2 \mid a$.

(ii) $A_2 \to b$ is in the required form. We apply Lemma 1 of GNF and we get, they are $A_2 \to A_1 A_1$. The resulting productions are $A_2 \to A_2 A_2 A_1$, $A_2 \to aA_1$. Thus the A_2-productions $A_2 \to A_2 A_2 A_1$, $A_2 \to aA_1$, $A_2 \to b$.

Step 3 – A_2-productions as we have $A_2 \to A_2 A_2 A_1$. Let Z_2 be the new variable. The resulting productions are

$$A_2 \to aA_1, \quad A_2 \to b$$
$$A_2 \to aA_1Z_2, \quad A_2 \to bZ_2$$
$$Z_2 \to A_2 A_1, \quad Z_2 \to A_2 A_1Z_2.$$

Step 4 – (i) A_2-productions are $A_2 \to aA_1 \mid b \mid aA_1Z_2 \mid bZ_2$.

(ii) Among the A_1-productions we retain $A_1 \to a$ and eliminate $A_1 \to A_2 A_2$ using Lemma 1 of GNF. The resulting productions are $A_1 \to aA_1 A_2 \mid bA_2$, $A_1 \to aA_1Z_2 A_2|bZ_2 A_2$. The set of all (modified) A_1-productions are $A_1 \to a|aA_1 A_2|bA_2|aA_1Z_2 A_2|bZ_2 A_2$.

Step 5 – The Z_2-productions to be modified are $Z_2 \to A_2 A_1$, $Z_2 \to A_2 A_1Z_2$. We apply Lemma 1 of GNF get

$$Z_2 \to aA_1 A_1 \mid bA_1 \mid aA_1Z_2 A_1 \mid bZ_2 A_1$$
$$Z_2 \to aA_1 A_1Z_2 \mid bA_1Z_2 \mid aA_1Z_2 A_1Z_2 \mid bZ_2 A_1Z_2$$

Hence, the equivalent grammar is

$$G' = (\{A_1, A_2, Z_2\}, \{a, b\}, P_1, A_1)$$

where P_1 consists of

$$A_1 \to a \mid aA_1 A_2 \mid bA_2 \mid aA_1Z_2 A_2 \mid bZ_2 A_2$$
$$A_2 \to aA_1 \mid b \mid aA_1Z_2 \mid bZ_2$$
$$Z_2 \to aA_1 A_1 \mid bA_1 \mid aA_1Z_2 A_1 \mid bZ_2 A_1$$
$$Z_2 \to aA_1A_1Z_2 \mid bA_1Z_2 \mid aA_1Z_2A_1Z_2 \mid bZ_2 A_1Z_2$$

Prob.53. Find a Greibach normal form grammar equivalent to the following CFG –

$$S \to AA|0$$
$$A \to SS|1$$

Or

Convert the following grammar into Greibach normal form.

$$S \to AA \mid 0$$
$$A \to SS \mid 1$$

(R.G.P.V., Dec. 2012)

Or

Find the Greibach normal form grammar equivalent to the following CFG –

$$S \to XX \mid 0$$
$$X \to SS \mid 1$$

(R.G.P.V., Dec. 2006)

Or

Convert the following CFG into Greibach normal form –

$$S \to AA|b$$
$$A \to SS|a$$

(R.G.P.V., June 2008)

Sol. Refer to Prob.52.

Prob.54. Convert the following grammar G into GNF

$$S \to XA \mid BB$$
$$B \to b \mid SB$$
$$X \to b$$
$$A \to a$$

(R.G.P.V., June 2016)

Or

Reduce the following grammar G into GNF

$$S \to XA \mid BB$$
$$B \to SB \mid b$$
$$X \to b$$
$$A \to a$$

(R.G.P.V., Nov. 2018)

***Sol.* Step 1** – The given grammar is in CNF. S, B, X and A are renamed as A_1, A_2, A_3 and A_4, respectively. So the productions are $A_1 \to A_3A_4 \mid A_2A_2$, $A_2 \to b \mid A_1A_2$, $A_3 \to b$, $A_4 \to a$. Now, we proceed to step 2.

Step 2 – A_1 productions are in the required form. They are $A_1 \to A_3A_4 \mid A_2A_2$. The $A_3 \to b$ production is in the required form. The $A_4 \to a$ production is also in required form. The production $A_2 \to b \mid A_1A_2$ are not in required form. Thus the required productions are as follows –

$$A_2 \to b, \; A_2 \to A_3A_4A_2, \; A_2 \to A_2A_2A_2$$

Again, substitute $A_3 \to b$, we get

$$A_2 \to bA_4A_2, \; A_2 \to A_2A_2A_2$$

Step 3 – The A_2 production as we have $A_2 \to A_1A_2A_2A_2$ Let Z_4 be the new variable. The resulting productions are –

$$A_2 \to b,\ A_2 \to bA_4A_2$$
$$A_2 \to bZ,\ A_2 \to bA_4A_2Z$$
$$Z \to A_2A_2,\ Z \to A_2A_2Z$$

Step 4 – A_2 productions are –

$$A_2 \to b \mid bA_4A_2 \mid bZ \mid bA_4A_2Z$$

Among the productions A_1, we eliminate $A_1 \to A_3A_4 \mid A_2A_2$. The resulting productions are $A_1 \to bA_4$, $A_1 \to bA_2$, $A_1 \to bA_4A_2A_2$, $A_1 \to bZA_2$, $A_1 \to bA_4A_2ZA_2$. The set of all (modified) A_1 productions are $A_1 \to bA_4 \mid bA_2 \mid bA_4A_2A_2 \mid bZA_2 \mid bA_4A_2ZA_2$.

Step 5 – The Z productions to be modified are –

$$Z \to A_2A_2 \text{ and } Z \to A_2A_2Z, \text{ we get}$$
$$Z \to bA_2 \mid bA_4A_2A_2 \mid bZA_2 \mid bA_4A_2ZA_2$$
$$Z \to bA_2Z \mid bA_4A_2A_2Z \mid bZA_2Z \mid bA_4A_2ZA_2Z$$

Hence, the equivalent grammar is

$$G' = (\{A_1, A_2, A_3, A_4\}, (a, b), P_1, A_1)$$

where P_1 consists of

$$A_1 \to bA_4 \mid bA_2 \mid bA_4A_2A_2 \mid bZA_2 \mid bA_4A_2ZA_2$$
$$A_2 \to b \mid bA_4A_2 \mid bZ \mid bA_4A_2Z$$
$$A_3 \to b$$
$$A_4 \to a$$
$$Z \to bA_2 \mid bA_4A_2A_2 \mid bZA_2 \mid bA_4A_2ZA_2 \mid bA_2Z \mid bA_4A_2A_2Z \mid bZA_2Z \mid bA_4A_2ZA_2Z$$

Prob.55. Convert the following CFG to Greibach Normal Form (GNF)–

$$S \to AB \mid 0$$
$$A \to BC \mid 1$$
$$B \to CD \mid 2$$
$$C \to AD \mid 0$$
$$D \to 1$$

(R.G.P.V., June 2007, Dec. 2009)

***Sol.* Step-1** – The given grammar is in CNF. S, A, B, C and D are renamed as A_1, A_2, A_3, A_4 and A_5, respectively. So, the productions are $A_1 \to A_2A_3|0$, $A_2 \to A_3A_4 \mid 1$, $A_3 \to A_4A_5|2$, $A_4 \to A_2A_5|0$ and $A_5 \to 1$. Now we proceed to next step.

Step-2 –

(i) A_1-productions are in the required form. They are $A_1 \to A_2A_3|0$.

(ii) The $A_2 \to A_3A_4 \mid 1$ are in the required form.

(iii) $A_3 \to A_4A_5 \mid 2$ are in the required form.

(iv) $A_5 \to 1$ is also in required form.

The productions $A_4 \to A_2\ A_5 \mid 0$ are not in required form. Thus the required productions are as follows –

$$A_4 \to A_3\ A_4\ A_5,\ A_4 \to 1A_5,\quad A_4 \to 0$$

Again, eliminating $A_4 \to A_3\ A_4\ A_5$, we get,

$$A_4 \to A_4A_5A_4A_5\ ,\quad A_4 \to 2\ A_4\ A_5$$

Step-3 – The A_4-production as we have $A_4 \to A_4\ A_5\ A_4\ A_5$. Let Z_4 be the new variable. The resulting productions are –

$$A_4 \to 1\ A_5 \qquad A_4 \to 0$$
$$A_4 \to 2A_4A_5 \qquad A_4 \to 0Z_4$$
$$A_4 \to 1A_5\ Z_4 \qquad A_4 \to 2A_4A_5Z_4$$
$$Z_4 \to A_5A_4A_5 \qquad Z_4 \to A_5A_4A_5Z_4$$

Step-4 – (i) A_4-productions are as –

$$A_4 \to 0 \mid 1A_5 \mid 2\ A_4A_5 \mid 0Z_4 \mid 1\ A_5Z_4|2A_4A_5Z_4 \qquad ....(i)$$

(ii) Among A_3-productions, we retain $A_3 \to 2$ and eliminate $A_3 \to A_4A_5$, we get

$$A_3 \to 0A_5 \mid 1A_5\ A_5 \mid 2A_4\ A_5\ A_5 \mid 0Z_4\ A_5 \mid 1\ A_5\ Z_4\ A_5|2A_4A_5Z_4A_5$$

The modified A_3-productions are

$$A_3 \to 2 \mid 0A_5 \mid 1\ A_5\ A_5 \mid 2\ A_4\ A_5\ A_5 \mid 0\ Z_4\ A_5 \mid 1\ A_5\ Z_4\ A_5 \mid 2A_4A_5Z_4A_5 \qquad ....(ii)$$

(iii) Similarly, among A_2-productions,

$$A_2 \to 1 \mid 2A_4 \mid 0A_5\ A_4 \mid 1A_5\ A_5\ A_4 \mid 2A_4A_5A_5A_4 \mid 0Z_4A_5A_4 \mid 1A_5Z_4A_5A_4|2A_4A_5Z_4A_5A_4 \qquad ....(iii)$$

(iv) Among A_1-productions, we have

$$A_1 \to 0 \mid 1A_3 \mid 2A_4A_3 \mid 0A_5A_4A_3 \mid 1\ A_5A_5A_4A_3 \mid 2\ A_4A_5A_5A_4A_3| 0Z_4A_5A_4A_3 \mid 1A_5Z_4A_5A_4A_3|2A_4A_5Z_4A_5A_4A_3 \qquad ....(iv)$$

(v) Among A_5-productions,

$$A_5 \to 1 \qquad ....(v)$$

Step-5 – The Z_4-production to be modified are, $Z_4 \to A_5A_4A_5 \mid A_5A_4A_5Z_4$, we get,

$$Z_4 \to 1A_4A_5 \mid 1A_4A_5Z_4 \qquad ....(vi)$$

The required GNF is given by equations (i) to (vi).

Prob.56. Find a Greibach normal form equivalent to the following CFG–

$$S \to AB|a,\ A \to BS|b,\ B \to SA|c$$

(R.G.P.V., Dec. 2010)

Sol. Since the given grammar is in CNF we can omit step 1 and proceed to step 2 after renaming S, A, B as A_1, A_2, A_3, respectively. The productions are $A_1 \to A_2A_3|a$, $A_2 \to A_3A_1|b$, $A_3 \to A_1A_2|a$.

Step 2 – (i) The A_1-productions $A_1 \to A_2A_3|a$ are in the required form.

(ii) The A_2-productions $A_2 \to A_3A_1|b$ are in the required form.

(iii) $A_3 \to c$ is in the required form.

Applying lemma to $A_3 \to A_1A_2$. The resulting productions are $A_3 \to A_2A_3A_2|aA_2$. Applying the lemma once again to $A_3 \to A_2A_3A_2$, we get $A_3 \to A_3A_1A_3A_2|bA_3A_2$.

Step 3 – The A_3-productions are $A_3 \to c|bA_3A_2|aA_2$ and $A_3 \to A_3A_1A_3A_2$. As we have $A_3 \to A_3A_1A_3A_2$, we have to apply lemma to A_3-productions. Let Z_3 be the new variable. The resulting productions are

$$A_3 \to c|bA_3A_2|aA_2,\ A_3 \to cZ_3|bA_3A_2Z_3|aA_2Z_3$$

$$Z_3 \to A_1A_3A_2,\ Z_3 \to A_1A_3A_2Z_3$$

Step 4 – (i) The A_3-productions are

$$A_3 \to c|bA_3A_2|cZ_3|bA_3A_2Z_3|aA_2|aA_2Z_3 \quad ...(i)$$

(ii) Among A_2-productions, we retain $A_2 \to b$ and eliminate $A_2 \to A_3A_1$ using lemma. The resulting productions are

$$A_2 \to cA_1|bA_3A_2A_1|cZ_3A_1|bA_3A_2Z_3A_1|aA_2A_1|aA_2Z_3A_1$$

The modified A_2-productions are

$$A_2 \to b|cA_1|bA_3A_2A_1|cZ_3A_1|bA_3A_2Z_3A_1|aA_2A_1|aA_2Z_3A_1 \quad ...(ii)$$

(iii) We apply lemma to $A_1 \to A_2A_3$ to get

$$A_1 \to bA_3|cA_1A_3|bA_3A_2A_1A_3|cZ_3A_1A_3|\ bA_3A_2Z_3A_1A_3 \\ |aA_2A_1A_3|aA_2Z_3A_1A_3 \quad ...(iii)$$

Step 5 – The Z_3-productions to be modified are

$$Z_3 \to bA_3A_3A_2|bA_3A_3A_2Z_3$$

$$Z_3 \to cA_1A_3A_3A_2|cA_1A_3A_3A_2Z_3$$

$$Z_3 \to bA_3A_2A_1A_3A_3A_2|bA_3A_2A_1A_3A_3A_2Z_3 \quad ...(iv)$$

$$Z_3 \to cZ_3A_1A_3A_3A_2|cZ_3A_1A_3A_3A_2Z_3$$

$$Z_3 \to bA_3A_2Z_3A_1A_3A_3A_2|bA_3A_2Z_3A_1A_3A_3A_2Z_3$$

$$Z_3 \to aA_2A_1A_3A_3A_2|aA_2A_1A_3A_3A_2Z_3$$

$$Z_3 \to aA_2Z_3A_1A_3A_3A_2|aA_2Z_3A_1A_3A_3A_2Z_3 \quad ...(iv)$$

The required grammar in GNF is given by equations (i) to (iv).

Prob.57. Convert the following grammar to GNF.

$$A_1 \to A_2A_3,\ A_2 \to A_3A_1 \mid b,\ A_3 \to A_1A_2 \mid a$$

(R.G.P.V., Dec. 2013, Nov. 2019)

Or

Convert the following CFG to GNF –

$$A \to BC$$
$$B \to CA \mid b$$
$$C \to AB \mid a$$

(R.G.P.V., Dec. 2014)

Or

Convert the grammar $S \to AB, A \to BS \mid b, B \to SA \mid a$ into Greibach normal form.

(R.G.P.V., June 2015)

Sol. The grammar is

$$G = (\{A_1, A_2, A_3\}, \{a, b\}, P, A_1)$$

where P consists of the following –

$$A_1 \to A_2A_3$$
$$A_2 \to A_3A_1|b$$
$$A_3 \to A_1A_2|a$$

The steps to convert the grammar into GNF are as follows –

Step-1 – Since the right-hand side of the productions for A_1 and A_2 begin with terminals or higher-numbered variables, we start with the production $A_3 \to A_1A_2$ and substitute the string A_2A_3 for A_1.

The resulting set of productions is

$$A_1 \to A_2A_3$$
$$A_2 \to A_3A_1|b$$
$$A_3 \to A_2A_3A_2|a$$

Since the right side of the production $A_3 \to A_2A_3A_2$ begins with a lower-numbered variable, we substitute for the first occurrence of A_2 both A_3A_1 and b. Thus, the new set is

$$A_1 \to A_2A_3$$
$$A_2 \to A_3A_1|b$$
$$A_3 \to A_3A_1A_3A_2|bA_3A_2|a$$

Now, apply lemma to the productions

$$A_3 \to A_3A_1A_3A_2|bA_3A_2|a$$

Symbol B_3 is introduced, and the production $A_3 \to A_3A_1A_3A_2$ is replaced by $A_3 \to bA_3A_2B_3$, $A_3 \to aB_3$, $B_3 \to A_1A_3A_2$, and $B_3 \to A_1A_3A_2B_3$. The resulting set is

$$A_1 \to A_2A_3$$
$$A_2 \to A_3A_1|b$$
$$A_3 \to bA_3A_2B_3|aB_3|bA_3A_2|a$$
$$B_3 \to A_1A_3A_2|A_1A_3A_2B_3$$

Step-2 – Now all the productions with A_3 on the left have right-hand sides that start with terminals. These are used to replace A_3 in the production $A_2 \to A_3A_1$ and then the productions with A_2 on the left are used to replace A_2 in the production $A_1 \to A_2A_3$. The result is as follows –

$$A_3 \to bA_3A_2B_3 \qquad A_3 \to bA_3A_2$$
$$A_3 \to aB_3 \qquad A_3 \to a$$
$$A_2 \to bA_3A_2B_3A_1 \qquad A_2 \to bA_3A_2A_1$$
$$A_2 \to aB_3A_1 \qquad A_2 \to aA_1$$
$$A_2 \to b$$
$$A_1 \to bA_3A_2B_3A_1A_3 \qquad A_1 \to bA_3A_2A_1A_3$$
$$A_1 \to aB_3A_1A_3 \qquad A_1 \to aA_1A_3$$
$$A_1 \to bA_3$$
$$B_3 \to A_1A_3A_2 \qquad B_3 \to A_1A_3A_2B_3$$

Step-3 – The two B_3 productions are converted to proper form resulting in 10 more productions. That is, the productions

$B_3 \to A_1A_3A_2$ and $B_3 \to A_1A_3A_2B_3$

are changed by substituting the right side of each of the five productions with A_1 on the left for the first occurrences of A_1. Thus, $B_3 \to A_1A_3A_2$ becomes

$B_3 \to bA_3A_2B_3A_1A_3A_3A_2$, $B_3 \to aB_3A_1A_3A_3A_2$

$B_3 \to bA_3A_3A_2$, $B_3 \to bA_3A_2A_1A_3A_3A_2$, $B_3 \to aA_1A_3A_3A_2$

The other production for B_3 is replaced similarly. The final set of productions is

$$A_3 \to bA_3A_2B_3 \qquad A_3 \to bA_3A_2$$
$$A_3 \to aB_3 \qquad A_3 \to a$$
$$A_2 \to bA_3A_2B_3A_1 \qquad A_2 \to bA_3A_2A_1$$
$$A_2 \to aB_3A_1 \qquad A_2 \to aA_1$$
$$A_2 \to b$$
$$A_1 \to bA_3A_2B_3A_1A_3 \qquad A_1 \to bA_3A_2A_1A_3$$
$$A_1 \to aB_3A_1A_3 \qquad A_1 \to aA_1A_3$$
$$A_1 \to bA_3$$
$$B_3 \to bA_3A_2B_3A_1A_3A_3A_2B_3 \qquad B_3 \to bA_3A_2B_3A_1A_3A_3A_2$$
$$B_3 \to aB_3A_1A_3A_3A_2B_3 \qquad B_3 \to aB_3A_1A_3A_3A_2$$
$$B_3 \to bA_3A_3A_2B_3 \qquad B_3 \to bA_3A_3A_2$$
$$B_3 \to bA_3A_2A_1A_3A_3A_2B_3 \qquad B_3 \to bA_3A_2A_1A_3A_3A_2$$
$$B_3 \to aA_1A_3A_3A_2B_3 \qquad B_3 \to aA_1A_3A_3A_2$$

UNIT 4

PUSH DOWN AUTOMATA

EXAMPLE OF PDA, DETERMINISTIC AND NON-DETERMINISTIC PDA, CONVERSION OF PDA INTO CONTEXT-FREE GRAMMAR AND VICE VERSA, CFG EQUIVALENT TO PDA, PETRINET MODEL

Q.1. Explain PDA with an example.

Or

Give the definition of pushdown automata with the help of diagram.

(R.G.P.V., Dec. 2014)

Or

Explain PDA. ***(R.G.P.V., June 2016)***

Or

Give the formal definition of PDA. ***(R.G.P.V., Dec. 2016)***

Ans. A ***pushdown automaton*** is a 7-tuple machine, viz. $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ consists of –

(i) A finite nonempty set of states denoted by Q

(ii) A finite nonempty set of input symbols denoted by Σ

(iii) A finite nonempty set of pushdown symbols denoted by Γ

(iv) A special state called the initial state denoted by q_0

(v) A special pushdown symbol called the initial symbol on the pushdown stack denoted by Z_0

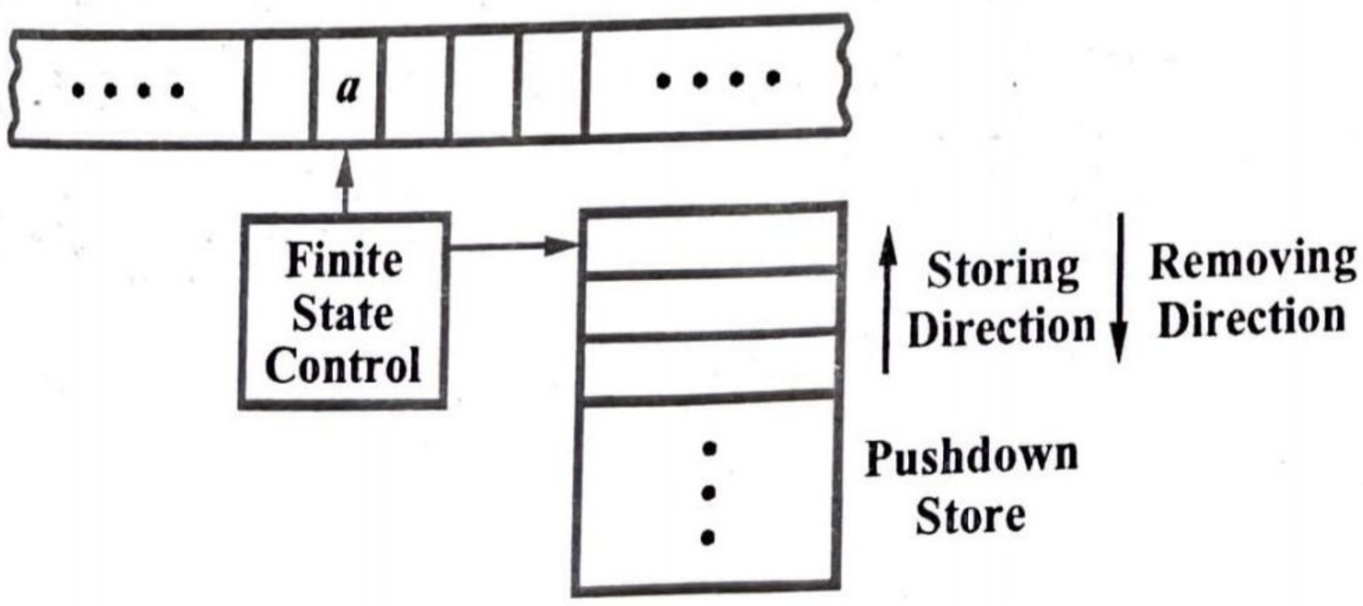


Fig. 4.1 Pushdown Automata

(vi) A set of final states, which is a subset of Q denoted by F

(vii) The transition function δ from $Q \times (\Sigma \cup \{\in\}) \times \Gamma$ to the set of finite subsets of $Q \times \Gamma^*$.

Example – The formal description of the PDA that recognizes the language $\{0^n 1^n \mid n \geq 0\}$

Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$

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

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

$\Gamma = \{0, \$\}$

$F = \{q_1, q_4\}$

and δ is given by the table 4.1, where blank entries are ϕ.

Table 4.1

Input :	*0*			*1*			$\in$		
Stack :	*0*	*$*	$\in$	*0*	*$*	$\in$	*0*	*$*	$\in$
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \in)\}$					
q_3				$\{(q_3, \in)\}$				$\{(q_4, \in)\}$	
q_4									

State diagram can also be used to describe a PDA, as shown in fig. 4.2. Such diagrams are similar to the state diagrams used to describe finite automata, to show when going from state to state how the PDA uses its stack. We write "$a, b \to c$" to signify that when the machine is reading an a from the input it may replace the symbol b on the top of the stack with a, c. Any of a, b and c may be $\in$.

If a is $\in$, the machine may make this transition without reading any symbol from the input. If b is $\in$ the machine may make this transition without reading and popping any symbol from the stack. If c is $\in$, the machine does not write any symbol on the stack when going along this transition.

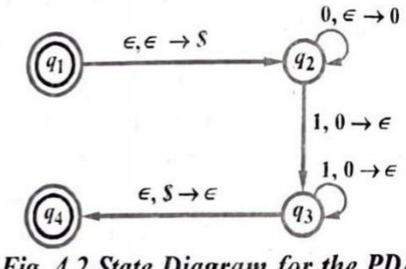


Fig. 4.2 State Diagram for the PDA M_1 that Recognizes $\{0^n 1^n | n \geq 0\}$

Q.2. What is PDA ? Explain instantaneous description of PDA. ***(R.G.P.V., Dec. 2015)***

Ans. **PDA** – Refer to Q.1.

The triplet (q, w, u), where q is the state of the control unit, w is the remaining unread part of the input string, and u is the stack contents (with the leftmost symbol indicating the top of the stack) is called an instantaneous description (ID) of a pushdown automaton.

A move from one instantaneous description to another will be denoted by the symbol '$\vdash$', thus

$$(q_1, aw, bx) \vdash (q_2, w, yx)$$

is possible if and only if

$$(q_2, y) \in \delta(q_1, a, b).$$

Moves containing an arbitrary number of steps will be denoted by $\vdash^*$. On occasions where several automata are under consideration, we will use $\vdash_M$ to show that the move is made by the particular automaton M.

It must be noted that an initial ID is (q_0, w, Z_0). It means initially the PDA is in the initial state q_0, the input string to be processed is w, and the stack has only one symbol, viz. Z_0. Also in an ID (q, w, u), w may be $\in$. In this case the PDA makes a $\in$-move.

Q.3. What is the additional feature PDA has when compared with NFA ? ***(R.G.P.V., June 2015)***

Ans. Pushdown automata are like non deterministic finite automata but have an extra component called a stack. The stack provides additional memory beyond the finite amount available in the control. The stack allows pushdown automata to recognize some non regular languages.

Q.4. State the differences between PDA and the FA. (R.G.P.V., Dec. 2015)

Ans. Differences between PDA and FA are as follows –

S.No.	*Pushdown Automata*	*Finite Automata*
(i)	PDA is FA with extra memory element called stack.	Finite automata does not contains any memory element.
(ii)	It consist of 7-tuples $(Q, \Sigma, \delta, \Gamma, q_0, Z_0, F)$	It consist of 5-tuples $(Q, \Sigma, \delta, q_0, F)$
(iii)	It can accept the context free language i.e. it can recognize type-2 grammar.	It can only accept regular language i.e. it can only accept type-1 grammar.
(iv)	There is a pushdown automata for every finite automata.	We can't create a finite automata for every pushdown automata.
(v)	Language can be accepted by a pushdown automata either by empty stack or by final state.	Language in finite automata can only be accept by a final state.

Q.5. Differentiate between "PDA by empty stack" and "PDA by final state". ***(R.G.P.V., Dec. 2009)***

Ans. If the language accepted by a pushdown automaton to be the set of all inputs for which some sequence of moves causes the pushdown automaton to empty its stack then this language is referred to as the language accepted by the empty stack. And, the PDA is called "PDA by empty stack".

If the language accepted by a pushdown automaton to be the set of all inputs for which some choices of moves causes the pushdown automaton to enter a final state, then this language is referred to as the language accepted by the final state. And, the PDA is called "PDA by final state".

Q.6. Define the language accepted by a pushdown automaton. Also, define the set N(A) accepted by null store.

Or

Explain how many way's PDA can accept (final out null store).

(R.G.P.V., June 2016)

Ans. Let $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a pushdown automata. Then language accepted by

(i) Final State PDA –

$$L(A) = \{w/(q_0, w, Z_0) \vdash^* (q_f, \in, \alpha) \text{ for some } q_f \in F \text{ and } \alpha \in \Gamma^*\}$$

So, the language accepted by a final state PDA is set of all strings that can put A into a final state at the end of string. The stack content α is irrelevant to this definition of acceptance.

(ii) Empty Stack PDA (or NULL Stack PDA) –

$$N(A) = \{w|(q_0, w, Z_0) \vdash^* (q, \in, \in) \text{ for some } q \in Q\}$$

So, w is in $N(A)$ if A is in initial ID(q_0, w, Z_0) and empties the stack after processing all the symbols of w. In this case set of final states is irrelevant and we usually let the set of final states be empty.

For Example – Let construct a PDA for the language

$L = \{w \in \{a, b\}^* | n_a(w) = n_b(w)\}$ by using

(a) Final State PDA

$A = (\{q_0, q_f\}, \{a, b\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_f\})$ with δ given as –

$\delta(q_0, a, Z_0) = \{(q_0, 0Z_0)\}$

$\delta(q_0, b, Z_0) = \{(q_0, 1Z_0)\}$

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

$\delta(q_0, b, 1) = \{(q_0, 11)\}$

$\delta(q_0, a, 1) = \{(q_0, \in)\}$

$\delta(q_0, b, 0) = \{(q_0, \in)\}$

$\delta(q_0, \in, Z_0) = \{(q_f, Z_0)\}$

(b) Empty Stack PDA

$A = (\{q_0\}, \{a, b\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \phi)$ with δ given as –

$\delta(q_0, a, Z_0) = \{(q_0, 0Z_0)\}$

$\delta(q_0, b, Z_0) = \{(q_0, 1Z_0)\}$

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

$\delta(q_0, b, 1) = \{(q_0, 11)\}$

$\delta(q_0, a, 1) = \{(q_0, \in)\}$

$\delta(q_0, b, 0) = \{(q_0, \in)\}$

$\delta(q_0, \in, Z_0) = \{(q_0, \in)\}$

Q.7. Prove that, if $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA accepting L by empty store, we can find a PDA

$$B = (Q', \Sigma, \Gamma', \delta_B, q'_0, Z'_0, F')$$

which accepts L by final state, i.e., L = N(A) = T(B).

Ans. B is constructed in such a way that – (i) by the initial move of B, it reaches an initial ID of A, (ii) by the final move of B, it reaches its final state, and (iii) all intermediate moves of B are as in A.

Let us define B as follows –

$$B = (Q', \Sigma, \Gamma', \delta_B, q'_0, Z'_0, F')$$

where, q'_0 is a new state (not in Q).

$F' = \{q_f\}$, with q_f as a new state (not in Q),

$Q' = Q \cup \{q'_0, q_f\}$.

Z'_0 is a new start symbol for PDS of B,

$\Gamma' = \Gamma \cup (Z'_0)$, and

δ_B is given by the rules R_1, R_2, R_3

with

$R_1 : \delta_B(q'_0, \in, Z'_0) = \{(q_0, Z_0, Z'_0)\}$,

$R_2 : \delta_B(q, a, Z) = \delta(q, a, Z)$ for all (q, a, Z) in $Q \times (\Sigma \cup \{\in\}) \times \Gamma$

$R_3 : \delta_B(q, \in, Z'_0) = \{(q_f, \in)\}$ for all $q \in Q$

By R_1, the PDA B moves from an initial ID of B to an initial ID of A. R_1 gives a $\in$-move. As a result of R_1, B moves to the initial state of A with the start symbol Z_0 on the top of PDS (pushdown stack).

R_2 is used to simulate A. Once B reaches an initial ID of A, R_2 can be used to simulate moves of A. We can repeatedly apply R_2 until Z'_0 is pushed to the top of PDS. As Z'_0 is a new pushdown symbol, we have to use R_3.

R_3 gives a $\in$-move. Using R_3, B moves to the new (final) state q_f erasing Z'_0 in PDS.

Thus the behaviour of B and A are similar except for the $\in$-moves given by R_1 and R_3. Also, $w \in T(B)$ if and only if B reaches q_f, i.e., if and only if the

PDS has no symbols from Γ (since B can reach q_f only by the application of R_3). This suggests that $T(B) = N(A)$.

Now we prove rigorously that $N(A) = T(B)$. Suppose $w \in N(A)$. Then by the definition of $N(A)$, $(q_0, w, Z_0) \overset{*}{\underset{A}{\vdash}} (q, \in, \in)$ for some $q \in Q$. Using R_2 we see that

$$(q_0, w, Z_0) \overset{*}{\underset{B}{\vdash}} (q, \in, \in)$$

By result 2,

$$(q_0, w, Z_0 Z_0') \overset{*}{\underset{B}{\vdash}} (q, \in, Z_0') \quad \text{....(i)}$$

By R_1, $(q_0', \in, Z_0') \underset{B}{\vdash} (q_0, \in, Z_0 Z_0')$

By result 1, we have

$$(q_0', w, Z_0') \underset{B}{\vdash} (q_0, w, Z_0 Z_0') \quad \text{....(ii)}$$

By R_3, $(q, \in, Z_0') \underset{B}{\vdash} (q_f, \in, \in)$ ....(iii)

Combining equation (i) – equation (iii), we have

$$(q_0', w, Z_0') \overset{*}{\underset{B}{\vdash}} (q_f, \in, \in)$$

This proves that $w \in T(B)$, i.e., $N(A) \subseteq T(B)$.

To prove $T(B) \subseteq N(A)$, start with $w \in T(B)$. Then

$$(q'_0, w, Z_0') \overset{*}{\underset{B}{\vdash}} (q_f, \in, \alpha) \quad \text{....(iv)}$$

But B can reach q_f only by application of R_3. To apply R_3, Z_0' should be the topmost element on PDS. Z_0' is placed initially, and so when it is on the top there are no other elements in PDS. So $\alpha = \in$ and equation (iv) actually reduces to

$$(q'_0, w, Z_0') \overset{*}{\underset{B}{\vdash}} (q_f, \in, \in) \quad \text{...(v)}$$

In equation (v), the initial and final steps are effected only by $\in$-moves. The intermediate steps are induced by the corresponding moves of A. So equation (v) can be split as

$$(q'_0, \in w, Z_0') \underset{B}{\vdash} (q_0, w, Z_0 Z_0') \overset{*}{\underset{B}{\vdash}} (q, \in, Z_0')$$

for some $q \in Q$. Thus, $(q'_0, \in w, Z_0') \underset{B}{\vdash} (q_0, w, Z_0 Z_0') \overset{*}{\underset{B}{\vdash}} (q, \in, Z_0') \underset{B}{\vdash} (q_f, \in, \in)$. As we get $(q_0, w, Z_0 Z_0') \overset{*}{\underset{B}{\vdash}} (q_1, \in, Z_0')$ by applying R_2 several times and R_2 does not affect Z_0' at the bottom, we have $(q_0, w, Z_0) \overset{*}{\underset{B}{\vdash}} (q, \in, \in)$. By the construction of R_2, $(q_0, w, Z_0) \overset{*}{\underset{A}{\vdash}} (q, \in, \in)$, which means $w \in N(A)$. Thus, $T(B) \subseteq N(A)$, and hence $T(B) = N(A) = L$. **Hence Proved**

Q.8. Write short note on deterministic pushdown automata. ***(R.G.P.V., Dec. 2004)***

Or

Define deterministic PDA. ***(R.G.P.V., Dec. 2016)***

Ans. A deterministic pushdown automata is defined as

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where, Q = Finite set of states

Σ = Finite set of input symbol

Γ = Finite set of pushdown stack symbol

δ = Transition function which maps

$$Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma$$

q_0 = Initial state

Z_0 = Initial stack symbol

F = Finite set of final states such that $F \subseteq Q$.

A deterministic pushdown automata never has a choice in its move. So for any given input symbol and any stack top at most one move can be made. Another condition is that when a $\in$-move is possible for some configuration, no input consuming alternative is available.

i.e. if $\delta(q, \in, b)$ is not empty then $\delta(q, c, b)$ must be empty for every $c \in \Sigma$.

Q.9. Define nondeterministic pushdown accepter (NPDA) and deterministic pushdown accepter automaton (DPDA).

Or

Explain the difference between deterministic PDA and non-deterministic PDA with example. ***(R.G.P.V., June 2009)***

Ans. A ***nondeterministic pushdown accepter*** (NPDA) is defined by the seven tuple –

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where, Q is a finite set of internal states of the control unit (i.e., finite state control)

Σ is the input alphabet

Γ is a finite set of symbols called stack alphabet

$\delta: Q \times (\Sigma \cup \{\in\} \times \Gamma) \rightarrow$ finite subsets of $Q \times \Gamma^*$ is the transition function

$q_0 \in Q$ is the initial state of control unit

$Z_0 \in \Gamma$ is the stack start symbol

$F \subseteq Q$ is the set of final states.

Deterministic pushdown acceptor automaton (DPDA) – Refer to Q.8.

Q.10. Is it true that non-deterministic PDA is more powerful than that of deterministic PDA ? Justify your answer. ***(R.G.P.V., Dec. 2014)***

Ans. Yes, it is true that non-deterministic PDA is more powerful than that of deterministic PDA. A pushdown automata that has the property that a mapping from input symbol, state and stack symbol can result in more than one next state and/or stack state. A non-deterministic PDA can be used to parse a context-sensitive grammar and is consequently more powerful than a DPDA. The languages accepted by NPDA is large than that of the DPDA.

Q.11. Deterministic and nondeterministic models of PDA are not equivalent. Justify with example. ***(R.G.P.V., June 2010)***

Or

Is it true that deterministic PDA and nondeterministic PDA are equivalent in the sense of language of acceptances ? Justify your answer. ***(R.G.P.V., June 2015)***

Ans. For finite automata, the deterministic and nondeterministic models are equivalent with respect to the languages accepted. The same is not true for PDA. In fact ww^R is accepted by a nondeterministic PDA, but not by any deterministic PDA.

For example – The following fig. 4.3, gives a PDA that accepts $\{ww^R \mid w \text{ in } (0 + 1)^*\}$. Rules (i) through (vi) allow M to store the input on the stack. In rules (iii) and (vi), M has a choice of two moves. M may decide that the middle of the input string has been reached and make the second choice – M goes to state q_2 and tries to match the remaining input symbols with the contents of the stack. If M guessed right, and if the input is of the form ww^R, then the inputs will match, M will empty its stack and thus accept the input string.

Like the nondeterministic finite automaton, a nondeterministic PDA, M accepts an input if any sequence of choices cause M to empty its stack. Thus M always "guesses right" because wrong guesses, in themselve, do not cause an input to be rejected. An input is rejected only if there is no "right guess". Fig. 4.4 shows the accessible ID's of M when processes the string 001100.

$M = (\{q_1, q_2\}, \{0, 1\}, \{R, B, G,\}, \delta, q_1, R, \phi)$

(i) $\delta(q_1, 0, R) = \{(q_1, BR)\}$

(ii) $\delta(q_1, 1, R) = \{(q_1, GR)\}$

(iii) $\delta(q_1, 0, B) = \{(q_1, BB), (q_2, \in)\}$

(iv) $\delta(q_1, 0, G) = \{(q_1, BG)\}$

(v) $\delta(q_1, 1, B) = \{(q_1, GB)\}$

(vi) $\delta(q_1, 1, G) = \{(q_1, GG), (q_2, \in)\}$

(vii) $\delta(q_2, 0, B) = \{(q_2, \in)\}$

(viii) $\delta(q_2, 1, G) = \{(q_2, \in)\}$

(ix) $\delta(q_1, \in, R) = \{(q_2, \in)\}$

(x) $\delta(q_2, \in, R) = \{(q_2, \in)\}$

Fig. 4.3 A Nondeterministic PDA that accepts $\{ww^R \mid w \text{ in } (0 + 1)^*\}$ by Empty Stack

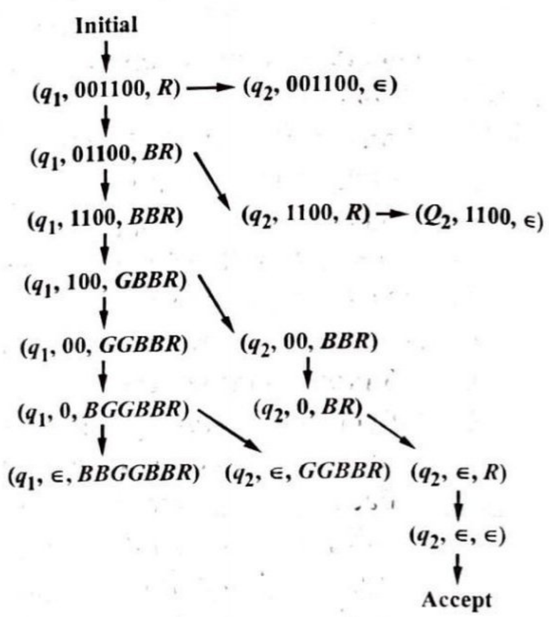


Fig. 4.4 Accessible ID's for the PDA with Input 001100

Q.12. Define deterministic pushdown automata (DPDA). It is true that DPDA and PDA are equivalent in the sense of language acceptance is concern ? Justify your answer. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.8 and Q.11.

Q.13. Write a brief note on 2-way PDA. ***(R.G.P.V., Dec. 2009)***

Ans. A two-way pushdown automaton or 2-PDA is a pushdown automaton that is permitted to move in either way on its input. Like the 2-way finite automaton (2FA), it accepts by moving off the right end of its input in a final state. The language $L = \{0^n1^n2^n \mid n \geq 1\}$ is accepted by a two-way pushdown automaton. The L is not a context-free language, by the way, so 2-PDA's are not equivalent to PDA's.

Any language accepted by a 2DPDA is recognizable in linear time on a computer. Thus, the existence of a context-free language requiring more than linear time to recognize on a computer, would imply that there are context-free languages not accepted by 2DPDAs. However, no one to date has proved that such a language exists. Incidentally, the language $\{0^n1^n2^n \mid n \geq 1\}$ is an example of a non context-free language accepted by a 2DPDA.

Q.14. If $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA, then construct a context-free grammar such that

$$L(G) = N(A).$$

Or

Show the procedure of converting a PDA into equivalent CFG using suitable example. ***(R.G.P.V., June 2010)***

Ans. **Construction of G** – We define $G = (V, T, P, S)$,

where $V = \{S\} \cup \{[q, Z, q'] \mid q, q' \in Q, Z \in \Gamma\}$

i.e., any element of V is either the new symbol S working as the start symbol for G or an ordered triple whose first and third elements are states and the second element is a pushdown symbol.

The productions in P are induced by moves of PDA as follows –

R_1 – S-productions are given by $S \to [q_0, Z_0, q]$ for every q in Q.

R_2 – Each move erasing a pushdown symbol given by $(q', \in) \in \delta(q, a, Z)$ induce the prodcution $[q, Z, q'] \to a$.

R_3 – Each move not erasing a pushdown symbol given by $(q_1, Z_1Z_2 \ldots Z_m) \in \delta(q, a, Z)$ induces many productions of the form

$$[q, Z, q'] \to a\,[q_1, Z_1, q_2]\,[q_2, Z_2, q_3] \ldots [q_m, Z_m, q']$$

where each of the states $q', q_2 \ldots q_m$ can be any state in Q. Each move yields many productions because of R_3. It is better understood by taking an example.

Construct a context-free grammar G which accepts $N(A)$, where

$A = (\{q_0, q_1\}, \{a, b\}, \{Z_0, Z\}, \delta, q_0, Z_0\ \phi\}$, where δ is given by

$$\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$$
$$\delta(q_0, \in, Z_0) = \{(q_0, \in)\}$$
$$\delta(q_0, b, Z) = \{(q_0, ZZ)\}$$
$$\delta(q_0, a, Z) = \{(q_1, Z)\}$$
$$\delta(q_1, b, Z) = \{(q_1, \in)\}$$
$$\delta(q_1, a, Z_0) = \{q_0, Z_0)\}$$

Now, we define G as follows –

$$G = (V, \{a, b\}, P, S)$$

where V consists of S, $[q_0, Z_0, q_0]$, $[q_0, Z_0, q_1]$ $[q_0, Z, q_0]$, $[q_0, Z, q_1]$, $[q_1, Z_0, q_0]$, $[q_1, Z_0, q_1]$, $[q_1, Z, q_0]$, $[q_1, Z, q_0]$.

The productions are –

$$P_1 - S \to [q_0, Z_0, q_0]$$
$$P_2 - S \to [q_0, Z_0, q_1]$$
$$\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\} \text{ yields}$$
$$P_3 - [q_0, Z_0, q_0] \to b[q_0, Z, q_0]\ [q_0, Z_0, q_0]$$
$$P_4 - [q_0, Z_0, q_0] \to b[q_0, Z, q_1]\ [q_1, Z_0, q_0]$$
$$P_5 - [q_0, Z_0, q_1] \to b[q_0, Z, q_0]\ [q_0, Z_0, q_1]$$
$$P_6 - [q_0, Z_0, q_1] \to b[q_0, Z, q_1]\ [q_1, Z_0, q_1]$$
$$\delta(q_0, \lambda, Z_0) = \{q_0, \in)\} \text{ gives}$$
$$P_7 - [q_0, Z_0, q_0] \to \in$$
$$\delta(q_0, b, Z) = \{(q_0, ZZ)\} \text{ gives}$$
$$P_8 - [q_0, Z, q_0] \to b[q_0, Z, q_0]\ [q_0, Z, q_0]$$
$$P_9 - [q_0, Z, q_0] \to b[q_0, Z, q_1]\ [q_1, Z, q_0]$$
$$P_{10} - [q_0, Z, q_1] \to b[q_0, Z, q_0]\ [q_0, Z, q_1]$$
$$P_{11} - [q_0, Z, q_1] \to b[q_0, Z, q_1]\ [q_1, Z, q_1]$$
$$\delta(q_0, a, Z) = \{(q_1, Z)\} \text{ yields}$$
$$P_{12} - [q_0, Z, q_0] \to a[q_1, Z, q_0]$$
$$P_{13} - [q_0, Z, q_1] \to a[q_1, Z, q_1]$$
$$\delta(q, b, Z) = \{(q_1, \in)\} \text{ gives}$$
$$P_{14} - [q_1, Z, q_1] \to b$$
$$\delta(q_1, a, Z_0) = \{(q_0, Z_0)\} \text{ gives}$$
$$P_{15} - [q_1, Z_0, q_0] \to a\ [q_0, Z_0, q_0]$$
$$P_{16} - [q_1, Z_0, q_1] \to a\ [q_0, Z_0, q_1]$$

Thus, $P_1 - P_{16}$ give the productions in P.

Q.15. Prove that, if L = L(M) for some PDA M, then L is a context-free language.

Ans. **Proof** – Assume that $M = (Q, \Sigma, (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \{q_f\})$ satisfies the following conditions –

(i) It has a single final state q_f that is entered iff the stack is empty

(ii) All transitions must have the form $\delta(q_i, a, A) = \{c_1, c_2, \ldots, c_n\}$.

where,
$$c_i = (q_i, \in)$$
or
$$c_i = (q_j, BC).$$

i.e., each move either increases or decreases the stack content by a single symbol.

We use the suggested construction to get the grammar $G = (V, T, S, P)$, with $T = \Sigma$ and V consisting of elements of the form $(q_i\, cq_j)$. We will show that the grammar so obtained is such that for all $q_i, q_j \in Q, A \in \Gamma, X \in \Gamma^*, u, v \in \Sigma^*$,

$$(q_i, uv, AX) \vdash^* (q_j, X) \qquad \ldots(i)$$

implies that $(q_i A q_j) \overset{*}{\Rightarrow} u$, and vice versa.

The first part is to prove that, whenever the PDA is such that the symbol A and its effects can be removed from the stack while scanning u and going from state q_i to q_j, then the variable $(q_i\, A\, q_j)$ can derive u. This is not hard to observe since the grammar was explicitly constructed to do this. We only need an induction on the number of moves to make it precise.

For the converse, consider a single step in the derivation as

$$(q_i A q_k) \Rightarrow a(q_j B q_1)(q_1 C q_k).$$

Using the corresponding transition for the PDA

$$\delta(q_i, a, A) = \{(q_j, BC, \ldots\ldots)\}, \qquad \ldots(ii)$$

we observe that A can be removed from the stack, BC put on, reading a, with the control unit going from state q_i to q_j. In the same way, if

$$(q_i A q_j) \Rightarrow a, \qquad \ldots(iii)$$

then there must be a corresponding transition

$$\delta(q_i, a, A) = \{(q_j, \in)\} \qquad \ldots(iv)$$

whereby the A can be popped off the stack. We observe from this that the sentential forms derived from $(q_i A q_j)$ define a sequence of possible configurations of the PDA by which equation (i) can be achieved.

Notice that $(q_i A q_j) \Rightarrow a(q_j B q_l)(q_l C q_k)$ might be possible for some $(q_j B q_l)$ $(q_l\, C q_k)$ for which there is no corresponding transition of the form (ii) or (iv). But, in that case, at least one of the variables on the right will be useless. For all sentential forms leading to a terminal string the argument given holds.

If we now use the conclusion to

$$(q_0, w, Z_0) \overset{*}{\vdash} (q_f, \in, \in),$$

we see that this can be so if and only if

$$(q_0 Z_0 q_f) \overset{*}{\Rightarrow} w$$

Consequently $L(M) = L(G)$ **Hence Proved**

Q.16. Prove that the class of language accepted by PDA is exactly class of CFG, hence construct PDA for the following grammar –

$$S \to aSa|bSb|a|b|\in$$

What language is represented by it ? ***(R.G.P.V., Dec. 2010)***

***Ans.* Proof** – Refer to Q.15.

PDA – The grammar is not in GNF, therefore, first we obtain an equivalent grammar in GNF, which is

$$S \to aSA|bSB|a|b|\in$$
$$A \to a$$
$$B \to b$$

Therefore, the PDA equivalent to the given grammar is –

$$M = (\{q\}, \{a, b\}, \{S, A, B\}, \delta, q, S, \phi)$$

where, $\delta(q, a, S) = \{(q, SA), (q, \in)\}$, because of $S \to a\,SA$, and $S \to a$

$\delta(q, b, S) = \{(q, SB), (q, \in)\}$ because of $S \to bSB$ and $S \to b$

$\delta(q, a, A) = \{(q, \in)\}$, because of $A \to a$

$\delta(q, b, B) = \{(q, \in)\}$, because of $B \to b$

Language – $L(G) = \{w{:}w \in$ set of all palindromes over $\{a, b\}\}$

Q.17. If L is a context-free language, then construct a PDA A accepting L by empty store, i.e., L = N(A).

Or

Write short note on equivalence of CFGs for PDA.

(R.G.P.V., June 2005)

Ans. We construct the PDA A by making use of productions in G.

Construction of A – Let $L = L(G)$, where $G = (V, T, P, S)$ is a context-free grammar. We construct a PDA A as follows –

$$A = (\{q\}, \Sigma, V \cup T, \delta, q, S, \phi)$$

where δ is defined by the following rules –

$$R_1 - \delta(q, \in, A) = \{(q, \alpha) \mid A \to \alpha \text{ is in } P\}$$
$$R_2 - \delta(q, a, a) = \{(q, \in)\} \text{ for every } a \text{ in } \Sigma$$

The construction can be explained as follows. The pushdown symbols in A are variables and terminals. If the PDA reads a variable A on the top of the stack, it makes a $\in$-move by placing the R.H.S. of any A-production (after erasing A).

If the PDA reads a terminal a on the top of the stack and if it matches with the current input symbol, then the PDA erases a. The PDA halts in other cases.

If $w \in L(G)$ is obtained by a leftmost derivation

$$S \Rightarrow u_1 A_1 \alpha_1 \Rightarrow u_1 u_2 A_2 \alpha_2 \alpha_1 \Rightarrow \ldots \Rightarrow w,$$

then A can empty the stack on application of input string w. The first move of a is by a $\in$-move corresponding to $S \to u_1 A_1 \alpha_1$. The PDA erases S and stores $u_1 A_1 \alpha_1$. Then using R_2, the PDA erases the symbols in u_1 by processing a prefix of w. Now, the topmost symbol in the stack is A_1. Once again by applying the $\in$-move corresponding to $A_1 \to u_2 A_2 \alpha_2$, the PDA erases A_2 and stores $u_2 A_2 \alpha_2$ above α_1. Proceeding in this way, the PDA empties the stack by processing the entire string w. Let us take an example.

Construct a PDA A equivalent to the following CFG – $S \to 0BB$, $B \to 0S|1S|0$. Test whether 010^4 is in $N(A)$.

Now, we define PDA A as follows –

$$A = (\{q\}, \{0, 1\}, \{S, B, 0, 1\}, \delta, q, S, \phi\}$$

where δ is defined by the following rules –

$$R_1 - \delta(q, \in, S) = \{(q, 0BB)\}$$
$$R_2 - \delta(q, \in, B) = \{(q, 0S), (q, 1S), (q, 0)\}$$
$$R_3 - \delta(q, 0, 0) = \{(q, \in)\}$$
$$R_4 - \delta(q, 1, 1) = \{(q, \in)\}$$

Let us test whether 010^4 is in $N(A)$ or not.

$(q, 010^4, S) \vdash (q, 010^4, 0BB)$ by rule R_1

$\vdash (q, 10^4, BB)$ by rule R_3

$\vdash (q, 10^4, 1SB)$ by rule R_2, since $(q, 1S) \in \delta(q, \in, B)$

$\vdash (q, 0^4, SB)$ by rule R_4

$\vdash (q, 0^4, 0BBB)$ bu rule R_1

$\vdash (q, 0^3, BBB)$ by rule R_2, since $(q, 0) \in \delta(q, \in, B)$

$\overset{*}{\vdash} (q, \in, \in)$ by rule R_3

Thus, $010^4 \in N(A)$

Note that after entering $(q, 10^4, BB)$, the PDA may halt for a different sequence of moves, for example, $(q, 10^4, BB) \vdash (q, 10^4, 0B) \vdash (q, 10^4, 00)$. As $\delta(q, 1, 0)$ is the empty set, the PDA halts.

Q.18. Prove that, for any context-free language L, there exists a PDA M such that –

$$L = L(M)$$

Or

Prove that every context-free grammar is equivalent to PDA.

(R.G.P.V., Dec. 2017)

If we now use the conclusion to

$$(q_0, w, Z_0) \overset{*}{\vdash} (q_f, \in, \in),$$

we see that this can be so if and only if

$$(q_0 Z_0 q_f) \overset{*}{\Rightarrow} w$$

Consequently

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

Hence Proved

Q.16. Prove that the class of language accepted by PDA is exactly class of CFG, hence construct PDA for the following grammar –

$$S \to aSa|bSb|a|b|\in$$

What language is represented by it ?

(R.G.P.V., Dec. 2010)

Ans. **Proof** – Refer to Q.15.

PDA – The grammar is not in GNF, therefore, first we obtain an equivalent grammar in GNF, which is

$$S \to aSA|bSB|a|b|\in$$
$$A \to a$$
$$B \to b$$

Therefore, the PDA equivalent to the given grammar is –

$$M = (\{q\}, \{a, b\}, \{S, A, B\}, \delta, q, S, \phi)$$

where, $\delta(q, a, S) = \{(q, SA), (q, \in)\}$, because of $S \to aSA$, and $S \to a$

$\delta(q, b, S) = \{(q, SB), (q, \in)\}$ because of $S \to bSB$ and $S \to b$

$\delta(q, a, A) = \{(q, \in)\}$, because of $A \to a$

$\delta(q, b, B) = \{(q, \in)\}$, because of $B \to b$

Language – $L(G) = \{w : w \in$ set of all palindromes over $\{a, b\}\}$

Q.17. If L is a context-free language, then construct a PDA A accepting L by empty store, i.e., L = N(A).

Or

Write short note on equivalence of CFGs for PDA.

(R.G.P.V., June 2005)

Ans. We construct the PDA A by making use of productions in G.

Construction of A – Let $L = L(G)$, where $G = (V, T, P, S)$ is a context-free grammar. We construct a PDA A as follows –

$$A = (\{q\}, \Sigma, V \cup T, \delta, q, S, \phi)$$

where δ is defined by the following rules –

$$R_1 - \delta(q, \in, A) = \{(q, \alpha) \mid A \to \alpha \text{ is in } P\}$$
$$R_2 - \delta(q, a, a) = \{(q, \in)\} \text{ for every } a \text{ in } \Sigma$$

The construction can be explained as follows. The pushdown symbols in A are variables and terminals. If the PDA reads a variable A on the top of the stack, it makes a $\in$-move by placing the R.H.S. of any A-production (after erasing A)

If the PDA reads a terminal a on the top of the stack and if it matches with the current input symbol, then the PDA erases a. The PDA halts in other cases.

If $w \in L(G)$ is obtained by a leftmost derivation

$$S \Rightarrow u_1 A_1 \alpha_1 \Rightarrow u_1 u_2 A_2 \alpha_2 \alpha_1 \Rightarrow \ldots \Rightarrow w,$$

then A can empty the stack on application of input string w. The first move of A is by a $\in$-move corresponding to $S \to u_1 A_1 \alpha_1$. The PDA erases S and stores $u_1 A_1 \alpha_1$. Then using R_2, the PDA erases the symbols in u_1 by processing a prefix of w. Now, the topmost symbol in the stack is A_1. Once again by applying the $\in$-move corresponding to $A_1 \to u_2 A_2 \alpha_2$, the PDA erases A_2 and stores $u_2 A_2 \alpha_2$ above α_1. Proceeding in this way, the PDA empties the stack by processing the entire string w. Let us take an example.

Construct a PDA A equivalent to the following CFG – $S \to 0BB$, $B \to 0S|1S|0$. Test whether 010^4 is in $N(A)$.

Now, we define PDA A as follows –

$$A = (\{q\}, \{0, 1\}, \{S, B, 0, 1\}, \delta, q, S, \phi\}$$

where δ is defined by the following rules –

$$R_1 - \delta(q, \in, S) = \{(q, 0BB)\}$$
$$R_2 - \delta(q, \in, B) = \{(q, 0S), (q, 1S), (q, 0)\}$$
$$R_3 - \delta(q, 0, 0) = \{(q, \in)\}$$
$$R_4 - \delta(q, 1, 1) = \{(q, \in)\}$$

Let us test whether 010^4 is in $N(A)$ or not.

$(q, 010^4, S)$	$\vdash (q, 010^4, 0BB)$	by rule R_1
	$\vdash (q, 10^4, BB)$	by rule R_3
	$\vdash (q, 10^4, 1SB)$	by rule R_2, since $(q, 1S) \in \delta(q, \in, B)$
	$\vdash (q, 0^4, SB)$	by rule R_4
	$\vdash (q, 0^4, 0BBB)$	bu rule R_1
	$\vdash (q, 0^3, BBB)$	by rule R_2, since $(q, 0) \in \delta(q, \in, B)$
	$\overset{*}{\vdash} (q, \in, \in)$	by rule R_3

Thus, $010^4 \in N(A)$

Note that after entering $(q, 10^4, BB)$, the PDA may halt for a different sequence of moves, for example, $(q, 10^4, BB) \vdash (q, 10^4, 0B) \vdash (q, 10^4, 00)$. As $\delta(q, 1, 0)$ is the empty set, the PDA halts.

Q.18. Prove that, for any context-free language L, there exists a PDA M such that –

$$L = L(M)$$

Or

Prove that every context-free grammar is equivalent to PDA.

(R.G.P.V., Dec. 2017)

***Ans.* Proof** – If L is a $\in$-free context-free language, there exists a conext-free grammar in Greibach normal form for it. Let $G = (V, T, P, S)$ be such a grammar. We then construct a PDA which simulates leftmost derivations in this grammar. The simulation will be done in such a way, that the unprocessed part of the sentential form is in the stack, while the terminal prefix of any sentential form matches the corresponding prefix of the input string.

Specifically, the PDA will be

$$M = (\{q_0, q_1, q_f\}, \Sigma, V \cup \{Z_0\}, \delta, q_0, Z_0, \{q_f\}),$$

where $\Sigma = T$ and $Z_0 \notin V$. Note that the input alphabets of M are identical with the set of terminals of G and that the stack alphabets contains the set of variables of the grammar.

The transition function will have

$$\delta(q_0, \in, Z_0) = \{(q_1, SZ_0)\} \quad \ldots(i)$$

so that after the first move of M, the stack contains the start symbol S of the derivation. (The stack start symbol Z_0 is a marker to allow us to detect the end of the derivation.) In addition, the set of transition rules is such that

$$(q_1, u) \in \delta(q_1, a, A), \quad \ldots(ii)$$

whenever $A \to au$ is in P. This reads input a and removes the variable A from the stack, replacing it with u. In this way, it generates the transitions that allow the PDA to simulate all derivations. Finally, we have

$$\delta(q_1, \in, Z_0) = \{(q_f, Z_0)\}, \quad \ldots(iii)$$

to get M into a final state.

To show that M accepts any $w \in L(G)$, consider the partial leftmost derivation as follows –

$$S \overset{*}{\Rightarrow} a_1a_2 \ldots a_nA_1A_2\ldots A_m$$
$$\Rightarrow a_1a_2\ldots a_nbB_1\ldots B_k A_2\ldots A_m.$$

If M is to simulate this derivation, then after scanning $a_1a_2\ldots a_n$, the stack must contain $A_1A_2\ldots A_m$. To take the next step in the derivation, G must have a production as follows –

$$A_1 \to bB_1\ldots B_k.$$

But the construction is such that then M has a transition rule in which

$$(q_1, B_1\ldots B_k) \in \delta(q_1, b, A_1)$$

Thus the stack now contains $B_1\ldots B_k A_2\ldots A_m$ after having read $a_1a_2\ldots a_nb$.

A simple induction principle on the number of steps in the derivation then shows that if, $S \overset{*}{\Rightarrow} w$,

then $(q_1, w, SZ_0) \overset{*}{\vdash} (q_1, \in, Z_0)$

Using rules (i) and (iii) we have

$$(q_0, w, Z_0) \vdash (q_1, w, SZ_0) \overset{*}{\vdash} (q_1, \in, Z_0) \vdash (q_f, \in, Z_0)$$

so that $L(G) \subseteq L(M)$.

To prove that $L(M) \subseteq L(G)$, let $w \in L(M)$. Then by definition

$$(q_0, w, Z_0) \overset{*}{\vdash} (q_f, \in, u)$$

But there is only one way to get from q_0 to q_1 and only one way from q_1 to q_f. Therefore, we must have the following –

i.e., $(q_1, w, SZ_0) \overset{*}{\vdash} (q_1, \lambda, Z_0)$

Now let us write $w = a_1a_2a_3\ldots a_n$. Then the first step in

$$(q_1, a_1a_2a_3\ldots a_n, SZ_0) \overset{*}{\vdash} (q_1, \in, Z_0) \quad \ldots(iv)$$

must be a rule of the form (ii) to get

$$(q_1, a_1a_2a_3\ldots a_n, SZ_0) \vdash (q_1, a_2a_3\ldots a_n, u_1Z_0).$$

But then the grammar has a rule of the form $S \to a_1u_1$, so that

$$S \Rightarrow a_1u_1.$$

Repeating this, writing $u_1 = Au_2$, we have the following –

$$(q_1, a_2a_3\ldots a_n, Au_2Z_0) \vdash (q_1, a_3\ldots a_n, u_3u_2Z_0),$$

It implies that $A \to a_2u_3$ is in the grammar and that

$$S \overset{*}{\Rightarrow} a_1a_2u_3u_2.$$

This makes it quite clear at any point the stack contents (excluding Z_0) are identical with the unmatched part of the sentential form, so that (iv) implies

$$S \overset{*}{\Rightarrow} a_1a_2\ldots a_n.$$

In consequence, $L(M) \subseteq L(G)$, completing the proof if the language does not contain $\in$.

If $\in \in L$, we add to the constructed PDA the transition –

$$\delta(q_0, \in, Z_0) = \{(q_f, Z_0)\}$$

so that the empty string is also accepted.

Q.19. What is petrinet model ? Describe it.

Ans. A petrinet is also known as Place / Transition (PT) net. Petrinet is one of the several mathematical modelling languages. Petrinet is a directed bipartite graph, in which nodes represent transitions (i.e. events that may occur, represented by bars and places (i.e. conditions, represented by circles).

In theory of computation a petrinet is defined as 5-tuple –

$$P = \{P, T, F, W, M_0\}$$

where, $P = \{p_1, p_2, \ldots, p_m\}$

is a finite set of places,

$$T = \{t_1, t_2, \ldots, t_m\}$$

is a finite set of transitions,

$$F \subseteq (P \times T) \cup (T \times P)$$

is a set of arcs (flow relation),

$$W : F \rightarrow \{1, 2, 3 \ldots..\}$$

is a weight function,

$$M_0 : P \rightarrow \{0, 1, 2, 3 \ldots..\}$$

is the initial marking,

$$P \cap T = \phi, P \cup T \neq \phi$$ (if img(P) = {0, 1}, we have a elementary net, otherwise an integer net)

A petrinet structure N = (P, T, W) without any specific initial marking is denoted by N, and a petrinet with the given initial marking is denoted by (N, M_0).

In a petrinet places represent conditions, states, or predicates.

In petrinet transitions represent the firing of events. The firing of events is represented as follows –

(i) If a transition has one input place, then the event fires immediately if a token arrives in that place.

(ii) If a transition has several input places, the event fires when all input places have tokens.

NUMERICAL PROBLEMS

Prob.1. Construct a PDA A accepting the set of all strings over {a, b} having equal number of a's and b's.

Or

Design a PDA which accepts the language L = {W ∈(a, b)*/W has the equal number of a's and b's}. ***(R.G.P.V., Dec. 2013)***

Or

Design a pushdown automata which accepts equal number of a's and b's over Σ = {a, b}. Also illustrate the moves of PDA for any input string. ***(R.G.P.V., Nov. 2018)***

Sol. Refer to Q.6.

Prob.2. Design a PDA to accept the following language –

$$\{0^n 1^n \mid n \geq 1\}.$$

(R.G.P.V., Dec. 2008)

Or

Define pushdown automata. Draw PDA for the language $\{0^n 1^n | n \geq 0\}$. ***(R.G.P.V., Dec. 2007)***

Sol. **Pushdown Automata** – Refer to Q.1.

Let $A = (\{q_0, q_1\}, \{0, 1\}, \{0, Z_0\}, \delta, q_0, Z_0, \phi)$

where δ is given by

$$\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$$
$$\delta(q_0, 0, 0) = \{(q_0, 00)\}$$
$$\delta(q_0, 1, 0) = \{(q_1, \in)\}$$
$$\delta(q_1, 1, 0) = \{(q_1, \in)\}$$
$$\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$$

The PDA will start in state q_0 and go on pushing 0's onto stack without changing its state, until it gets 1. When it gets 1, it changes state to q_1 and pops 0 from the top of the stack. Then for every 1 appearing next in the input it simply pops 0 from the top of the stack, without changing a state. When it reaches end of the input, if top of the stack is Z_0,then it empties its stack.

Prob.3. Construct a PDA for accepting language $L = \{a^n b^n \; n \geq 1\}$ by the null stack. ***(R.G.P.V., June 2017)***

Sol. Refer to Prob.2.

Note – Replace 0 by *a* and 1 by *b*.

Prob.4. Construct PDA for the set $L = \{a^n b^{2n} \mid n \geq 1\}$. ***(R.G.P.V., June 2008, Dec. 2010)***

Or

Design a PDA to accept the language $L = a^n b^{2n} \mid n \geq 1$. ***(R.G.P.V., Dec. 2017)***

Or

Design pushdown automata which accepts $L = \{0^n 1^{2n} \mid n \geq 1\}$. ***(R.G.P.V., June 2016)***

Sol. The required PDA *A* is defined as follows –

$A = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$, where δ is defined by

$$\delta(q_0, a, Z_0) = \{(q_1, aZ_0)\}$$
$$\delta(q_1, a, a) = \{(q_1, aa)\}$$
$$\delta(q_1, b, a) = \{(q_2, a)\}$$
$$\delta(q_2, b, a) = \{(q_1, \in)\}$$
$$\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$$

Let us see the acceptability for the string *aabbbb*. The sequence of moves in accepting *aabbbb* is

$$(q_0, aabbbb, Z_0) \vdash (q_1, abbbb, aZ_0)$$
$$\vdash (q_1, bbbb, aaZ_0)$$
$$\vdash (q_2, bbb, aaZ_0)$$
$$\vdash (q_1, bb, aZ_0)$$

$$\vdash (q_2, b, aZ_0)$$
$$\vdash (q_1, \in, Z_0)$$
$$\vdash (q_1, \in, \in)$$

Thus, $N(A) = \{a^n b^{2n} | n \geq 1\}$

Prob.5. Construct PDA for the set $L = \{a^{2n} b^n | n \geq 1\}$.

(R.G.P.V., Dec. 2003, 2010, June 2011)

Sol. The required PDA A is defined as follows –

$A = (\{q_0, q_1\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$, where δ is defined by

$$\delta(q_0, a, Z_0) = \{(q_1, aZ_0)\}$$
$$\delta(q_0, a, a) = \{(q_1, aa)\}$$
$$\delta(q_1, a, a) = \{(q_0, a)\}$$
$$\delta(q_0, b, a) = \{(q_0, \in)\}$$
$$\delta(q_0, \in, Z_0) = \{(q_0, \in)\}$$

Let us see the acceptability for the string *aaaabb*. The sequence of moves in accepting the string is –

$$(q_0, aaaabb, Z_0) \vdash (q_1, aaabb, aZ_0)$$
$$\vdash (q_0, aabb, aZ_0)$$
$$\vdash (q_1, abb, aaZ_0)$$
$$\vdash (q_0, bb, aaZ_0)$$
$$\vdash (q_0, b, aZ_0)$$
$$\vdash (q_0, \in, Z_0)$$
$$\vdash (q_0, \in, \in)$$

Thus, $N(A) = \{a^{2n} b^n | n \geq 1\}$

Prob.6. Construct PDA for the following CFG –
$L = \{a\, b^n\, c\, d^m \mid n \geq 1\}$.

(R.G.P.V., June 2007)

Sol. The PDA accepting $\{ab^n cd^n \mid n \geq 1\}$ is defined as follows –

$$A = (\{q_0, q_1, q_2\}, \{a, b, c, d\}, \{b, Z_0\}, \delta, q_0, Z_0\ \phi\}$$

where, δ is given by,

$$\delta(q_0, a, Z_0) = \{(q_1, Z_0)\}$$
$$\delta(q_1, b, Z_0) = \{(q_1, bZ_0)\}$$
$$\delta(q_1, b, b) = \{(q_1, bb)\}$$
$$\delta(q_1, c, b) = \{(q_2, b)\}$$
$$\delta(q_2, d, b) = \{(q_2, \in)\}$$
$$\delta(q_2, \in, Z_0) = \{(q_2, \in)\}$$

Prob.7. Construct PDA for the following language –
$L = \{a^m\, b^n\, c^{m+n} \mid m, n \geq 1\}$.

(R.G.P.V., Dec. 2006, 2014, June 2015)

Sol. The PDA accepting $\{a^m\, b^n\, c^{m+n} \mid m, n \geq 1\}$ is defined as follows –

$$A = (\{q_0, q_1, q_2, q_3\}, \{a, b, c\}, \{Z_2, Z_1, Z_0\}, \delta, q_0, Z_0\ \phi)$$

where, δ is given by,

$$\delta(q_0, a, Z_0) = \{(q_1, Z_1Z_0)\}$$
$$\delta(q_1, a, Z_1) = \{(q_1, Z_1Z_1)\}$$
$$\delta(q_1, b, Z_1) = \{(q_2, Z_2Z_1)\}$$
$$\delta(q_2, b, Z_2) = \{(q_2, Z_2Z_2)\}$$
$$\delta(q_2, c, Z_2) = \{(q_3, \in)\}$$
$$\delta(q_3, c, Z_2) = \{(q_3, \in)\}$$
$$\delta(q_3, c, Z_1) = \{(q_3, \in)\}$$
$$\delta(q_3, \in, Z_0) = \{(q_3, \in)\}$$

Prob.8. Construct PDA accepting $L = \{a^i b^j \mid j = i \text{ or } j = 2i\}$.

(R.G.P.V., June 2009)

Sol. The logic that we use for the PDA to be constructed is – the PDA will start in state q_0, and either go on pushing the a's onto the stack without changing its state, until it gets b, or will push two a's first time when it gets a, in the input and change a state to q_1 and then go on pushing two a's for every a appearing next in the input remaining in state q_1, until it gets b. When it gets b it pops a from the top of the stack advances the tape head one position right and changes the state to q_2 irrespective of whether it is presently in state q_0 or q_1. Then for every b appearing next in the input it simply pops a from the top of the stack, without changing a state. When it reaches end of the input, if top of the stack is Z_0, then it empties its stack. Therefore, the moves of the PDA are –

$\delta(q_0, \in, Z_0) = \{(q_2, \in)\}$ To take care of acceptance of empty string $\in$.

$$\delta(q_0, a, Z_0) = \{(q_0, aZ_0), (q_1, aaZ_0)\}$$
$$\delta(q_0, a, a) = \{(q_0, aa)\}$$
$$\delta(q_1, a, a) = \{(q_1, aaa)\}$$
$$\delta(q_0, b, a) = \{(q_2, \in)\}$$
$$\delta(q_1, b, a) = \{(q_2, \in)\}$$
$$\delta(q_2, b, a) = \{(q_2, \in)\}$$
$$\delta(q_2, \in, Z_0) = \{(q_2, \in)\}$$

Therefore, PDA is –

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$$

where δ is given above.

Prob.9. Construct npda's that accept the language

$$L = \{a^n b^m : n \le m \le 3n\}$$
$$\text{on } \Sigma = \{a, b\}$$

Sol. The required NPDA can be defined as follows –

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

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

$\Gamma = \{a, b, Z_0\}$

And δ is defined by

$$\delta(q_0, a, Z_0) = (q_1, aZ_0)$$
$$\delta(q_1, a, a) = (q_0, aa)$$
$$\delta(q_1, b, a) = \{(q_1, \in), (q_2, a)\}$$
$$\delta(q_2, b, a) = \{(q_1, \in), (q_3, a)\}$$
$$\delta(q_3, b, a) = \{(q_1, \in)\}$$
$$\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$$
$$\delta(q_0, \in, Z_0) = \{(q_1, \in)\}$$

Prob.10. Construct a PDA accepting $\{a^n b^m a^n / m, n \ge 1\}$ by empty store. ***(R.G.P.V., June 2006)***

Or

Design PDA to accept the language –

$$L(G) = \{a^n b^m a^n / m, n \ge 1\}$$ ***(R.G.P.V., Dec. 2016)***

Sol. The PDA A accepting $\{a^n b^m a^n / m, n \ge 1\}$ is defined as follows –

$$A = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$$

where δ is given by

$$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$
$$\delta(q_0, a, a) = \{(q_0, aa)\}$$
$$\delta(q_0, b, a) = \{(q_1, a)\}$$
$$\delta(q_1, b, a) = \{(q_1, a)\}$$
$$\delta(q_1, a, a) = \{(q_2, \in)\}$$
$$\delta(q_2, a, a) = \{(q_2, \in)\}$$
$$\delta(q_2, \in, Z_0) = \{(q_2, \in)\}$$

Prob.11. Design DPDA for $L = \{0^n 1^m 0^n / m, n \ge 1\}$. (R.G.P.V., June 2011)

Sol. Refer to Prob.10. (assume $a = 0$ and $b = 1$)

Prob.12. Design a PDA that accepts the language $L = \{a^n b^m a^n : m, n > 0\}$. ***(R.G.P.V., Nov. 2019)***

Sol. Refer to Prob.10.

Prob.13. Design PDA for the language –

$$L = \{w | w \in (a + b)^* \text{ and } n_a(w) > n_b(w)\}$$ ***(R.G.P.V., Dec. 2011)***

Or

Give transition table for deterministic PDA recognizing the following language –

$$L = \{x \in \{a, b\}^* \mid n_a(x) > n_b(x)\}$$ ***(R.G.P.V., Dec. 2005)***

Or

Design a PDA to accept the language $\{x \in (a, b)^* | n_a(x) > n_b(x)\}$. ***(R.G.P.V., Dec. 2015)***

Sol. The required PDA M for language

$L = \{w | w \in (a + b)^* \text{ and } n_a(w) > n_b(w)\}$ is defined as –

$$M = (\{q_0, q_f\}, \{a, b\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_f\})$$

where δ is defined as –

$$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$
$$\delta(q_0, b, Z_0) = \{(q_0, bZ_0)\}$$
$$\delta(q_0, a, a) = \{(q_0, aa)\}$$
$$\delta(q_0, b, b) = \{(q_0, bb)\}$$
$$\delta(q_0, a, b) = \{(q_0, \in)\}$$
$$\delta(q_0, b, a) = \{(q_0, \in)\}$$
$$\delta(q_0, \in, a) = \{(q_f, a)\}$$

Prob.14. Design a pushdown automata which accepts set of balanced parantheses. $(\{\{(\ )\}\})$ ***(R.G.P.V., June 2016)***

Sol. The pushdown automata is given by –

$$A = (\{q_0\}, \{(, \{, \},)\}, \{c, \{\ \}, \delta, q_0, Z_0, \phi\}$$

where δ is given by –

$$\delta(q_0, \{, Z_0) = (q_0, \{\ Z_0)$$
$$\delta(q_0, (, Z_0) = (q_0, (\ Z_0)$$
$$\delta(q_0, \{, \{\ ) = (q_0, \{\{\ )$$
$$\delta(q_0, (, (\ ) = (q_0, ((\ )$$
$$\delta(q_0, \{, (\ ) = (q_0, \{(\ )$$
$$\delta(q_0, (, \{\ ) = (q_0, (\{\ )$$
$$\delta(q_0, \}, \{\ ) = (q_0, \in)$$
$$\delta(q_0,), (\ ) = (q_0, \in)$$
$$\delta(q_0, \in, Z_0) = (q_0, \in)$$

Prob.15. Construct the PDA that accepts $\{wcw^R \mid w \in (0 + 1)^*\}$ by empty stack, where w^R represents the reverse of w. ***(R.G.P.V., June 2004)***

Sol. We define the PDA A as follows –

$A = (\{q_0, q_1, q_f\}, \{0, 1, c\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_f\})$ where δ is defined as –

$$\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$$
$$\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$$
$$\delta(q_0, 0, 0) = \{(q_0, 00)\}$$

$$\delta(q_0, 1, 0) = \{(q_0, 10)\}$$
$$\delta(q_0, 0, 1) = \{(q_0, 01)\}$$
$$\delta(q_0, 1, 1) = \{(q_0, 11)\}$$
$$\delta(q_0, c, 0) = \{(q_1, 0)\}$$
$$\delta(q_0, c, 1) = \{(q_1, 1)\}$$
$$\delta(q_0, c, Z_0) = \{(q_1, Z_0)\}$$
$$\delta(q_1, 0, 0) = \{(q_1, \in)\}$$
$$\delta(q_1, 1, 1) = \{(q_1, \in)\}$$
$$\delta(q_1, \in, Z_0) = \{(q_f, Z_0)\}$$

We observe that from the construction of A, these rules cannot erase Z_0. For erasing Z_0 from the stack we consider an additional rule –

$$\delta(q_f, \in, Z_0) = \{(q_f, \in)\}$$

By this rule stack can be emptied by $\in$-moves iff the PDA reaches the ID $(q_f, \in, Z_0)$. Hence,

$$N(A) = \{wcw^R | w \in \{0, 1\}^*\}.$$

Prob.16. Construct PDA that accepts language $L = \{WW^R \mid W \text{ in } (0 + 1)^*\}$. **(R.G.P.V., Dec. 2012)**

Or

Design PDA to accept the language $L(G) = \{ww^R / w \in (0, 1)^* \text{ and } w^R \text{ is the reverse of word } w\}$. **(R.G.P.V., Dec. 2016)**

Sol. We define the PDA A as follows –

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

where δ is defined as –

$$\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$$
$$\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$$
$$\delta(q_0, 0, 0) = \{(q_0, 00)\}$$
$$\delta(q_0, 1, 0) = \{(q_0, 10)\}$$
$$\delta(q_0, 0, 1) = \{(q_0, 01)\}$$
$$\delta(q_0, 1, 1) = \{(q_0, 11)\}$$
$$\delta(q_0, \in, 0) = \{(q_1, 0)\}$$
$$\delta(q_0, \in, 1) = \{(q_1, 1)\}$$
$$\delta(q_0, \in, Z_0) = \{(q_1, Z_0)\}$$
$$\delta(q_1, 0, 0) = \{(q_1, \in)\}$$
$$\delta(q_1, 1, 1) = \{(q_1, \in)\}$$
$$\delta(q_1, \in, Z_0) = \{(q_2, Z_0)\}$$

Thus, $N(A) = \{WW^R | W \text{ in } (0 + 1)^*\}$

or $\{WW^R / W \in (0, 1)^*\}$

Prob.17. Construct a PDA for accepting the language $L = \{ww^R \mid w \in \{a,b\}^+\}$

Also parse the string abba.

Or

Make a PDA accepting the language of palindromes. **(R.G.P.V., Dec. 2005)**

Or

Construct a PDA for set of palindrome over the alphabet $\{a, b\}$ $L(M) = \{WcW^R\}$. **(R.G.P.V., May 2018)**

Sol. We use the fact that the symbols are retrieved from a stack in the reverse order of their insertion. When scanning the first part of the string we push consecutive symbol on the stack. For the second part, we compare the current input symbol with the symbol on the top of the stack and keep continuing as long as the two match. Since symbols are retrieved from the stack in reverse of the order in which they were inserted, a complete match will be achieved iff the input is of the form ww^R.

The difficulty with this idea is that we do not know the middle of the strings, i.e., where w ends and w^R starts. But the nondeterministic nature of the automaton helps us with this. The PDA correctly guesses where the middle is and switches states at that point. The solution to the problem is given as –

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where $Q = \{q_0, q_1, q_2\}$,
$\Sigma = \{a, b\}$,
$\Gamma = \{a, b, Z_0\}$,
$F = \{q_2\}$.

The transition functions can be defined as – A set to push w on the stack.

$$\delta(q_0, a, a) = \{(q_0, aa)\},$$
$$\delta(q_0, b, a) = \{(q_0, ba)\},$$
$$\delta(q_0, a, b) = \{(q_0, ab)\},$$
$$\delta(q_0, b, b) = \{(q_0, bb)\},$$
$$\delta(q_0, a, Z_0\} = \{(q_0, aZ_0)\},$$
$$\delta(q_0, b, Z_0) = \{(q_0, bZ_0)\},$$

A set to guess the middle of the strings, where the PDA switches from state q_0 to q_1.

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

A set to match w^R against the content of the stack

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

and finally, $\delta(q_1, \in, Z_0) = \{(q_2, Z_0)\}$.

These are the rules to recognize a successful match for ww^R.

The sequence of moves in accepting abba is

$$\begin{aligned}(q_0, abba, Z_0) &\vdash (q_0, baa, aZ_0)\\ &\vdash (q_0, ba, baZ_0)\\ &\vdash (q_1, ba, baZ_0)\\ &\vdash (q_1, a, aZ_0)\\ &\vdash (q_1, \in, Z_0)\\ &\vdash (q_2, \in, Z_0)\end{aligned}$$

The nondeterministic alternative for locating the middle of the string is taken at the third move. At that stage, the PDA is in ID (q_0, ba, baZ_0) and has two choices for its next move. One is to use $\delta(q_0, b, b) = \{(q_0, bb)\}$ and make the move.

$$(q_0, ba, baZ_0) \vdash (q_0, a, bbaZ_0),$$

the second is the one used above, namely $\delta(q_0, \in, b) = \{(q_1, b)\}$. Only the later leads to acceptance of the inputs. Hence,

$$N(A) = \{ww^R | w \in \{a, b\}^+\}$$

Prob.18. Consider a PDA A such that

$$A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = (a, Z_0), F = \phi,$$

where

and δ is given by

$$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$
$$\delta(q_0, a, a) = \{(q_0, aa)\}$$
$$\delta(q_0, b, a) = \{(q_1, \in)\}$$
$$\delta(q_1, b, a) = \{(q_1, \in)\}$$
$$\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$$

Find the language accepted by PDA and explain your answer.

Sol. The given PDA is defined as follows –

$A = (\{q_0, q_1\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$, where δ is given by –

$$R_1 - \delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$
$$R_2 - \delta(q_0, a, a) = \{(q_0, aa)\}$$
$$R_3 - \delta(q_0, b, a) = \{(q_1, \in)\}$$
$$R_4 - \delta(q_1, b, a) = \{(q_1, \in)\}$$
$$R_5 - (q_1, \in, Z_0) = \{(q_1, \in)\}$$

R_1 is used to store a in the stack if it is the first symbol of an input string. R_2 can be used repeatedly to store a^n in the stack. When b is encountered for the first time in the input string, a is erased (in the stack) using R_3. Also, the PDA makes a transition to state q_1. After processing the entire input string, if Z_0 remains in the stack, it can be erased using the null move given by R_5. So,

if $u = a^n b^n$, then we have

$$\begin{aligned}(q_0, a^n b^n, Z_0) &\overset{*}{\vdash} (q_0, b^n, a^n Z_0) && \text{by applying } R_1 \text{ and } R_2\\ &\overset{*}{\vdash} (q_1, \in, Z_0) && \text{by applying } R_3 \text{ and } R_4\\ &\vdash (q_1, \in, \in) && \text{by applying } R_5\end{aligned}$$

Therefore, $\quad \boldsymbol{a^n b^n \in N(A)}$.

If $u \in N(A)$, then $(q_0, u, Z_0) \overset{*}{\vdash} (q_1, \in, \in)$ (Note that the stack can be empty only when A is in state q_1). Also, u should start with a. Otherwise, we cannot make any move. We store the symbol a in the stack if the current input symbol is a and the top most symbol in the stack is a or Z_0. On looking the input symbol b, the PDA erases the symbol a in the stack. The PDA enters the ID $(q_1, \in, \in)$only by the application of R_5. The PDA can reach the ID $(q_1, \in, Z_0)$ only by erasing the a's in the stack. This is possible only when the number of b's is equal to number of a's, and so $u = a^n b^n$. Thus, we have proved that

$$N(A) = \{a^n b^n | n \geq 1\}.$$

Now, let $\quad B = (Q', \{a,b\}, \Gamma', \delta_B, q'_0, Z'_0, F')$

where $Q' = \{q_0, q'_0, q_1, q_f\}$,

$$F' = \{q_f\}, \Gamma' = \{a, b, Z'_0\}$$

and δ_B is defined by –

$$\delta_B(q'_0, \in, Z'_0) = \{(q_0, Z_0 Z'_0)\}$$
$$\delta_B(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$
$$\delta_B(q_0, a, a) = \{(q_0, aa)\}$$
$$\delta_B(q_0, b, a) = \{(q_1, \in)\}$$
$$\delta_B(q_1, b, a) = \{(q_1, \in)\}$$
$$\delta_B(q_1, \in, Z_0) = \{(q_1, \in)\}$$
$$\delta_B(q_0, \in, Z'_0) = \{(q_f, \in)\}$$
$$\delta_B(q_1, \in, Z'_0) = \{(q_f, \in)\}$$

Thus, $\quad \boldsymbol{L(B) = N(A) = \{a^n b^n | n \geq 1\}}$

Prob.19. For the given CFG, construct a PDA that accepts the same language they generate –

$$S \rightarrow Saa \mid aSa \mid aaS.$$

(R.G.P.V., Dec. 2009)

Sol. The PDA $A = (\{q\}, \{a\}, \{S\}, \delta, q, S, \phi)$

where δ is defined as follows –

$$R_1 : \delta(q, \in, S) = \{(q, Saa), (q, aSa), (aaS)\}$$
$$R_2 : \delta(q, a, a) = \{(q, \in)\}$$

Prob.20. Design PDA corresponding to given CFG –

$S \to aSA,\ S \to bSb,\ S \to a$ **(R.G.P.V., Dec. 2013)**

Sol. The grammar is not in GNF. Hence, first we get an equivalent grammar in GNF, which is –

$$S \to aSA|bSB|a,\quad A \to a,\quad B \to b$$

Hence, the PDA equivalent to the grammar is –

$$M = (\{q\}, \{a, b\}, \{S, A, B\}, \delta, q, S, \phi)$$

where, $\delta(q, a, S) = \{(q, SA), (q, \in)\}$, because of $S \to aSA$, and $S \to a$

$\delta(q, b, S) = \{q, SB\}$, because of $S \to bSB$

$\delta(q, a, A) = \{(q, \in)\}$, because of $A \to a$

$\delta(q, b, B) = \{(q, \in)\}$, because of $B \to b$

Prob.21. Construct a PDA equivalent to the following context-free grammar –

$S \to 0BB$

$B \to 0S \mid 1S \mid 0$

Test whether 010000 is in N (PDA). **(R.G.P.V., June 2006)**

Or

Construct a PDA equivalent to the following grammar –

$S \to 0BB$

$B \to 0S \mid 1S \mid 0$

Test whether 010^4 is in N (A). **(R.G.P.V., June 2017)**

Sol. Define PDA A as follows –

$$A = (\{q\}, \{0, 1\}, \{S, B, 0, 1\}, \delta, q, S, \phi)$$

δ is defined by the following rules –

$R_1 : \delta(q, \in, S) = \{(q, 0BB)\}$

$R_2 : \delta(q, \in, B) = \{(q, 0S), (q, 1S), (q, 0)\}$

$R_3 : \delta(q, 0, 0) = \{(q, \in)\}$

$R_4 : \delta(q, 1, 1) = \{(q, \in)\}$

$(q, 010000, S) \vdash (q, 010000, 0BB)$ by rule R_1

$\vdash (q, 10000, BB)$ by rule R_3

$\vdash (q, 10000, 1SB)$ by rule R_2 Since $(q, 1S) \in \delta(q, \in, B)$

$\vdash (q, 0000, SB)$ by rule R_4

$\vdash (q, 0000, 0BBB)$ by rule R_1

$\vdash (q, 000, BBB)$ by rule R_3

$\vdash^* (q, 000, 000)$ by rule R_2 Since $(q, 0) \in \delta(q, \in, B)$

$\vdash^* (q, \in, \in)$ by rule R_3

Thus, $010000 \in N$ (PDA).

Prob.22. Construct the PDA equivalent to the following grammar –

$S \to aAA$

$A \to aS|bS|a$

(R.G.P.V., June 2003, Dec. 2006, June 2009, 2010, Dec. 2012)

Sol. Refer to Prob.20.

Note – Here, replace a by 0, A by B and b by 1.

Prob.23. Construct PDA for the following CFG –

$S \to aB \mid bA$

$A \to a \mid aS \mid bAA$

$B \to b \mid bS \mid aBB.$

Or **(R.G.P.V., June 2007)**

Convert the CFG into PDA –

$S \to aB \mid bA$

$A \to a \mid aS \mid bAA$

$B \to b \mid bS \mid aBB.$

(R.G.P.V., Dec. 2003)

Sol. The required PDA A can be defined as follows –

$$A = (\{q\}, \{a, b\}, \{S, A, B, a, b\}, \delta, q, S, \phi)$$

where, δ is defined by the following rules –

$R_1 : \delta(q, \in, S) = \{(q, aB), (q, bA)\}$

$R_2 : \delta(q, \in, A) = \{(q, a), (q, aS), (q, bAA)\}$

$R_3 : \delta(q, \in, B) = \{(q, b), (q, bS), (q, aBB)\}$

$R_4 : \delta(q, a, a) = \{(q, \in)\}$

$R_5 : \delta(a, b, b) = \{(q, \in)\}$

The language accepted by grammar is

$$L = \{w \in \{a, b\}^* \mid n_a(w) = n_b(w)\}$$

Let us see the sequence of moves in processing the string $baab$

$(q, baab, S) \vdash (q, baab, bA)$ by rule R_1

$\vdash (q, aab, A)$ by rule R_5

$\vdash (q, aab, aS)$ by rule R_2

$\vdash (q, ab, S)$ by rule R_4

$\vdash (q, ab, aB)$ by rule R_1

$\vdash (q, b, B)$ by rule R_4

$\vdash (q, b, b)$ by rule R_3

$\vdash (q, \in, \in)$ by rule R_5

Thus, $baab \in N(A)$

Hence, A is the required PDA.

Prob.24. Convert the following C.F.G into PDA –

$$S \to aS|aA$$
$$A \to bA|b$$

(R.G.P.V., June 2008)

Sol. The required PDA A can be defined as follows –

$$A = (\{q\}, \{a, b\}, \{S, A, a, b\}, \delta, q, s, \phi)$$

where δ is defined by the following rules –

$$R_1 : \delta(q, \in, S) = \{(q, aS), (q, aA)\}$$
$$R_2 : \delta(q, \in, A) = \{(q, bA), (q, b)\}$$
$$R_3 : \delta(q, a, a) = \{q, \in)\}$$
$$R_4 : \delta(q, b, b) = \{(q, \in)\}$$

Let us see the sequence of moves in processing the string $aabb$.

$$\begin{aligned}(q, aabb, S) &\vdash (q, aabb, aS) \text{ by rule } R_1\\ &\vdash (q, abb, S) \text{ by rule } R_3\\ &\vdash (q, abb, aA) \text{ by rule } R_1\\ &\vdash (q, bb, A) \text{ by rule } R_3\\ &\vdash (q, bb, bA) \text{ by rule } R_2\\ &\vdash (q, b, A) \text{ by rule } R_4\\ &\vdash (q, b, b) \text{ by rule } R_2\\ &\vdash (q, \in, \in) \text{ by rule } R_4\end{aligned}$$

Thus $aabb \in N(A)$

Hence, A is the required PDA.

Prob.25. Construct PDA for the following grammar –

$$S \to AB \qquad A \to CD$$
$$B \to b \qquad C \to a$$
$$D \to a$$

(R.G.P.V., Dec. 2011)

Sol. The grammar is not in GNF, first we obtain an equivalent grammar in GNF, which is

$$S \to aDB$$
$$A \to aD$$
$$B \to b$$
$$C \to a$$
$$D \to a$$

Therefore, the PDA equivalent to the given grammar is –

$$M = (\{q\}, \{a, b\}, \{S, A, B, C, D, a, b\}, \delta, q, S, \phi)$$
$$R_1 : (q, \in, S) = \{(q, aDB)\}$$
$$R_2 : (q, \in, A) = \{(q, aD)\}$$
$$R_3 : (q, \in, B) = \{(q, b)\}$$
$$R_4 : (q, \in, C) = \{(q, a)\}$$

$$R_5 : (q, \in, D) = \{(q, a)\}$$
$$R_6 : (q, a, a) = \{(q, \in)\}$$
$$R_7 : (q, b, b) = \{(q, \in)\}$$

Prob.26. Consider the grammar

$$S \to aA$$
$$A \to aABC\,|bB|a$$
$$B \to b$$
$$C \to c$$

Construct PDA corresponding to this grammar. Also provide moves of the PDA and the leftmost derivation for any string in the language defined by the grammar.

(R.G.P.V., Dec. 2015)

Sol. The given grammar is already in Greibach normal form. In addition to rules

$$\delta(q_0, \in, Z) = \{(q_1, SZ)\}$$

and

$$\delta(q_1, \in, Z) = \{(q_f, Z)\}$$

the PDA will also have transition rules

$$\delta(q_1, a, S) = \{(q_1, A)\}$$
$$\delta(q_1, a, A) = \{(q_1, ABC), (q_1, \lambda)\}$$
$$\delta(q_1, b, A) = \{(q_1, B)\}$$
$$\delta(q_1, b, B) = \{(q_1, \in)\}$$
$$\delta(q_1, c, C) = \{(q_1, \in)\}$$

The sequence of moves made by M in processing $aaabc$ is

$$\begin{aligned}(q_0, aaabc, Z) &\vdash (q_1, aaabc, SZ)\\ &\vdash (q_1, aabc, AZ)\\ &\vdash (q_1, abc, ABCZ)\\ &\vdash (q_1, bc, BCZ)\\ &\vdash (q_1, c, CZ)\\ &\vdash (q_1, \in, Z)\\ &\vdash (q_f, \in, Z)\end{aligned}$$

This corresponds to the derivation

$$S \to aA$$
$$S \to aaABC$$
$$S \to aaaBC$$
$$S \to aaabc$$
$$S \to aaabc$$

Prob.27. Construct an NPDA corresponding to the grammar.

$$S \to aABB/aAA$$
$$A \to aBB/a$$
$$B \to bBB/A$$

(R.G.P.V., June 2005)

Sol. The required NPDA A can be defined as follows –

$$A = (\{q\}, \{a, b\}, \{S, A, B, a, b\}, \delta, q, S, \phi)$$

where δ is defined by the following rules –

$$R_1 : \delta(q, \in, S) = \{(q, aABB), (q, aAA)\}$$
$$R_2 : \delta(q, \in, A) = \{(q, aBB), (q, a)\}$$
$$R_3 : \delta(q, \in, B) = \{(q, bBB), (q, A)\}$$
$$R_4 : \delta(q, a, a) = \{(q, \in)\}$$
$$R_5 : \delta(q, b, b) = \{(q, \in)\}$$

Let us see, the sequence of moves in processing the string *aabaaa*.

$(q, aabaaa, S) \vdash (q, aabaaa, aABB)$	by rule R_1
$\vdash (q, abaaa, ABB)$	by rule R_4
$\vdash (q, abaaa, aBB)$	by rule R_2
$\vdash (q, baaa, BB)$	by rule R_4
$\vdash (q, baaa, bBB)$	by rule R_3
$\vdash (q, aaa, BBB)$	by rule R_5
$\vdash (q, aaa, ABB)$	by rule R_3
$\vdash (q, aaa, aBB)$	by rule R_2
$\vdash (q, aa, BB)$	by rule R_4
$\vdash (q, aa, AB)$	by rule R_3
$\vdash (q, aa, aB)$	by rule R_2
$\vdash (q, a, B)$	by rule R_4
$\vdash (q, a, A)$	by rule R_3
$\vdash (q, a, a)$	by rule R_2
$\vdash (q, \in, \in)$	by rule R_4

Thus,

$$aabaaa \in N(A).$$

Hence, A is the required NPDA.

Prob.28. Give a grammar for the language N(M) where –

$$M = (\{q_0, q_1\}, \{0, 1\}, \{x, Z_0\}, \delta, q_0, Z_0, \phi)$$

where δ is given by –

$$\delta(q_0, 0, Z_0) = \{(q_0, xZ_0)\}$$
$$\delta(q_0, 0, x) = \{(q_0, xx)\}$$
$$\delta(q_0, 1, x) = \{(q_1, \in)\}$$
$$\delta(q_1, 1, x) = \{(q_1, \in)\}$$
$$\delta(q_1, \in, x) = \{(q_1, \in)\}$$
$$\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$$

(R.G.P.V., Dec. 2012)

Or

Construct CFG for the following PDA –

$$M = (\{q_0, q_1\}, \{0, 1\}, \{Z_0, X\}, \delta, q_0, Z_0, \phi)$$

$$\delta(q_0, 0, Z_0) = (q_0, XZ_0)$$
$$\delta(q_1, \in, X) = (q_1, \in)$$
$$\delta(q_0, 1, X) = (q_1, \in)$$
$$\delta(q_0, 0, X) = (q_0, XX)$$
$$\delta(q_1, 1, X) = (q_1, \in)$$
$$\delta(q_1, \in, Z_0) = (q_1, \in)$$

(R.G.P.V., June 2007)

Sol. Let $G = (V, \{0, 1\}, P, S)$

where V consists of S, $[q_0, Z_0, q_0]$, $[q_0, Z_0, q_1]$, $[q_0, x, q_0]$ $[q_0, x, q_1]$ $[q_1, Z_0, q_0]$, $[q_1, Z_0, q_1]$, $[q_1, x, q_0]$, $[q_1, x, q_1]$.

The productions are –

P_1 – $S \to [q_0, Z_0, q_0]$

P_2 – $S \to [q_0, Z_0, q_1]$

$\delta(q_0, 0, Z_0) = \{(q_0, xZ_0)\}$ yields

P_3 – $[q_0, Z_0, q_0] \to 0[q_0, x, q_0]\,[q_0, Z_0, q_0]$

P_4 – $[q_0, Z_0, q_0] \to 0[q_0, x, q_1]\,[q_1, Z_0, q_0]$

P_5 – $[q_0, Z_0, q_1] \to 0[q_0, x, q_0]\,[q_0, Z_0, q_1]$

P_6 – $[q_0, Z_0, q_1] \to 0\,[q_0, x, q_1]\,[q_1, Z_0, q_1]$

$\delta(q_0, 0, x) = \{(q_0, xx)\}$ gives

P_7 – $[q_0, x, q_0] \to 0\,[q_0, x, q_0]\,[q_0, x, q_0]$

P_8 – $[q_0, x, q_0] \to 0\,[q_0, x, q_1]\,[q_1, x, q_0]$

P_9 – $[q_0, x, q_1] \to 0\,[q_0, x, q_0]\,[q_0, x, q_1]$

P_{10} – $[q_0, x, q_1] \to 0[q_0, x, q_1]\,[q_1, x, q_1]$

$\delta(q_0, 1, x) = \{(q_1, \in)\}$ gives

P_{11} – $[q_0, x, q_1] \to 1$

$\delta(q_1, 1, x) = \{(q_1, \in)\}$ gives

P_{12} – $[q_1, x, q_1] \to 1$

$\delta(q_1, \in, x) = \{(q_1, \in)\}$ gives

P_{13} – $[q_1, x, q_1] \to \in$

$\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$ gives

P_{14} – $[q_1, Z_0, q_1] \to \in$

$P_1 - P_{14}$ give the productions in P. **Hence, G is the required grammar.**

Prob.29. Obtain CFG for the PDA given as below –

$$A = (\{q_0, q_1\}, \{0, 1\}, \{A, Z\}, \delta, Z, \{q_1\})$$

where δ is as given below –

$$\delta(q_0, 0, Z) = (q_0, AZ)$$
$$\delta(q_0, 1, A) = (q_0, AA)$$
$$\delta(q_0, 0, A) = (q_1, \in)$$

(R.G.P.V., June 2011)

Sol. Let, $G = (V_N, \{0, 1\}, P, S)$

where V_N consists of S, $[q_0, A, q_0]$, $[q_0, A, q_1]$, $[q_0, Z, q_0]$, $[q_0, Z, q_1]$, $[q_1, A, q_0]$, $[q_1, A, q_1]$, $[q_1, Z, q_0]$, $[q_1, Z, q_1]$.

The productions are

$$P_1 : S \to [q_0, Z, q_0]$$
$$P_2 : S \to [q_0, Z, q_1]$$
$$\delta(q_0, 0, Z) \to \{(q_0, A\,Z)\} \text{ yields}$$
$$P_3 : [q_0, Z, q_0] = 0\,[q_0, A, q_0]\,[q_0, Z, q_0]$$
$$P_4 : [q_0, Z, q_0] \to 0\,[q_0, A, q_1]\,[q_1, Z, q_0]$$
$$P_5 : [q_0, Z, q_1] \to 0\,[q_0, A, q_0]\,[q_0, Z, q_1]$$
$$P_6 : [q_0, Z, q_1] \to 0\,[q_0, A, q_1]\,[q_1, Z, q_1]$$
$$\delta(q_0, 1, A) \to \{(q_0, AA)\} \text{ gives}$$
$$P_7 : [q_0, A, q_0] \to 1\,[q_0, A, q_0]\,[q_0, A, q_0]$$
$$P_8 : [q_0, A, q_0] \to 1\,[q_0, A, q_1]\,[q_1, A, q_0]$$
$$P_9 : [q_0, A, q_1] \to 1\,[q_0, A, q_0]\,[q_0, A, q_1]$$
$$P_{10} : [q_0, A, q_0] \to 1\,[q_0, A, q_1]\,[q_1, A, q_1]$$
$$\delta(q_0, 0, A) \to \{(q_1, \in)\} \text{ gives}$$
$$P_{11} : [q_0, A, q_1] \to 0$$
$$P_{12} : [q_0, A, q_0] \to 0$$

$P_1 - P_{12}$ give the productions in P.

Prob.30. Convert the following PDA into CFG –

$$M = (\{q_0, q_1\}, \{0, 1\}, \{Z_0, x\}, \delta, q_0, Z_0, \phi)$$

where δ is defined as follows –

$$\delta(q_0, 1, Z_0) = \{(q_0, xZ_0)\}$$
$$\delta(q_0, 1, x) = \{(q_0, xx)\}$$
$$\delta(q_0, 0, x) = \{(q_1, x)\}$$
$$\delta(q_1, 1, x) = \{(q_1, \in)\}$$
$$\delta(q_1, 0, Z_0) = \{(q_0, Z_0)\}$$
$$\delta(q_0, \in, Z_0) = \{(q_0, \in)\}$$

(R.G.P.V., June 2003, Dec. 2003, June 2004, 2009)

Or

Give CFG equivalent to the following PDA –

$$M = (\{q_0, q_1\}, \{0, 1\}, \{Z_0, X\}, \delta, q_0, Z_0, \phi)$$

Here, 'δ' is given by

$$\delta(q_0, 1, Z_0) = (q_0, XZ_0)$$
$$\delta(q_0, \in, Z_0) = (q_0, \in)$$
$$\delta(q_0, 1, X) = (q_0, XX)$$

$$\delta(q_1, 1, X) = (q_1, \in)$$
$$\delta(q_0, 0, X) = (q_1, X)$$
$$\delta(q_1, 0, Z_0) = (q_0, Z_0)$$

(R.G.P.V., Dec. 2006)

Sol. Let $G = (V, \{0, 1\}, P, S)$

where V consists of S, $[q_0, Z_0, q_0]$, $[q_0, Z_0, q_1]$, $[q_0, x, q_0]$, $[q_0, x, q_1]$, $[q_1, Z_0, q_0]$, $[q_1, Z_0, q_1]$, $[q_1, x, q_0]$, $[q_1, x, q_1]$.

The productions are –

$$P_1 : S \to [q_0, Z_0, q_0]$$
$$P_2 : S \to [q_0, Z_0, q_1]$$
$$\delta(q_0, 1, Z_0) = \{(q_0, xZ_0)\} \text{ yields}$$
$$P_3 : [q_0, Z_0, q_0] \to 1[q_0, x, q_0]\,[q_0, Z_0, q_0]$$
$$P_4 : [q_0, Z_0, q_0] \to 1[q_0, x, q_1]\,[q_1, Z_0, q_0]$$
$$P_5 : [q_0, Z_0, q_1] \to 1\,[q_0, x, q_0]\,[q_0, Z_0, q_1]$$
$$P_6 : [q_0, Z_0, q_1] \to 1[q_0, x, q_1]\,[q_1, Z_0, q_1]$$
$$\delta(q_0, 1, x) = \{(q_0, xx)\} \text{ gives}$$
$$P_7 : [q_0, x, q_0] \to 1[q_0, x, q_0]\,[q_0, x, q_0]$$
$$P_8 : [q_0, x, q_0] \to 1[q_0, x, q_1]\,[q_1, x, q_0]$$
$$P_9 : [q_0, x, q_1] \to 1[q_0, x, q_0]\,[q_0, x, q_1]$$
$$P_{10} : [q_0, x, q_1] \to 1[q_0, x, q_1]\,[q_1, x, q_1]$$
$$\delta(q_0, 0, x) = \{(q_1, x)\} \text{ yields}$$
$$P_{11} : [q_0, x, q_0] \to 0[q_1, x, q_0]$$
$$P_{12} : [q_0, x, q_1] \to 0[q_1, x, q_1]$$
$$\delta(q_1, 1, x) = \{(q_1, \in)\} \text{ gives}$$
$$P_{13} : [q_1, x, q_1] \to 1$$
$$\delta(q_1, 0, Z_0) = \{(q_0, Z_0)\} \text{ gives}$$
$$P_{14} : [q_1, Z_0, q_0] \to 0\,[q_0, Z_0, q_0]$$
$$P_{15} : [q_1, Z_0, q_1] \to 0[q_0, Z_0, q_1]$$
$$\delta(q_0, \in, Z_0) = \{(q_0, \in)\} \text{ gives}$$
$$P_{16} : [q_0, Z_0, q_0] \to \in$$

P_1-P_{16} give the productions in P. ***Hence, G is the required CFG.***

Prob.31. Construct the CFG corresponding to PDA,

$A = (\{q_0, q_1\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \phi)$ and δ is given by

$$\delta(q_0, a, z_0) = (q_0, az_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_1, a)$$
$$\delta(q_1, b, a) = (q_1, a)$$
$$\delta(q_1, a, a) = (q_1, n)$$
$$\delta(q_1, n, z_0) = (q_1, n).$$

(R.G.P.V., June 2017)

Sol. Let G = $(V, \{a, b\}, P, S)$, where V consist of

$[q_0, Z_0, q_0]$, $[q_1, Z_0, q_0]$, $[q_0, a, q_0]$, $[q_1, a, q_0]$,
$[q_0, Z_0, q_1]$, $[q_1, Z_0, q_1]$, $[q_0, a, q_1]$, $[q_1, a, q_1]$

The productions in P are constructed as follows –

The S-productions are

$P_1 : S \to [q_0, Z_0, q_0]$ $\quad P_2 : S \to [q_0, Z_0, q_1]$

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$ gives

$P_3 : [q_0, Z_0, q_0] \to a[q_0, a, q_0]\,[q_0, Z_0, q_0]$
$P_4 : [q_0, Z_0, q_0] \to a[q_0, a, q_1]\,[q_1, Z_0, q_0]$
$P_5 : [q_0, Z_0, q_1] \to a[q_0, a, q_0]\,[q_0, Z_0, q_1]$
$P_6 : [q_0, Z_0, q_1] \to a[q_0, a, q_1]\,[q_1, Z_0, q_1]$

$\delta(q_0, a, a) = \{(q_0, aa)\}$ gives

$P_7 : [q_0, a, q_0] \to a[q_0, a, q_0]\,[q_0, a, q_0]$
$P_8 : [q_0, a, q_0] \to a[q_0, a, q_1]\,[q_1, a, q_0]$
$P_9 : [q_0, a, q_1] \to a[q_0, a, q_0]\,[q_0, a, q_1]$
$P_{10} : [q_0, a, q_1] \to a[q_0, a, q_1]\,[q_1, a, q_1]$

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

$P_{11} : [q_0, a, q_0] \to b[q_1, a, q_0]$
$P_{12} : [q_0, a, q_1] \to b[q_1, a, q_1]$

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

$P_{13} : [q_1, a, q_0] \to b[q_1, a, q_0]$
$P_{14} : [q_1, a, q_1] \to b[q_1, a, q_1]$

$\delta(q_1, a, a) = \{(q_1, n)\}$ gives

$P_{15} : [q_1, a, q_1] \to a$

$\delta(q_1, n, Z_0) = \{(q_1, n)\}$ gives

$P_{16} : [q_1, Z_0, q_1] \to n$

UNIT 5

TURING MACHINE

TECHNIQUES FOR CONSTRUCTION, UNIVERSAL TURING MACHINE MULTITAPE, MULTIHEAD AND MULTIDIMENSIONAL TURING MACHINE

Q.1. Give definition of a turing machine.

Ans. More clearly, a turing machine M is defined as –

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

where,

Q is the set of internal states

Σ is the set of input alphabet

Γ is a finite set of symbol known as ***tape alphabet***

δ is the transition function

$B \in \Gamma$ is a special symbol called ***blank***

$q_0 \in Q$ is the initial state

$F \subseteq Q$ is the set of final states.

In the definition of a turing machine, we assume that $T \subseteq \Gamma - \{B\}$, i.e., the input alphabet is a subset of the tape alphabet, not including the blank. Blanks are ruled out as input.

The transition function δ is defined as follows –

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

In general, δ is a partial function on $Q \times \Gamma$. Its interpretation gives the principle by which a turing machine operate. The arguments of δ are the current state of the control unit and the current tape symbol being read. The output is a new state of the control unit, a new tape symbol, which replace the previous one, and a move symbol, L or R. The move symbol indicates whether the read-write head moves left or right one cell after the new symbol has been written on the tape.

Q.2. Explain turing machine as a mathematical model of computation. ***(R.G.P.V., June 2003)***

Or

Explain the concept of turing machine model. (R.G.P.V., June 2015)

Ans. Turing machine (TM) is a simple mathematical model of a general purpose computer. We can say that, the turing machine model is a computing power of the computer, i.e., turing machine is capable of performing any calculation which can be performed by any computing machine.

Turing machine can be thought of as a finite-state automaton connected to a R/W (read/write) head. It has a tape for temporary storage, which is divided into number of cells. Each cell can store only one symbol. The input to and the output from the finite state automaton are effected by the read/write head which can examine one cell at a time. The machine examines the present symbol under the read/write head on the tape, in one move and the present state of an automaton to determine –

(i) A new symbol to be written on the tape in the cell under the read/write head

(ii) A motion of the read/write head along the tape, which may be either one cell left (L) or one cell right (R)

(iii) The next state of the automaton

(iv) Whether to halt or not.

Fig. 5.1 shows an intuitive visualization of a turing machine.

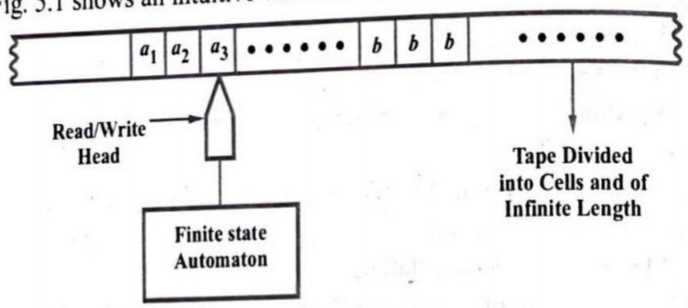


Fig. 5.1 Turing Machine Model

Q.3. Explain TM and its parameter. ***(R.G.P.V., June 2011)***

Or

Write short note on turing machine. ***(R.G.P.V., June 2017)***

Ans. Refer to Q.2 and Q.1.

Q.4. Give similarities and differences of turing machines and finite state automata. ***(R.G.P.V., Dec. 2008)***

Or

Differentiate the purpose of the study of turing machine with finite automata/pushdown automata. ***(R.G.P.V., Dec. 2015)***

Ans. **Differences –** (i) A finite state automaton is described by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, while a turing machine has a 7-tuple, viz. $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$.

(ii) A finite automaton has only read head, while a turing machine has R/W head.

Similarities – (i) The turing machine can be thought of as a finite state automaton connected to a R/W head.

(ii) Both have a finite control, an input tape that is divided into cells, and a tape head that scans one cell of the tape at a time.

Q.5. Turing machine models the computing capacity of a general purpose computer. Justify with complete example. ***(R.G.P.V., June 2010)***

Ans. Fig. 5.2 shows the situation before and after the move caused by the transition

$\delta(q_0, a) = (q_1, b, R)$.

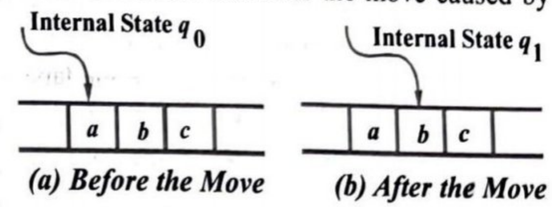


Fig. 5.2

We can consider a turing machine as a rather simple computer. It has a processing unit, which has a finite memory, and in its tape, it has a secondary storage of unlimited capacity. The instructions that such a computer (i.e., turing machine) can carry out are very limited, it can sense a symbol on its tape and use the result to take decision what to do next. The only action the machine can perform are to rewrite the current symbol, to change the state of the control, and to move the read-write head. This small instruction set may look inadequate for doing complicated things, but this is not so. Turing machines are much powerful in principle. The transition function δ define how this computer acts, and we often call it the "program" of the machine.

As always, the automaton starts in the given initial state having some information on the tape. It then goes through a sequence of steps controlled by the transition function (δ). During this process, the contents of any cell on the tape may be examined and changed as many times as it need. Eventually, the whole process may terminate, which we achieve in a turing machine by putting it into a ***halt state***. A Turing machine is said to be in halt state, whenever it reaches a configuration for which δ is not defined, this is possible because δ is a partial function. In fact, we will assume that no transitions are defined for any final state, so the turing machine will halt whenever it comes in final state.

Let us take an example –

Consider the turing machine defined by –

$$Q = \{q_0, q_1\}$$
$$\Sigma = \{a,b\}$$
$$\Gamma = \{a, b, B\}$$
$$F = \{q_1\}$$

and

$$\delta(q_0, a) = (q_0, b, R)$$
$$\delta(q_0, b) = (q_0, b, R)$$
$$\delta(q_0, B) = (q_1, B, L)$$

If this turing machine is started in state q_0 with the input symbol a under the read-write head, the applicable transition rule is $\delta(q_0,a) = (q_0, b, R)$. So, the read-write head will replace the a with b, then move right on the tape. The machine will still remain in state q_0. Any subsequent a will also be replaced with b, but b's will not be modified. When the machine finds the first blank, it will move left one cell, then half in final state q_1. Fig. 5.3 shows a sequence of moves and several stages of the process for a simple initial configuration.

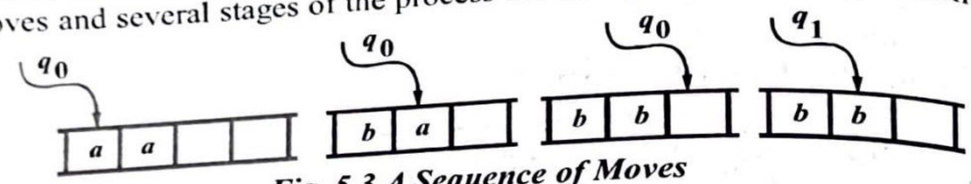


Fig. 5.3 A Sequence of Moves

Q.6. Demonstrate the working of your TM with an example. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.2 and Q.5.

Q.7. Explain representation of turing machines with the help of examples.

Ans. We can describe a turing machine using – (i) Instantaneous description using move-relations, (ii) Transition table, and (iii) Transition diagram.

(i) Representation by Instantaneous Description – 'Snapshots' of a turing machine in action can be used to describe a turing machine. These give "instantaneous description" of a turing machine. An ID of a Turing machine is defined in terms of the entire input string and the current state.

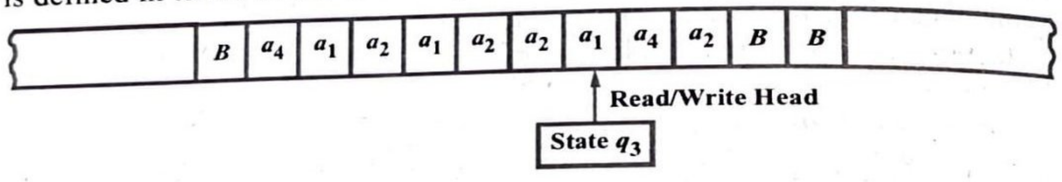


Fig. 5.4 Snapshot of Turing Machine

An ID of a turing machine M is a string $\alpha\beta\gamma$, where β is the present state of M, the entire input string is split as $\alpha\gamma$, the first symbol of γ is the current symbol, say a, under R/W head and γ has all the subsequent symbols of the input string, and the string α is the substring of the input string formed by all the symbols to the left of a.

A snapshot of turing machine is shown in fig. 5.4 and the corresponding ID is given in fig. 5.5.

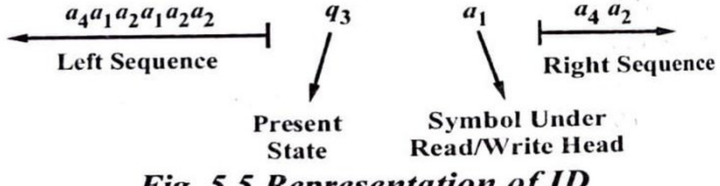


Fig. 5.5 Representation of ID

Moves in a TM – $\delta(q, x)$ induces a change in ID of the turing machine. This change in ID is called a move.

For example – Suppose $\delta(q, x_i) = (p, y, L)$. The input string to be processed is $x_1\ x_2 \ldots x_n$, and the present symbol under R/W head is x_i. So the ID before processing x_i is

$$x_1x_2 \ldots x_{i-1}\, qx_i \ldots x_n$$

After processing of x_i, the resulting ID is

$$x_1x_2 \ldots x_{i-2}\, px_{i-1}\, yx_{i+1} \ldots x_n$$

This change in ID is represented by

$$x_1\ x_2 \ldots x_{i-1}\, qx_i \ldots x_n \vdash x_1x_2 \ldots x_{i-2}\, px_{i-1}\, yx_{i+1} \ldots x_n$$

If $\delta(q, x_i) = (p, y, R)$, then the change of ID is represented by

$$x_1x_2 \ldots x_{i-1}qx_i \ldots x_n \vdash x_1x_2 \ldots x_{i-1}\, ypx_{i+1} \ldots x_n.$$

We can denote an ID by I_j for some j. $I_j \vdash I_k$ defines a relation among IDs. So the symbol $\vdash^*$ denotes the reflexive-transitive closure of the relation $\vdash$. If $I_1 \vdash^* I_n$, then we can split this as $I_1 \vdash I_2 \vdash \ldots \vdash I_n$ for some IDs, $I_2, \ldots, I_{n-1}$.

It is noted that the description of moves, by IDs is very much useful to represent the processing of input strings.

(ii) Representation by Transition Table – The definition of δ can be given in the form of a table called the transition table. If $\delta(q, a) = (\gamma, \alpha, \beta)$, we write (γ, α, β) under a-column and q-row. So if we get (γ, α, β) in the table, it means that α is written in the current cell, β gives movement of the head (L or R) and γ denotes the new state into which the turing machine enters.

For example, consider a turing machine with five states $q_1, \ldots, q_5$, where q_1 is the initial state and q_5 is the final state. The tape symbols are 0, 1, and B. The transition table given in table 5.1 describes δ.

Table 5.1 Transition Table of a Turing Machine

Present State	Tape Symbol		
	B	0	1
→ q_1	$(q_2, 1, L)$	$(q_1, 0, R)$	
q_2	(q_3, B, R)	$(q_2, 0, L)$	$(q_2, 1, L)$
q_3		(q_4, B, R)	(q_5, B, R)
q_4	$(q_5, 0, R)$	$(q_4, 0, R)$	$(q_4, 1, R)$
q_5	$(q_2, 0, L)$		

(iii) Representation by Transition Diagram – Turing machine can be represented by transition system. The states are represented by vertices. Directed edges are used to represent transition of states. The labels are triples of the form (α, β, γ), where $\alpha, \beta \in \Gamma$ and $\gamma \in \{L, R\}$. When there is a directed edge from q_i to q_j with label (α, β, γ), it means that

$$\delta(q_i, \alpha) = (q_j, \beta, \gamma)$$

During the processing of an input string, suppose the turing machine enters q_i and R/W head scans the (present) symbol α. As a result, the symbol β is written in the cell under R/W head. The R/W head moves to the left or to the right, depending on γ, and the new state is q_j.

Every edge in the transition system can be represented by a 5-tuple $(q_i, \alpha, \beta, \gamma, q_j)$. So each turing machine can be described by the sequence of 5-tuples representing all the directed edges. The initial state is indicated by → and any final state is marked with circle.

For example, a turing machine represented by the transition system is given in fig. 5.6.

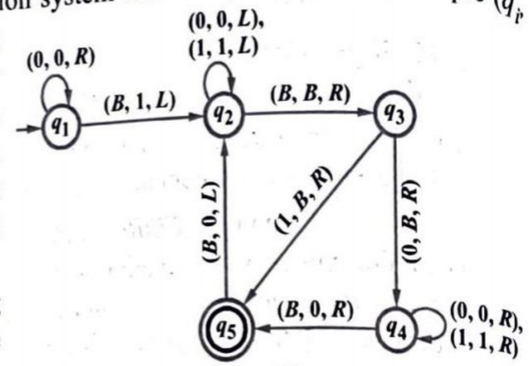


Fig. 5.6 Transition Diagram of Turing Machine of Table 5.1

Q.8. Briefly elaborate the representation of turing machine by Instantaneous Descriptors (IDs). **(R.G.P.V., Nov. 2018)**

Or

Explain ID of a turing machine. **(R.G.P.V., June 2016)**

Ans. Refer to Q.7.

Q.9. What type of language can a turing machine accept ?

Ans. Turing machine can be viewed as accepters in the following sense –

A string w is written on the tape, with blanks filling out the unused portion. The machine is started in the initial state q_0 with the read-write head positioned on the leftmost symbol of w. If, after a sequence of moves, the turing machine enters a final state and halts, then w is considered to be accepted.

Formally, we can say that –

If $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is a turing machine. Then the languages accepted by M is

$$L(M) = \{w \in \Sigma^+ : q_0 w \vdash^* x_1 q_f x_2 \text{ for some } q_f \in F, x_1, x_2 \in \Gamma^*\}.$$

The above said definition indicates that the input w is written on the tape with blanks on either side. The reason for excluding blanks from the input now becomes clear, i.e., it assures us that all the input restricted to a well-defined region of the tape, bracketed by blanks on the right and left side. Without this convention, the machine could not limit the region in which it must look for the input, no matter how many blanks it saw, it could never be sure that there was not some non-blank input somewhere else on the tape.

The definition tells us what must happen when $w \in L(M)$. It says nothing about the outcome for any other input. When w is not in $L(M)$, one of two things can happen, i.e., the machine can halt in a non-final state or it can enter an infinite loop and never halt. Any string for which M does not halt is by definition not in $L(M)$.

Q.10. Write down the various techniques for the turing machine construction.

Ans. For constructing a turing machine some higher-level conceptual tools are needed. Some of them are as follows –

(i) Storage in the Finite Control – The finite control can be used to hold a finite amount of information. For this purpose, the state is written as a pair of elements, one exercising control and the other storing a symbol. It should be noted that this arrangement is for conceptual purposes only, and for this no modifications in the definition of the turing machine has been made.

(ii) Multiple Tracks – We can imagine that the tape of turing machine is divided into m tracks, for any finite m. With $m = 3$, this arrangement is shown in fig 5.7. The symbols on the tape are considered m-tuples, one component for each track.

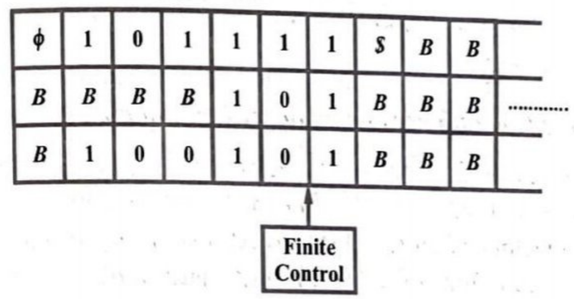


Fig. 5.7 A Three-track Turing Machine

(iii) Checking Off Symbols – To show how a turing machine recognizes languages defined by repeated strings, checking off is a useful trick, such as –

$$\{ww \mid w \text{ in } \Sigma^*\},\ \{wcy \mid w \text{ and } y \text{ in } \Sigma^*, w \neq y\} \text{ or}$$
$$\{ww^R \mid w \text{ in } \Sigma^*\}$$

It is also useful when lengths of substrings must be compared, such as

$$\{a^i b^i \mid i \geq 1\} \text{ or } \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$$

An extra track is introduced on the tape that holds a blank or √. The √ appears when the symbol below it has been considered by the turing machine in one of its comparisons.

(iv) Shifting Over – A turing machine can make space on its tape by shifting all non blank symbols a finite number of cells to the right. For this

purpose, the tape head makes an excursion to the right repeatedly, storing the symbols read in its finite control and replacing them with symbols read from cells to the left. Then the turing machine can come to the vacated cells and print symbols of its choosing. It can push blocks of symbols left in a similar manner, if the space is available.

(v) ***Subroutines*** – In programs, if we use subroutines to define elementary processes, a modular or top-down design is facilitated. A turing machine can simulate any type of subroutine found in programming languages, including recursive procedures and any of the known parameter-passing mechanisms. Here only the use of parameterless, nonrecursive subroutines is described, but even these are quite powerful tools.

Q.11. What is turing-computable function ? Define recursive function. ***(R.G.P.V., Dec. 2015)***

Ans. A function f with domain D is said to be Turing-computable or just computable if there exists some Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ such that

$$q_0 w \overset{*}{\underset{M}{\vdash}} q_f f(w), q_f \in F,$$

for all $w \in D$,

The class of recursive functions include the class of primitive recursive functions.

Let $g(a_1, a_2, \ldots\ldots, a_n)$ be a total function over N. g is regular function if there exists some natural number b_0 such that $g(a_1, a_2, \ldots\ldots, a_n, b_0) = 0$ for all values $a_1, a_2, \ldots\ldots, a_n$ in N.

For instance, $g(a, b) = \min(a, b)$ is a regular function since $g(a, 0) = 0$ for all a in N. But $f(a, b) = |a - b|$ is not regular since $f(a, b) = 0$ only when $a = b$, and so we cannot find a fixed b such that $f(a, b) = 0$ for all a in N.

A function $f(a_1, a_2, \ldots\ldots, a_n)$ over N is defined from a total function $g(a_1, a_2, \ldots\ldots, a_n, b)$ by minimization if –

(i) $f(a_1, a_2, \ldots\ldots, a_n)$ is the least value of all b's such that $g(a_1, a_2, \ldots\ldots, a_n, b) = 0$ if it exists. The least value is denoted by $\mu b(g(a_1, a_2, \ldots\ldots, a_n, b) = 0)$,

(ii) $f(a_1, a_2, \ldots\ldots, a_n)$ is undefined if there is no b such that $g(a_1, a_2, \ldots\ldots, a_n, b) = 0$.

It is noted that, f is partial. But, if g is regular then f is total.

A function is recursive if it can be obtained from the initial function by a finite number of applications of composition, recursion and minimization over regular functions.

A function is partial recursive if it can be obtained from the initial functions by a finite number of applications of composition, recursion and minimization.

Q.12. Write brief note on design of TM. ***(R.G.P.V., Dec. 2009)***

Ans. The basic guidelines for designing a turing machine are as follows –

(i) The fundamental objective in scanning a symbol by R/W head is to 'know' what to do in the future. The machine must remember the past symbols scanned. The turing machine can remember this by going to the next unique state.

(ii) The number of states must be minimized. This can be achieved by changing the states only when there is a change in the written symbol or when there is a change in the movement of R/W head.

Q.13. What are the types of turing machine ?

Ans. The various types of turing machines are as follows –

(i) Multitape turing machine
(ii) Multidimensional turing machine
(iii) Off-line turing machine
(iv) Multiple tracks turing machine
(v) Multiple heads turing machine
(vi) Universal turing machine
(vii) Two-way infinite tape turing machine.

Q.14. What do you mean by turing machine ? Explain different types of turing machine. ***(R.G.P.V., Dec. 2017)***

Ans. Refer to Q.1, Q.2 and Q.13.

Q.15. What do you mean by turing machine ? Explain multiple tapes turing machine. ***(R.G.P.V., Dec. 2013)***

Ans. **Turing Machine** – Refer to Q.1 and Q.2.

Multitape Turing Machine – A multitape turing machine consists of a finite control with several tape with its own controlled R/W heads as shown in fig. 5.8.

Each tape is infinite in both direction. On a single move depending upon the state of the finite control and the symbol scanned by each of the tape heads, the machine can –

(i) Change state

(ii) Print a new symbol on each of the cells scanned by its tape heads

(iii) 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.

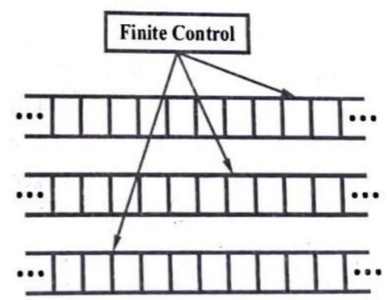


Fig. 5.8 Multitape Turing Machine

The formal definition of a multitape TM goes beyond the definition of a standard TM, since it requires a modified transition function. Typically, we define an n-tape machine by $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where Q, Σ, Γ, q_0, B and F are as in the definition of a turing machine and the transition function δ is defined as –

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

δ specifies what happens on all the tapes. For example, if $n = 2$, with a current configuration shown in fig. 5.9 (a), then

$$\delta(q_0, a, e) = (q_1, x, y, L, R)$$

is interpreted as follows. The transition rule can be applied only if the machine is in state q_0 and the first R/W head encouters an a and the second an e. The symbol on the first tape will then be replaced with an x and its R/W head will move towards left. At the same time, the symbol on the second tape is rewritten as y and the R/W head moves right. The control unit then changes its state to q_1 and the machine goes into the new configuration as shown in fig. 5.9 (b).

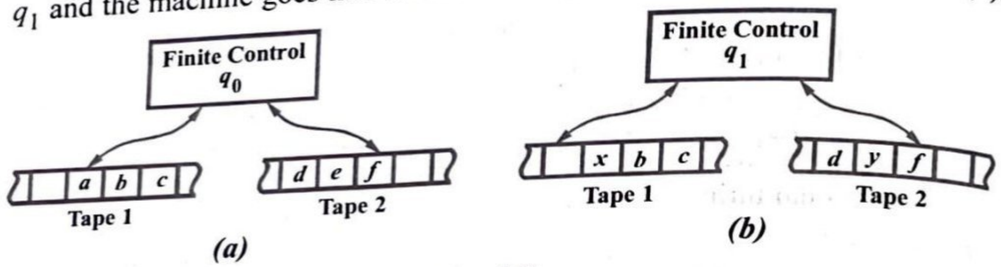


Fig. 5.9

Q.16. Show that if a language L is accepted by a multitape turing machine, then it is also accepted by a single-tape turing machine.

Ans. Let L be accepted by M_1, a turing machine with k-tapes. We can construct M_2, a one-tape turing machine with $2k$ tracks, two tracks for each M_1's tape. One track records the contents of the corresponding tape of M_1 and other is blank, except for a marker in the cell that holds the symbol scanned by the corresponding head of M_1, as shown in fig. 5.10. The finite control of M_2 stores the state of M_1, along with a count of the number of head markers to the right of M_2's tape head.

Each move of M_1 is simulated by a sweep from left to right and then from right to left by the tape head of M_2. Initially, M_2's head is at the leftmost cell containing a head marker. To simulate a move of M_1, M_2 sweeps right, visiting each of the cells with head marker and recording the symbol scanned by each head of M_1.

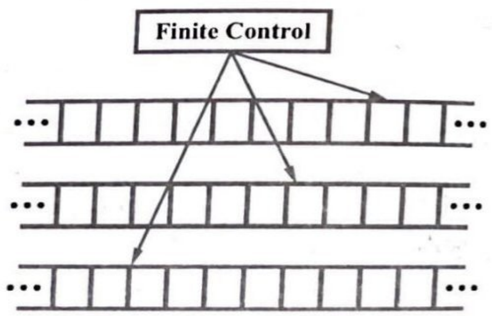


Fig. 5.10 Multitape Turing Machine

When M_2 crosses a head marker, it must update the count of head marker to its right. When no more head marker are to the right, M_2 has seen the symbols scanned by each of M_1's heads, so M_2 has enough information to determine the move of M_1. Now, M_2 makes a pass left, until it reaches the leftmost head marker. The count of markers to the right enables M_2 to tell when it has gone far enough. As M_2 passes each head marker on the leftward pass, it updates the tape symbol of M_1 "scanned" by the head marker, and it moves the head marker one symbol left or right to simulate the move of M_1. Finally, M_2 change the state of M_1 recorded in M_2's control to complete the simulation of one move of M_1. If the new state of M_1 is accepting, then M_2 accepts.

Head 1		X				
Tape 1	A_1	A_2	...		...	A_m
Head 2				X		
Tape 2	B_1	B_2	...		...	B_m
Head 3	X					
Tape 3	C_1	C_2	...		...	C_m

Fig. 5.11 Simulation of Three Tapes by One

Q.17. Write a brief note on multidimensional turing machine.
(R.G.P.V., Dec. 2002)

Ans. Multidimensional turing machine is one in which the tape can be viewed as extending infinitely in more than one dimension. A diagram of a two-dimensional turing machine is shown in fig. 5.12 (a).

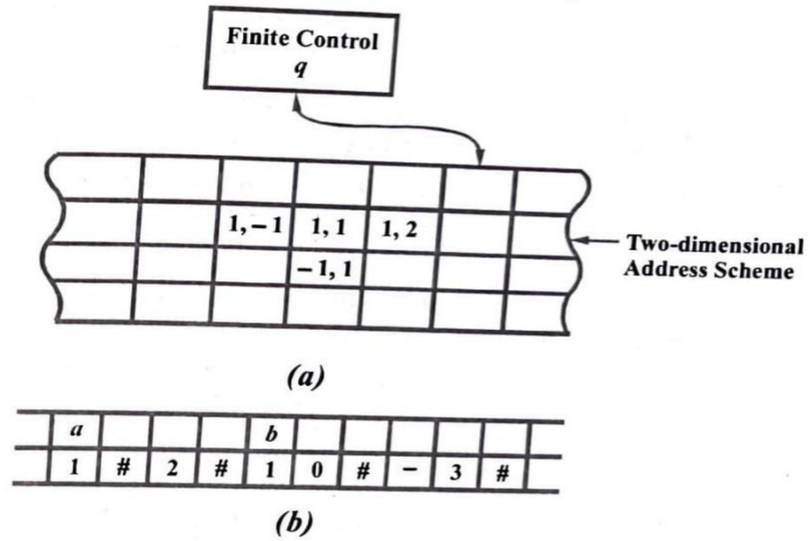


Fig. 5.12

The formal definition of a two-dimensional turing machine involves a transition function δ of the following form –

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, U, D\},$$

where U and D specify movement of the R/W head up and down, respectively

To simulate this machine on a standard turing machine, we can use the two-track model depicted in fig. 5.12 (b). First, we associate an ordering or address with the cells of the two-dimensional tape. This can be done in a number of ways, for example, in two-dimensional fashion indicated in fig. 5.12 (a). The two-track tape of the simulating machine will use one track to store cell contents and the other one to keep the associated address. In the scheme of fig. 5.12 (a), the configuration in which cell (1, 2) contains a and cell (10, – 3) contains b is shown in fig. 5.12 (b). Note one complication – the cell address can involve arbitrarily large integers, so the address track cannot use a fixed-size field to store address. Instead, we must use a variable field-size arrangement, using some special symbols to delimit the fields, as shown in fig. 5.12.

Let us suppose that, at the start of the simulation of each move, the R/W head of the two-dimensional machine M and the R/W head of the simulating machine M' are always on the corresponding cells. To simulate a move, the simulating machine, i.e., M', first compute the address of the cell to which M is to move. Using the two-dimensional address scheme, it is a simple computation. Once the address is computed, simulating machine M' finds the cell with this address on track 2 and then change the cell contents to account for the move of M. Again, given M, there is a straightforward construction for M'.

Q.18. Show that if L is accepted by a two-dimensional turing machine M_2, then L is accepted by a one-dimensional turing machine M_1.

Ans. M_1 represents the tape of M_2 as shown in fig. 5.13 (b). M_1 will also have a second tape used for purposes and the tapes are two-way infinite.

Suppose that M_2 makes a move in which the head does not leave the rectangle already represented by M_1's tape. If the move is horizontal, M_1 simply moves its head marker one cell left or right after printing a new symbol and changing the state of M_2 recorded in M_1's control. If the move is vertical, T_1 uses its second tape to count the number of cells between the tape head position and the * to its left. Then M_1 moves to the * to the right, if the move is down, or the * to the left if the move is up, and puts the tape head marker at the corresponding position in the new block (between *'s region) by using the count on the second tape.

A	A	A	a_1	A	A	A
A	A	a_2	a_3	a_4	a_5	A
a_6	a_7	a_8	a_9	A	a_{10}	A
A	a_{11}	a_{12}	a_{13}	A	a_{14}	a_{15}
A	A	a_{16}	a_{17}	A	A	A

(a) Two-dimension Tape

$$** AAA\, a_1\, AAA{*}\, AA\, a_2 a_3 a_4 a_5\, A{*}\, a_6 a_7 a_8 a_9 A a_{10} A{*}$$
$$A\, a_{11} a_{12} a_{13}\, A a_{14} a_{15}{*}\, AA\, a_{16} a_{17}\, AAA{**}$$

(b) One-dimensional Simulation

Fig. 5.13 Simulation of Two-dimensions by One

Now consider the situation when M_2's head moves off the rectangle represented by M_1. If the move is vertical, add a new block of blank to the left or right, using the second tape to count the current length of blocks. If the move is horizontal, M_1 uses the "shifting over" technique to add a blank at the left or right end of each block, as appropriate. It is noted that double *'s mark the ends of the region used to hold blocks, so M_1 can tell when it has augmented all blocks. After creating room to make the move, M_1 simulates the move of M_2.

Q.19. Formally define an off-line turing machine. Also give the arguments that any language accepted by an off-line turing machine is also accepted by some standard machine.

Or

Write a short note on off-line turing machine. (R.G.P.V., June 2003, 2004)

Ans. An off-line turing machine is a machine in which each move is governed by the internal state, what is currently read from the input file and what is seen by the R/W head. A schematic representation of an off-line turing machine is shown in fig. 5.14.

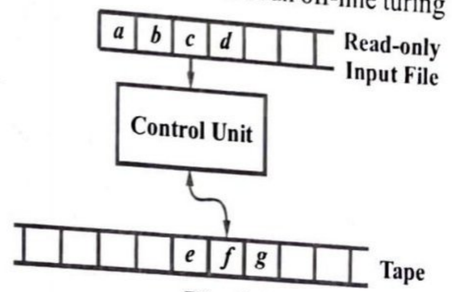


Fig. 5.14

The behaviour of any standard turing machine can be simulated by some off-line model. All that needs to be done by the simulating machine is to copy the input from the input file to the tape. Then it can proceed in the same way as the standard machine.

The simulation of an off-line machine M by a standard machine M' requires a lengthy description. A standard machine can simulate the computations of an off-line machine by using the four-track arrangement shown in fig. 5.15. In which, the tape contents represent the specific configuration of fig. 5.14. Each of the four tracks of M' plays a specific role in the simulation. The first track has the input, the second marks the position where the input is read, the third represents the tape of M, and the fourth shows the position of M's R/W head.

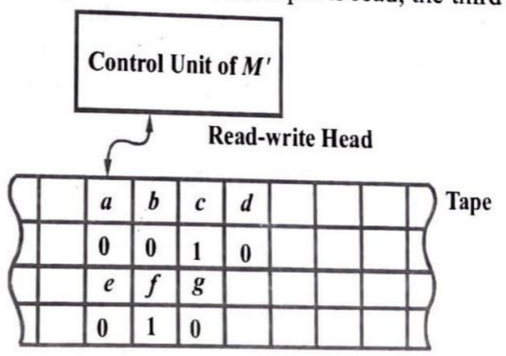


Fig. 5.15

The simulation of each move of M requires a number of moves of M'. Beginning from some standard position, say the left end, and with the relevant information marked by special end markers, M' seeks track 2 to locate the

position where the input file of M is read. The symbol found in the corresponding cell on track 1 is remembered by putting the control unit of M' into a state chosen for this purpose. Next, track 4 is searched for the position of the R/W head of M. With the remembered input and the symbol on track 3, we now know what M is to do. This information is again remembered by M' with an appropriate internal state. Next, all four tracks of simulating machine's tape are modified to reflect the move of M,. Finally, the R/W head of M' returns to the standard position for the simulation of the next move.

Q.20. Write short note on multiple tracks turing machine ?

Ans. We know that, a turing machine M is defined by

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

where, Q is the set of internal states
Σ is the set of input alphabet
Γ is a finite set of symbols called the tape alphabet
δ is the transition function
$B \in \Gamma$ is a special symbol called the blank
$q_0 \in Q$ is the initial state
$F \subseteq Q$ is the set of final states.

By the above definition, it is implicit that each tape symbol can be a composite of characters rather than just a single one. This can be made more explicit by fig. 5.16, in which the tape symbols are triplets from some simpler alphabet.

In fig. 5.16, we have divided each cell of the tape into three parts, which are known as tracks, and each containing one member of the triplet. Based on this visualization of fig. 5.16, an automaton is sometimes called a turing machine with multiple tracks, but such a view in not requirement for doing modifi-cation in the definition of the turing machine, since all we need to do is make Γ an alphabet in which each symbol is composed of several parts.

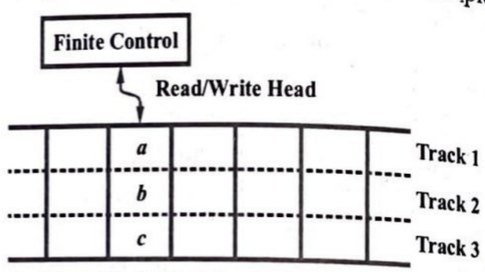


Fig. 5.16 Multiple Tracks in Turing Machine

Q.21. Explain multitape and multihead turing machine.
(R.G.P.V., Dec. 2011)

Ans. **Multitape Turing Machine** – Refer to Q.15.

Multihead Turing Machine – If we allow a turing machine to have one tape, but several heads on it, then we say in one step, the all heads sense the scanned symbols and move or write independently. What happen when two

heads, that happen to be scanning the same tape square, attempt to write different symbols, for this same convention must be adopted perhaps the head with the lower number wins out. Also, let us assume that the head cannot sense each other's presence in the same tape square, except perhaps indirectly, through unsuccessful writes.

It is not hard to see that a simulation like the one we used for n-tape machines can be carried out for turing machines with several heads on a tape. The basic idea is to divide the tape into tracks, all but one of which are used solely to record the head positions. To simulate one computational step by the multiple-head machine, the tape must be scanned twice, once to find the symbols at the head position and next to change those symbols or move the head as appropriate. The number of steps needed is quadratic.

The use of multiple heads, like multiple tapes, can sometimes drastically simplify the construction of a turing machine. A two-head version of the copying machine could function in much more natural than the one-head version or even the two-tape version machine.

Q.22. Explain in detail notes on universal turing machines with example.
(R.G.P.V., May 2018)

Or

Write a note on universal turing machine. ***(R.G.P.V., Dec. 2009)***

Or

How UTM overcomes the limitation of turing machine ? Also define UTM.
(R.G.P.V., Dec. 2015)

Or

Briefly describe the universal turing machine. ***(R.G.P.V., Nov. 2018)***

Ans. Consider the following argument against turing thesis – A turing machine is a special purpose computer. Once δ is defined the machine is restricted to carrying out one particular type of computation. Digital computers, on the other hand, are general purpose machines that can be programmed to do different jobs at different times. Consequently, turing machines cannot be considered equivalent to general purpose digital computers. This objection can be overcome by designing a reprogrammable turing machine called a universal turing machine.

A ***universal turing machine*** M_u is an automaton that, given as input the description of any turing machine M and a string u, can simulate the computation of M on u. To construct such an M_u we first select a standard way of describing turing machines. We may, without loss of generality, assume that,

$$Q = \{q_1, q_2, \ldots\ldots\ldots, q_n\}$$

with q_1 the initial state, q_2 the only final state, and

$$\Gamma = \{a_1, a_2, \ldots\ldots\ldots, a_m\}$$

where a_1 represents the blank. Then we pick an encoding in which q_1 is represented by 1, q_2 is represented by 11, and so on. Similarly, a_1 is encoded as 1, a_2 as 11, and so on. The symbol 0 will be used as a separator between the 1's. With the initial and final state and the blank defined by this convention, any turing machine can be explained completely with δ only. The transition function is encoded according to this scheme, with the arguments and result in some prescribed sequence. For example, $\delta(q_1, a_2) = (q_2, a_3, L)$ might appear as

... 10110110111010...

It means that any turing machine has a finite encoding as a string on $\{0,1\}^+$, and that, given any encoding of M, we can decode it uniquely. Some strings will not represent any turing machine (for example, the string 00011), but we can easily spot these, so they are of no concern.

A universal turing machine M_u then has an input alphabet that includes $\{0,1\}$ and the structure of a multitape machine, as shown in fig. 5.17.

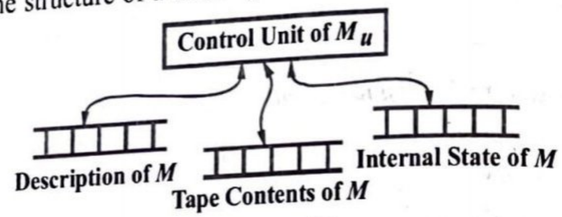


Fig. 5.17

For any input M and u, tape 1 will have an encoded definition of M, and tape 2 will contain the tape contents of M, and tape 3 the internal state of M. M_u sees first at the contents of tapes 2 and 3 to find the configuration of M. Then it consults tape 1 to see what M would do in this configuration of M. Finally, the tapes 2 and 3 will be modified to reflect the result of the move.

Q.23. What are the features of universal turing machine ?
(R.G.P.V., Dec. 2014)

Ans. The features of universal turing machine are as follows –

(i) Universal turing machine can simulate any other turing machine.

(ii) Universal turing machine has an ability to manipulate an unbounded amount of data in finite amount of time.

Q.24. Explain multitape and universal turing machine.
(R.G.P.V., June 2016)

Ans. Refer to Q.15 and Q.22.

Q.25. Write brief note on the two-way infinite tape TM.
(R.G.P.V., Dec. 2009)

Ans. A turing machine with a two-way infinite tape is denoted by $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, as in the original model. As its name implies, the tape is infinite to the left as well as to the right. We denote an ID of such a device as

for the one-way infinite TM. We imagine, however, that there is an infinity of blank cells both to the left and right of the current nonblank portion of the tape.

The relation $\vdash_M$, which relates two ID's if the ID on the right is obtained from the one on the left by a single move, is defined as for the original model with the exception that if $\delta(q, X) = (p, Y, L)$, then $qX\alpha \vdash_M pBY\alpha$ (in the original model, no move could be made), and if $\delta(q, X) = (p, B, R)$, then $qX\alpha \vdash p\alpha$ (in the original, the B would appear to the left of p).

The initial ID is q_0w. While there was a left end to the tape in the original model, there is no left end of the tape for the turing machine to "fall off", so it can proceed left as for as it wishes. The relation $\vdash_M^*$, as usual, relates two ID's if the one on the right can be obtained from the one on the left by some number of moves.

Q.26. Write short note on various types of turing machine.
(R.G.P.V., Dec. 2004, June 2006)

Or

Write short note on models of turing machine. ***(R.G.P.V., Dec. 2006)***

Or

Explain the types of turing machine. ***(R.G.P.V., Dec. 2016)***

Ans. (i) Multitape Turing Machine – Refer to Q.15.

(ii) Multidimensional Turing Machine – Refer to Q.17.

(iii) Off-line Turing Machine – Refer to Q.19.

(iv) Multiple Tracks Turing Machine – Refer to Q.20.

(v) Multiple Heads Turing Machine – Refer to Q.21.

(vi) Universal Turing Machine – Refer to Q.22.

(vii) Two-way Infinite Tape Turing Machine – Refer to Q.25.

Q.27. Give definition of a turing machine. What type of language can it accept ? Types of turing machine. ***(R.G.P.V., June 2010)***

Ans. **Definition** – Refer to Q.1.

Type of Language that a Turing Machine can Accept – Refer to Q.9.

Types of Turing Machine – Refer to Q.13.

Q.28. Explain Church's hypothesis.
(R.G.P.V., June 2011, Dec. 2012, June 2016)

Or

Write a brief note on Church's hypothesis. ***(R.G.P.V., Dec. 2002)***

Or

Write short note on Church's hypothesis. ***(R.G.P.V., Nov. 2018)***

Ans. The assumption that the intuitive notion of "computable function" can be identified with the class of partial recursive functions is known as Church's hypothesis or the Church-Turing thesis. While we cannot hope to "prove" Church's hypothesis as long as the informal notion of "computable" remains an informal notion, we can give evidence for its reasonableness. As long as our intuitive notion of "computable" places no bound on the number of steps or the amount of storage, it would seem that the partial recursive functions are intuitively computable, although some would argue that a function is not "computable" unless we can bound the computation in advance or at least establish whether or not the computation eventually terminates.

What is less clear is whether the class of partial recursive functions includes all "computable" functions. Logicians have presented many other formalisms such as the λ-calculus, post systems, and general recursive functions. All have been shown to define the same class of functions, i.e., the partial recursive functions. In addition, abstract computer models, such as the random access machine (RAM), also give rise to the partial recursive functions.

Note that a turing machine can simulate a RAM, provided that the elementary RAM instructions can themselves be simulated by a TM.

Q.29. Write short note on composite turing machine.

Ans. A TM is often described most simply by specifying several smaller TMs that combine to form it. We can construct a composite machine by executing first one TM and another. If T_1 and T_2 are TMs, with disjoint sets of nonhalting states and transition functions δ_1 and δ_2, respectively, we write T_1T_2 to denote this composite TM. The set of states is the union of the two sets. The TM T_1T_2 begins in the initial state of T_1 and executes the moves of T_1 (using the function δ_1) up to the point at which T_1 halts or crashes; for any move that causes T_1 to halt, T_1T_2 executes the same move, except that it moves instead to the initial state of T_2. At this point, the tape head is positioned at the square on which T_1 halted. From this point on, the moves of T_1T_2 are the moves of T_2 (using the function δ_2). If either T_1 or T_2 crashes during this process, then T_1T_2 crashes, T_1T_2 halts precisely if and when T_2 halts.

If we want to use this composite machine in a larger context, in a manner similar to a transition diagram but without showing the states explicitly, we might also write $T_1 \rightarrow T_2$.

We can also make the composition conditional, depending on the current tape symbol when T_1 halts. Thus, for an a that is either blank or a symbol in the tape alphabet of T_1, we might write

$$T_1 \xrightarrow{a} T_2$$

to stand for the composite machine $T_1T'T_2$, where T' is described by the diagram in fig. 5.18. This composite machine can be described

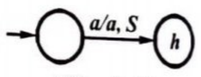


Fig. 5.18

informally as follows – it executes, the TM T_1; if and when T_1 halts, it executes T_2 if the current tape symbol is a and crashes otherwise.

In order to be able to describe composite TMs without having to describe every primitive operation as a separate TM, it is sometimes helpful to use mixed notation in which some but not all the states of some a TM are shown. For example, the diagram in fig. 5.19 (a), which is an abbreviated version of the one in fig. 5.19 (b), has a fairly obvious meaning. If the current symbol is a, execute the TM T, if it is b, halt; and if it is anything else; crash. In the first case, assuming T halts, repeat the execution of T until T halts scanning some symbol other than a. If at that point the tape symbol is b, halt, otherwise crash. (The machine might also crash during one of the iterations of T, and it might loop forever, either because one of the iterations of T does or because T halts with current tape symbol a every time).

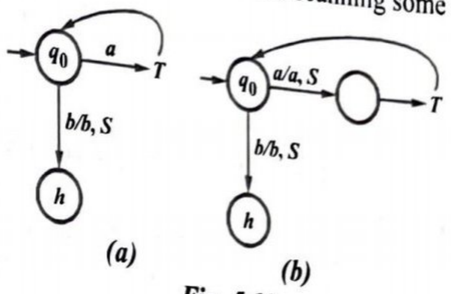


Fig. 5.19

Q.30. Differentiate between composite and iterated TM.

(R.G.P.V., June 2011)

Ans. Sometimes to obtain the solution we may combine two or more turing machine. T.M. will be combined in such a way that output of the first turing machine is input to the second, the output of the second T.M. is input to the third and so on. Also each of the turing machine is solving a smaller problem. This is known as composite turing machine.

For example, we can combine two turing machine to convert into the decimal form of the GCD of two unary numbers. For realizing a composite turing machine (CTM), the FM's of the component T.M's are appropriate, state rather the halt state at the completion of the performance of each component T.M.

Yet another way of having a combination of turing machine is by applying its own output as the next inputs.

The basic idea behind the composite and iterated turing machine is that we can design a more powerful turing machine which can solve a more complicated problem. The concept of splitting the problem into numbers of smaller problems and designing the turing machine has influenced the current computing.

Q.31. Explain how turing machine can be used as generating device.

(R.G.P.V., June 2009)

Or

Briefly elaborate the turing machines as enumerators.

(R.G.P.V., Nov. 2018)

Ans. Turing machine can be used as a generating device, generating strings over some alphabet, in addition to using it as a recognizer of language and computer of integer functions. A turing machine can be used as generator of strings as follows –

Consider a turing machine with multiple tapes. One of these tapes is used as output tape, which has the special property, that once a symbol is written on it, it cannot be changed, and tape head for this tape never moves to left. If the moves of this turing machine are designed in such a manner that it writes some string over some alphabet Σ on its output tape, separated by marker symbol #. Then set of all those strings that it can eventually write on its output tape between pair of #'s constitutes the language generated by it. Therefore, we formally define the language generated by the turing machine M as –

$G(M) = \{w | w$ is in Σ^*, and w is eventually written on the output tape between pair of #'s$\}$. For example consider a turing machine with following moves –

(i) It starts in the initial with blank tape, and replaces the blank symbol scanned by tape head with #, and enter into a state q_1, and moves tape head right.

(ii) In q_1 it replaces the next blank by 0, enters into state q_2, and moves the tape head right.

(iii) In state q_2, it either replaces the next blank by 1 or 0, and remains in state q_2, or replaces next blank by 1, and enters into state q_3, in both cases the tape head is moved right.

(iv) In state q_3, it replaces the next blank by 1, and enters into state q_4, moves tape head right.

(v) In state q_4, it replaces the next blank by #, and enters into state q_1.

The above turing machine, therefore, prints those strings that starts with 0, and ends will 11. Hence, the Turing machine generates the language given by the regular expression 0(0|1)*11.

The moves of the above turing machine can be represented by the following transition diagram shown in fig. 5.20.

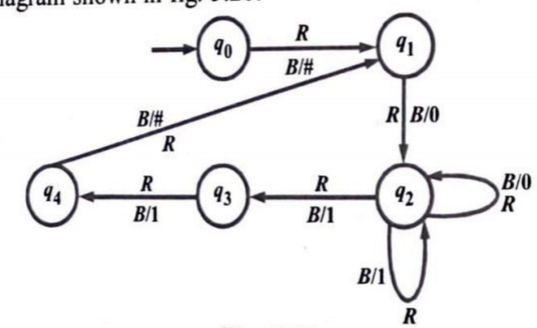


Fig. 5.20

NUMERICAL PROBLEMS

Prob.1. M is a turing machine represented by the transition system ahead. Obtain the computation sequence of M for processing the input string 0011 –

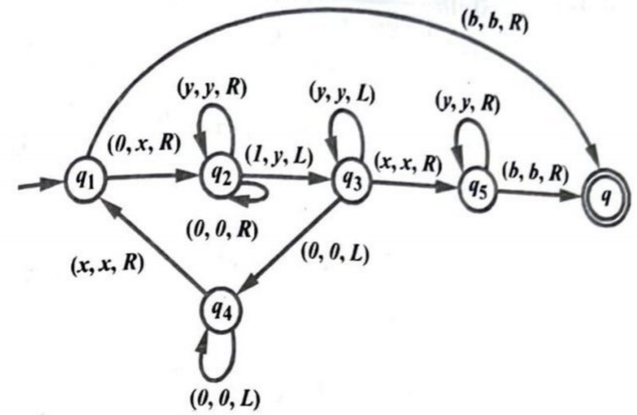


Fig. 5.21 Transition System for M

(R.G.P.V., Dec. 2009)

Sol. The initial tape input is $b0011b$. Let us assume that M is in state q_1 and the R/W head scans 0 (the first 0). We can represent this as in fig. 5.22.

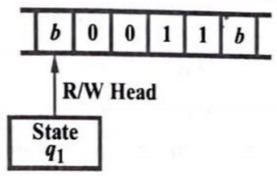


Fig. 5.22 TM Processing 0011

The figure can be represented by $\underset{q_1}{\underset{\downarrow}{b}}0011b$. From given fig. 5.21 we see that there is a directed edge from q_1 to q_2 with label (0, x, R). So the current symbol 0 is replaced by x and the head moves right. The new state is q_2.

Thus, we get $\underset{q_2}{\underset{\downarrow}{b}}x011b$. The change brought about by processing the symbol 0 can be represented as

$$\underset{q_1}{\underset{\downarrow}{b}}0011b \xrightarrow{(0,\,x,\,R)} \underset{q_2}{\underset{\downarrow}{b}}x011b$$

The entire computation sequence reads as follows –

$$\underset{q_1}{b0011b} \xrightarrow{(0,x,R)} \underset{q_2}{bx011b} \xrightarrow{(0,0,R)} \underset{q_2}{bx011b}$$

$$\xrightarrow{(1,y,L)} \underset{q_3}{bx0y1b} \xrightarrow{(0,0,L)} \underset{q_4}{bx0y1b} \xrightarrow{(x,x,R)} \underset{q_1}{bx0y1b}$$

$$\xrightarrow{(0,x,R)} \underset{q_2}{bxy1b} \xrightarrow{(y,y,R)} \underset{q_2}{bxy1b} \xrightarrow{(1,y,L)} \underset{q_3}{bxyyb}$$

$$\xrightarrow{(y,y,L)} \underset{q_3}{bxyyb} \xrightarrow{(x,x,R)} \underset{q_5}{bxyyb} \xrightarrow{(y,y,R)} \underset{q_5}{bxyyb}$$

$$\xrightarrow{(y,y,R)} \underset{q_5}{bxyyb} \xrightarrow{(b,b,R)} \underset{q_6}{bxyybb}$$

Prob.2. Design a turing machine to recognize all strings consisting of even number of 1's. ***(R.G.P.V., Dec. 2009)***

Sol. We design a turing machine to recognize all strings consisting of even number of 1's.

The construction is made by defining moves in the following manner –

(i) q_1 is the initial state. M enters state q_2 on reading 1 and writes B

(ii) If M is in state q_2 and reads 1, it enters q_1 and writes B

(iii) q_1 is the only accepting state.

So M accepts a string if it exhausts all input symbols and finally in state q_1. Symbolically,

$M = (\{q_1, q_2\}, \{1\}, \{1, B\}, \delta, q_1, B, \{q_1\})$

where δ is defined by the table 5.2.

Table 5.2 Transition Table

Present State	*Input 1*
$\rightarrow q_1$	(q_2, B, R)
q_2	(q_1, B, R)

Let us obtain the sequence of moves for string 11. Thus, $q_1 11 \vdash Bq_2 1 \vdash BBq_1$. As q_1 is the final state, 11 is accepted. $q_1 111 \vdash Bq_2 11 \vdash BBq_1 1 \vdash BBBq_2$. M halts in state q_2 and q_2 is not an accepting state. Thus, 111 is not accepted by M.

Prob.3. Construct TM accepting language

L = {w/w has even no. of 2}

(R.G.P.V., June 2015)

Sol. Refer to Prob.2 (Replace 1 by 2)

Prob.4. Design a TM that accepts the language denoted by the regular expression 00* for Σ = {0, 1}.

Sol. To solve the problem, we use the following logic –

(i) Change the state when first 0 comes, because regular expression contains atleast one 0.

(ii) The next state is final state for any number of 0.

(iii) Don't define any transition with input 1. So if at any time a 1 is encountered, the machine will halt without accepting the string. You can also create the transition with input 1 to any trap state. Symbolically,

$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1\}, \delta, q_0, B, \{q_1\})$

where δ is defined by table 5.3.

Table 5.3

Present State	*Input*	
	0	*1*
$\rightarrow q_0$	$(q_1, 0, R)$	$(q_2, 1, R)$
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$
q_2	–	–

Prob.5. Find a turing machine that recognizes the set of bit strings that have a 1 as their second bit over Σ = {0, 1}.

Or

Design a turing machine that recognizes the set of bit strings which have a 1 as their second bit i.e., the regular set (0 ∨ 1) 1 (0 ∨ 1)*.

Sol. We would like to have a turing machine, which, starting at the leftmost nonblank tape cell, moves right, and determines whether the second symbol is a 1. If the second symbol is 1, the machine should move into a final state. If the second symbol is not a 1, the machine should not halt or it should halt in a non-final state.

The required turing machine is defined as

$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1\}, \delta, q_0, B, \{q_2\})$

where δ is given by

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

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

$\delta(q_1, 1) = (q_2, 1, R)$

Here, q_2 is final state. No need to specify the transitions on q_2 because when the machine is on state q_2 it means that 1 is occurred as second bit so no need to traverse all the bits of string.

Prob.6. Construct turing machine for accepting L –

$L = \{a^n b^n | n \geq 0\}$ ***(R.G.P.V., June 2009)***

Or

Construct a turing machine for a language having equal number of a's and b's in it over the input set Σ = {a, b}. ***(R.G.P.V., Dec. 2014)***

Sol. The logic that we use for the turing machine to be constructed is as follows –

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 below in the form of transition table as shown in table 5.4.

Table 5.4 Transition Table

	a	b	B
$\rightarrow q_0$	(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,R)	
q_3	(q_3,a,R)	(q_3,b,R)	(q_0,B,R)
q_4			

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 δ is given above.

The transition diagram corresponding to the above table is shown below –

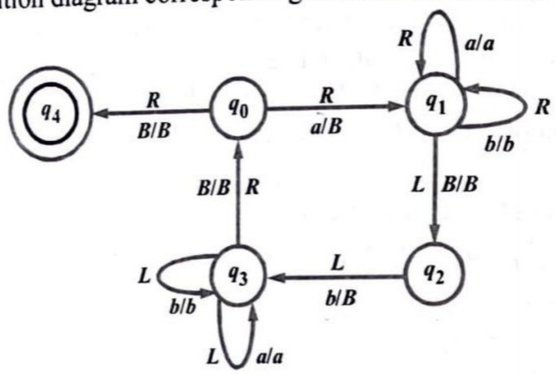


Fig. 5.23

Prob.7. Design a TM for language $\{L = a^n b^n | n \geq 1\}$.

(R.G.P.V., Dec. 2011)

Or

Design turing machine to accept –

$$L = \{a^n b^n | n \geq 1\}$$

(R.G.P.V., June 2010)

Or

Design a Turing machine for the following language –
The set of strings with an equal number of 0's and 1's.

(R.G.P.V., Dec. 2008)

Or

Design turing machine for the language –

$$L(G) = \{a^n b^n | n \geq 1\}$$

(R.G.P.V., Dec. 2016)

Or

Design a turing machine to accept the language $L = \{a^n b^n | n \geq 1\}$. Also illustrate the moves of turing machine for any input string in L.

(R.G.P.V., Nov. 2018)

Sol. Starting at the leftmost a, we replace it by a symbol, say x. Then we let the R/W head move to right to find the leftmost b, which in turn replaced by a symbol, say y. After that we return to the leftmost a, replace it with an x, then move to the leftmost b and replace it with y, and so on. Moving back and forth this way, we match each a with a corresponding b. If after sometime no a's or b's remain, then the string must be in L.

Working out the details, we arrive at a complete solution for which

$$Q = \{q_0, q_1, q_2, q_3, q_4\},$$
$$F = \{q_4\},$$
$$\Sigma = \{a, b\},$$
$$\Gamma = \{a, b, x, y, B\}.$$

The transitions can be broken into several parts. The set of the following transition –

$$\delta(q_0, a) = (q_1, x, R)$$
$$\delta(q_1, a) = (q_1, a, R)$$
$$\delta(q_1, y) = (q_1, y, R)$$
$$\delta(q_1, b) = (q_2, y, L)$$

replaces the leftmost a with an x, then causes the R/W head to move right to get the first b and replacing it with a y. When y is written, the machine enters a state q_2, indicating that an a has been successfully paired with a b.

The next set of transitions reverse the direction until an x is encountered, repositions the R/W head over the leftmost a, and returns control to the initial state.

$$\delta(q_2, y) = (q_2, y, L),$$
$$\delta(q_2, a) = (q_2, a, L),$$
$$\delta(q_2, x) = (q_0, x, R).$$

We are now back in the initial state q_0, ready to deal with the next a and b.

After one pass through this part of the computation, the machine will have carried out the partial computation.

$$q_0 aa \ldots abb \ldots b \overset{*}{\vdash} xq_0 a \ldots ayb \ldots b,$$

so that a single a has been matched with a single b. After two passes, we will have completed the partial computation.

$$q_0 aa \ldots abb \ldots b \overset{*}{\vdash} xxq_0 \ldots ayy \ldots b.$$

and so on indicating that the process of machine is being carried out properly.

When the input is a string $a^n b^n$, the rewriting continuous this way, halting only when there are no more a's and b's to be erased. When looking for the leftmost a, the R/W head moves to the left with the machine in state q_2. When an x is found, the direction is reversed to get the a. But now, instead of finding an a it will find a y. To terminate, a final check is made to see if all a's and b's have been replaced (to detect input where an a follows a b). This can be done by the following transitions

$$\delta(q_0, y) = (q_3, y, R)$$
$$\delta(q_3, y) = (q_3, y, R)$$
$$\delta(q_3, B) = (q_4, B, R)$$

If we input a string not in the language the computation will stop in some nonfinal state. For example, if we give the machine a string $a^n b^m$, with $n > m$, the machine will eventually find a blank in state q_1. It will halt because no transition is specified for this case. Other input not in the language, will lead to a nonfinal halting state.

The input $aabb$ gives the following successive instantaneous descriptions –

$$\begin{aligned} q_0aabb &\vdash xq_1abb \vdash xaq_1bb \vdash xq_2ayb \\ &\vdash q_2xayb \vdash xq_0ayb \vdash xxq_1yb \\ &\vdash xxyq_1b \vdash xxq_2yy \vdash xq_2xyy \\ &\vdash xxq_0yy \vdash xxyq_3y \vdash xxyyq_3B \\ &\vdash xxyyBq_4B. \end{aligned}$$

At this point the turing machine halts in a final state, so the string $aabb$ is accepted.

Prob.8. Build a turing machine that accepts the language

$$L = \{a^n b^{2n}\}$$

(R.G.P.V., Dec. 2013)

Sol. The resultant turing machine T is defined as

$$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where,

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}$

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

$\Gamma = \{a, b, B\}$

$F = \{q_f\}$

and δ is shown in transition table 5.5. and transition diagram is shown in fig. 5.24.

Table 5.5 Transition Table for *T*

States	*a*	*b*	*B*
$\rightarrow q_0$	(q_1, B, R)		(q_f, B, R)
q_1	(q_1, a, R)	(q_2, b, R)	
q_2		(q_2, b, R)	(q_3, B, L)
q_3		(q_4, B, L)	
q_4		(q_5, B, L)	
q_5	(q_5, a, L)	(q_5, B, L)	(q_0, B, R)
q_f	Accept	Accept	Accept

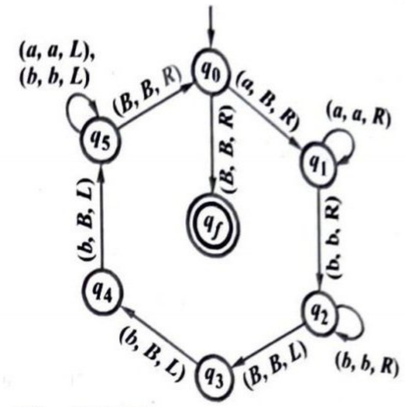


Fig. 5.24 Transition Diagram for T

Prob.9. Construct turing machine to accept the following language –

$$L = \{a^n b^n c^n | n \geq 1\}$$

(R.G.P.V., June 2008)

Or

Construct turing machine for the following set –

$$L = \{a^n\ b^n\ c^n \mid n \geq 1\}.$$

(R.G.P.V., Dec. 2006)

Or

Design a turing machine to accept –

$$L = \{a^n b^n c^n | \ n \geq 1\}.$$

(R.G.P.V., Dec. 2003)

Sol. The required turing machine M which accepts the language $L = \{a^n b^n c^n / n \geq 1\}$ is defined as

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{a, b, c\}, \{a, b, c, X, B\}, \delta, q_0, B, \{q_6\})$$

where δ is defined in transition table 5.6.

Table 5.6 Transition Table

Present State	*Input Tape Symbol*				
	a	*b*	*c*	*X*	*B*
$\rightarrow q_0$	(q_1, X, R)			(q_0, X, R)	
q_1	(q_1, a, R)	(q_2, X, R)		(q_1, X, R)	
q_2		(q_2, b, R)	(q_3, X, R)	(q_2, X, R)	
q_3			(q_4, C, L)		(q_5, B, L)
q_4	(q_4, a, L)	(q_4, b, L)		(q_4, X, L)	(q_0, B, R)
q_5				(q_5, B, L)	(q_6, B, R)
q_6	Accept				

Steps of Processing String –

(i) q_0 is initial state. If input = a then replace (a, X) and change state $q_0 \rightarrow q_1$. (direction $\rightarrow$ right)

(ii) State = q_1, for each $a \rightarrow$ no change.

If b encountered then replace (b, X) and change state $q_1 \rightarrow q_2$. (direction $\rightarrow$ right)

(iii) State = q_2, for each $b \rightarrow$ no change.

If c encountered then replace (c, X) and change state $q_2 \rightarrow q_3$ (direction $\rightarrow$ right)

(iv) q_3 is the state to check whether string is completely traversed or not. If not then change the state $q_3 \rightarrow q_4$ and direction – left otherwise, change the state $q_3 - q_5$ and direction – left.

(v) q_4 is the state to bring the head of turing machine to the start of the string, when B(blank) symbol encountered in q_4 state it changes the state $q_4 \rightarrow q_0$, keeping the blank symbol as it is and changing the direction to right.

(Now, since the string also contains an extra symbol X so add an extra δ transition on states q_0, q_1 and q_2 for input X without changing any state, direction or symbol).

(vi) q_5 is the state which checks whether

$n_a > n_c$ or $n_b > n_c$

Here, n_a is number of 'a' occurred in string. For 'c' its n_c and for 'b' its n_b. It replace only 'X' with B from scanning right to left. If it encounters any 'a' or 'b' then it will halt in non-final state. At the end when it receives a blank symbol it means that all the string is processed and the string is of the form $a^n b^n c^n$ so, it changes the state to final state i.e., q_6.

Prob.10. Define turing machine. Draw a turing machine for the language $\{0^n 1^n 0^n \mid n \geq 1\}$. **(R.G.P.V., Dec. 2007)**

Or

Design a turing machine to recognize the following language –

$$L = \{0^n 1^n 0^n \mid n \geq 1\}$$

(R.G.P.V., June 2004, Dec. 2012)

Sol. **Turing Machine** – Refer to Q.1.

Solution – Refer to Prob.9.

(Replace a with 0, b with 1 and c with 0)

Prob.11. Construct turing machine for language $L = 0^n 1^n 2^n \mid n \geq 1$. **(R.G.P.V., Dec. 2017)**

Sol. Refer to Prob.9.

(Replace a with 0, b with 1 and c with 2)

Prob.12. Construct TM for following language L over alphabets a, b and c –

$$L = \{a^n b^{2n} c^n \mid n \geq 0\}$$

Sol. This problem is similar or an extention of Prob.9. So the steps of making turing machine will be similar.

The difference is that, to recognize b^{2n}, add another state between q_1 and q_2 which replaces one more 'b' with X. And also for $n \geq 0$, the string must be accepted even for $\in$. So, add a stransition on state 'q_0' for input 'B' which change the state from q_0 to final state.

The required turing machine M which accepts $L = \{a^n b^{2n} c^n / n \geq 0\}$ is defined as

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}, \{a, b, c\}, \{a, b, c, X, B\}, \delta, q_0, B, \{q_7\})$$

where δ is defined in transition table 5.7.

Table 5.7 Transition Table

Present State	Input Tape Symbol				
	a	***b***	***c***	***X***	***B***
$\rightarrow q_0$	(q_1, X, R)			(q_0, X, R)	(q_1, B, R)
q_1	(q_1, a, R)	(q_2, X, R)		(q_1, X, R)	
q_2		(q_3, X, R)			
q_3		(q_3, b, R)	(q_4, X, R)	(q_3, X, R)	
q_4			(q_5, c, L)		(q_6, B, L)
q_5	(q_5, a, L)	(q_5, b, L)		(q_5, X, L)	(q_0, B, R)
q_6				(q_6, B, L)	(q_7, B, R)
q_7	Accept				

Prob.13. Construct TM for the following language L over the alphabets a, b and c –

$$L = \{a^n b^{2n} c^{3n} \mid n \geq 1\}.$$

Sol. This problem is similar or an extention of Prob.9. So the steps of designing turing machine will be similar.

The difference is that for recognizing b^{2n}, add another state between q_1 and q_2 and for recognizing c^{3n}, add two states between q_3 and q_4.

The required turing machine M which accepts $L = \{a^n b^{2n} c^{3n} / n \geq 1\}$ is defined as

$$M = \{Q, \Sigma, \Gamma, \delta, q_0, B, F\}$$

where $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_f\}$

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

$\Gamma = \{a, b, c, X, B\}$

$F = \{q_f\}$

and δ is defined in transition table 5.8.

Table 5.8 Transition Table

Present State	Input Tape Symbol				
	a	*b*	*c*	*X*	*B*
$\rightarrow q_0$	(q_1, X, R)			(q_0, X, R)	
q_1	(q_1, a, R)	(q_2, X, R)		(q_1, X, R)	
q_2		(q_3, X, R)			
q_3		(q_3, b, R)	(q_4, X, R)	(q_4, X, R)	
q_4			(q_5, X, R)		
q_5			(q_6, X, R)		
q_6			(q_7, C, L)		(q_8, B, L)
q_7	(q_7, a, L)	(q_7, b, L)		(q_7, X, L)	(q_0, B, R)
q_8				(q_8, B, L)	(q_f, B, R)
q_f	Accept				

Prob.14. Design turing machine for the language

$$L(G) = \{a^n b^m a^{n+m} | n \geq 1,\ m \geq 1\}.$$

(R.G.P.V., Dec. 2016)

Sol. The required turing machine T is defined as follows –

$$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_f\}$

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

$\Gamma = \{a, b, B\}$

$F = \{q_f\}$

and δ is given in transition table 5.9.

Table 5.9

States	*a*	*b*	*B*
$\rightarrow q_0$	(q_1, B, R)		
q_1	(q_1, a, R)	(q_2, b, R)	
q_2	(q_3, a, R)	(q_2, b, R)	
q_3	(q_3, a, R)		(q_4, B, L)
q_4	(q_5, B, L)		
q_5	(q_5, a, L)	(q_5, b, L)	(q_6, B, R)
q_6	(q_1, B, R)	(q_2, B, R)	(q_f, B, R)
q_f	Accept	Accept	Accept

and transition diagram is shown in fig. 5.25.

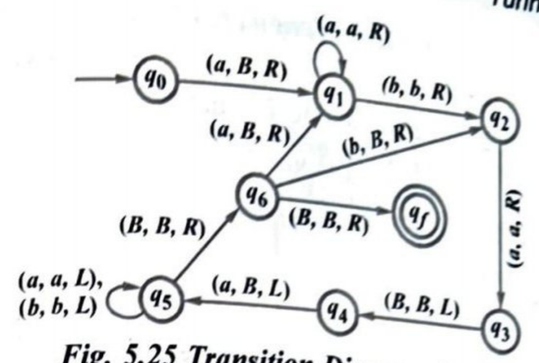


Fig. 5.25 Transition Diagram for T

Prob.15. All natural numbers are either even or odd. Construct a turing machine that computes the parity function for natural numbers, that is, that computes –

$$F(n) = \begin{cases} 0, & \text{if } n \text{ is even} \\ 1, & \text{if } n \text{ is odd} \end{cases}$$

(R.G.P.V., Dec. 2010)

Sol. Since last digit of any binary natural number decides whether the given number is even or odd. If the last digit of binary number is 0 then the number is even, otherwise odd.

(i) So for every 0 the state will be q_0 and for every 1, the state will be q_1. Scanning from left to right, changing the state according to bit and inserting the tape symbol as 0 untill the blank symbol encountered.

(ii) When blank symbol encountered –

(a) If the current state is q_0 then change state $q_0 \rightarrow q_2$, insert the tape symbol 0.

(b) If the current state is q_1 then change the state $q_0 \rightarrow q_2$, insert the tape symbol 1.

(iii) q_2 is the final state.

The required turing machine is defined as

$$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_2\})$$

where δ is defined in transition table 5.10.

Table 5.10 Transition Table

Present State	Input Tape Symbol		
	0	*1*	*B*
$\rightarrow q_0$	$(q_0, 0, R)$	$(q_1, 0, R)$	$(q_2, 0, R)$
q_1	$(q_0, 0, R)$	$(q_1, 0, R)$	$(q_2, 1, R)$
q_2	Halt		

Prob.16. Construct a TM for language of even no. of 1's and even no. of 0's over $\Sigma = \{0, 1\}$. **(R.G.P.V., Dec. 2011)**

Sol. The required turing machine T is defined as

$$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

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

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

$\Gamma = \{0, 1, B\}$

$F = \{q_f\}$

and δ transitions are shown in transition table 5.11.

Table 5.11

States	*0*	*1*	*B*
$\rightarrow q_0$	$(q_1, 0, R)$	$(q_3, 1, R)$	(q_f, B, L)
q_1	$(q_0, 0, R)$	$(q_2, 1, R)$	
q_2	$(q_3, 0, R)$	$(q_1, 1, R)$	
q_3	$(q_2, 0, R)$	$(q_0, 1, R)$	
q_f	Accept		

Transition diagram for turing machine T is shown in fig. 5.26.

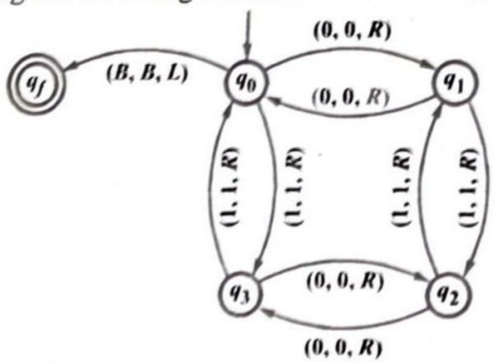


Fig. 5.26 Transition Diagram for Turing Machine T

Prob.17. Present a Turing machine that inserts symbol # in the beginning of a string on the Turing tape. Assume $\Sigma = \{a, b\}$. **(R.G.P.V., Dec. 2015)**

Sol. The required Turing machine T is defined as –

$$T = (Q, \Sigma, \Gamma, \delta, q, B, F)$$

where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}$

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

$\Gamma = \{a, b, B, \#\}$

$F = \{q_f\}$

and δ is given in transition table 5.12.

Table 5.12 δ Transition for T

States	*a*	*b*	*B*
$\rightarrow q_0$	(q_0, a, R)	(q_0, b, R)	(q_1, B, L)
q_1	(q_2, B, R)	(q_3, B, R)	(q_5, B, R)
q_2			(q_4, a, L)
q_3			(q_4, b, L)
q_4			(q_1, B, L)
q_5			$(q_f, \#, R)$
q_f	Halt		

and transition diagram is shown in fig. 5.27.

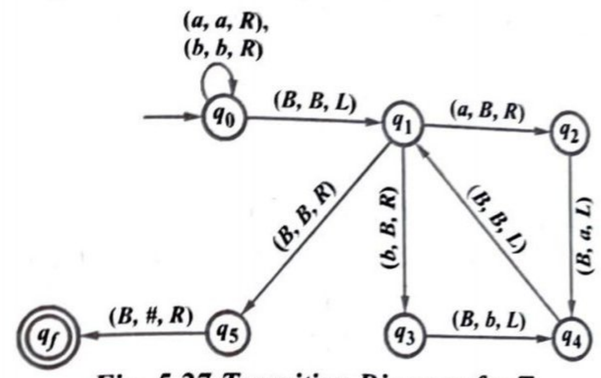


Fig. 5.27 Transition Diagram for T

Prob.18. Design a turing machine that copies string of 1's.

Or

Find a machine that performs the following computation –

$$q_0 u \vdash^* q_f uu,$$

for any $u \in \{1\}^+$.

Sol. To solve the problem, we implement the following intuitive process –

(i) Replace every 1 by an x

(ii) Find the rightmost x and replace it with 1

(iii) Move to the right end of the current non-blank region and create a 1 there

(iv) Repeat steps (ii) and (iii) until there are no more x's.

A turing machine version of this process is as follows –

$$\delta(q_0, 1) = (q_1, x, R)$$
$$\delta(q_1, 1) = (q_1, x, R)$$
$$\delta(q_1, B) = (q_2, B, L)$$
$$\delta(q_2, x) = (q_3, 1, R)$$
$$\delta(q_2, 1) = (q_2, 1, L)$$

$$\delta(q_2, B) = (q_f, B, R)$$
$$\delta(q_3, 1) = (q_3, 1, R)$$
$$\delta(q_3, B) = (q_2, 1, L)$$

Where q_f is the only final state. This may be somewhat hard to see at first, so let us trace the program with the simple string 11. The computation performed in this case is as follows –

$$q_0 11 \vdash xq_1 1 \vdash xxq_1 B \vdash xq_2 x \vdash x1q_3 B$$
$$\vdash xq_2 11 \vdash q_2 x11 \vdash 1q_3 11 \vdash 11q_3 1$$
$$\vdash 111q_3 B \vdash 11q_2 11 \vdash 1q_2 111 \vdash q_2 1111$$
$$\vdash q_2 B1111 \vdash q_f 1111$$

Prob.19. Construct a turing machine for checking the palindrome of a string of odd palindrome for $\Sigma = \{0, 1\}$. **(R.G.P.V., June 2015)**

Sol. Turing machine T for checking the palindrome of a string of odd palindrome for $\Sigma = \{0, 1\}$ is given by –

$$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}$
$\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, B\}$, $F = \{q_f\}$

and δ is given in form of transition table 5.13.

Table 5.13 Transition Table for Turing Machine of Odd Palindrome

States	*0*	*1*	*B*
$\rightarrow q_0$	(q_1, B, R)	(q_2, B, R)	
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_3, B, L)
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	(q_5, B, L)		(q_f, B, R)
q_4		(q_5, B, L)	(q_f, B, R)
q_5	$(q_5, 0, L)$	$(q_5, 1, L)$	(q_0, B, R)
q_f	Accept		

and transition diagram is shown in fig. 5.28.

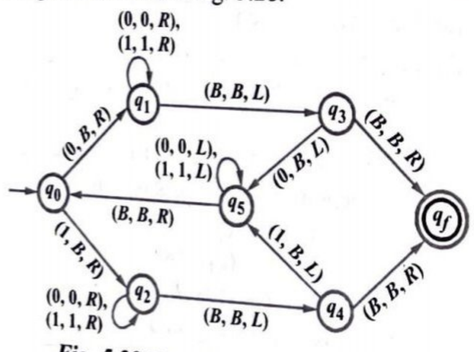


Fig. 5.28 Transition Diagram for T

Prob.20. Design a turing machine to recognize the following language – $\{ww^R \mid w \text{ is in } (0 + 1)^*\}$ where w^R is the reverse of w. **(R.G.P.V., June 2004)**

Or

Construct a turing machine that can accept the set of all even-length palindromes over {0, 1}.

Sol. Before designing the required turing machine M, let us evolve a procedure for processing the input string 1001. After processing, we require the ID to be of the form $BBBBq_6$.

We have the following steps for processing even-length palindromes –

Step 1 – The turing machine M scans the first symbol of the input tape (0, or 1), erase it and changes state (q_1 or q_2).

Step 2 – M scans the remaining part without changing the tape symbol until it encounters B.

Step 3 – The R/W head moves to the left. If the rightmost symbol tallies with the leftmost symbol is erased. Otherwise M halts.

Step 4 – The R/W head moves to the left until B is encountered.

Step 5 – Steps 1- 4 are repeated after changing the states suitably.

The changes of IDs due to processing of 1001 is given as –

$$q_0 1001 \vdash Bq_2 001 \overset{*}{\vdash} B001q_2 B \vdash B00q_4 1B \vdash B0q_5 0B \overset{*}{\vdash} q_5 B00B \vdash$$
$$Bq_0 00B \vdash BBq_1 0B \vdash BB0q_1 B \vdash BBq_3 0B \vdash Bq_5 BBB \vdash BBq_0 BB \overset{*}{\vdash}$$
$$BBBBq_6.$$

Thus, $q_0 1001 \overset{*}{\vdash} BBBBq_6$

As q_6 is an accepting state, the input string 1001 is accepted.

Now we can construct the transition table M. The transition table is given in table 5.14.

Table 5.14 Transition Table

Present State	*Input Tape Symbol*		
	a	*b*	*c*
$\rightarrow q_0$	(q_1, B, R)	(q_2, B, R)	(q_6, B, R)
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_3, B, L)
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	(q_5, B, L)		
q_4		(q_5, B, L)	
q_5	$(q_5, 0, L)$	$(q_5, 1, L)$	(q_0, B, R)
q_6	Accept		

It can be seen from the table that only the strings of the form ww^R are accepted where $w \in (0 + 1)^*$. Hence M recognizes the language

$$L = \{ww^R | w \in (0 + 1)^*\}$$

Prob.21. Design a turing machine M to recognize the language $\{ww : w \in \{a, b\}^*\}$. **(R.G.P.V., Dec. 2010)**

Sol. The idea behind the TM is to separate the processing into two parts – finding and marking the middle of the string and comparing the two halves. We accomplish the first task by working our way in from both ends simultaneously, changing symbols to their uppercase versions as we go. This means that our tape alphabet includes X and Y in addition to the input symbols a and b. Once we arrive at the middle – which happens only if the string is of even length – we may change the symbols in the first half back to their original form. The second part of the processing is to start at the beginning again and, for each lowercase symbol in the first half, compare it with the corresponding uppercase symbol in the second. We keep track of our progress by changing lowercase symbols to uppercase and erasing the matching uppercase symbols.

The required turing machine, which recognize the language $\{ww : w \in \{a, b\}^*\}$, is defined as

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

where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_f\}$

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

$\Gamma = \{a, b, X, Y, B\}$

$F = \{q_f\}$

and δ is defined in transition table 5.15.

Table 5.15 Transition Table

Present State	Input Tape Symbol				
	a	b	X	Y	B
$\rightarrow q_0$	(q_1, X, R)	(q_1, Y, R)	(q_4, X, L)	(q_4, Y, L)	(q_f, B, R)
q_1	(q_1, a, R)	(q_1, b, R)	(q_2, X, L)	(q_2, Y, L)	(q_2, B, L)
q_2	(q_3, X, L)	(q_3, Y, L)			
q_3	(q_3, a, L)	(q_3, b, L)	(q_0, X, R)	(q_0, Y, R)	
q_4			(q_4, a, L)	(q_4, b, L)	(q_5, B, R)
q_5	(q_6, X, R)	(q_7, Y, R)			(q_f, B, R)
q_6	(q_6, a, R)	(q_6, b, R)	(q_8, B, L)		(q_6, B, R)
q_7	(q_7, a, R)	(q_7, b, R)		(q_8, B, L)	(q_7, B, R)
q_8	(q_8, a, L)	(q_8, b, L)	(q_5, X, R)	(q_5, Y, R)	(q_8, B, L)
q_f	Halt				

The transition diagram of turing machine is shown in fig. 5.29.

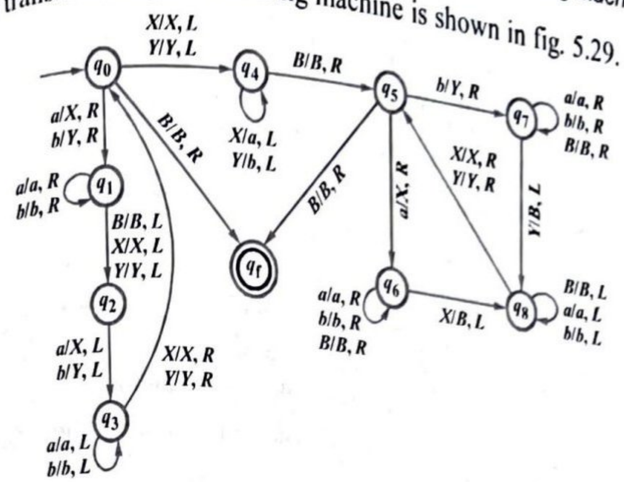


Fig. 5.29 A Turing Machine to Accept $\{ww | w \in \{a, b\}^*\}$

There are two ways that an input string can be rejected. If its length is odd, the TM will discover this in the first phase and crash. If it has even length but a symbol in the first half does not match the corresponding symbol in the second half, the TM will crash during the second phase.

Prob.22. Design turing machine to add two numbers a and b. **(R.G.P.V., June 2016)**

Sol. We first have to choose some convention for representing positive integers. We will use unary notation, for simplicity, in which any positive integer a is represented by

$$u(a) \in \{1\}^+ \text{ such that}$$
$$|u(a)| = a$$

We must also decide how a and b are placed on the tape initially and how their sum is to appear at the end of the computation. We will assume that $u(a)$ and $u(b)$ are on the tape in unary notation, separated by a single 0, with R/W head on the leftmost symbol of $u(a)$.

After the computation, $u(a + b)$ will be on the tape followed by a single 0, and the R/W head will be positioned at the left end of the result. Thus, we want to design a turing machine for performing the following computation –

$$q_0 u(a)\, 0 u(b) \vdash^* q_f\, u(a+b)\, 0,$$

where q_f is a final state. Constructing a program for this is relatively simple. All we need to do is to move the separating 0 to the right end of $u(y)$, so that the addition amounts to nothing more than the coalescing of the two strings. To achieve this, we construct $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, with

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

$$\delta(q_0,1) = (q_0,1,R),$$
$$\delta(q_0,0) = (q_1,1,R),$$
$$\delta(q_1,1) = (q_1,1,R),$$
$$\delta(q_1, B) = (q_2,B,L),$$
$$\delta(q_2,1) = (q_3,0,L),$$
$$\delta(q_3,1) = (q_3,1,L),$$
$$\delta(q_3,B) = (q_4,B,R),$$

It is noted that in moving the 0 right we temporarily create an extra 1, a fact that is remembered by putting the machine into state q_i. The transition $\delta(q_2, 1) = (q_3, 0, R)$ is needed to remove this at the end of the computation. This can be seen from the sequence of instantaneous descriptions for adding 111 to 11.

$$q_0 111011 \vdash 1q_0 11011 \vdash 11q_0 1011 \vdash 111q_0 011 \vdash 1111q_1 11 \vdash 11111q_1 1 + 111111q_1 B \vdash 11111q_2 1 \vdash 1111\, q_3 10 \overset{*}{\vdash} q_3 B111110 \vdash q_4 111110.$$

Adding numbers is one of the fundamental operations of any computer, one that plays a part in the synthesis of more complicated instructions. Other basic operations are copying string's and simple comparisons. Also this can be done on a turing machine.

Prob.23. Design a turing machine that adds two numbers presented in binary notation and leaves the answer on the tape in binary form.

(R.G.P.V., Dec. 2015)

Sol. Let us assume that binary numbers are stored on tape in following format –

···	B	X	+	Y	B	···

where, X and Y can be any binary number and no. of bits $(X) \geq$ no. of bits (Y) for example,

···	B	1	1	0	1	+	1	1	1	B	···

Then the turing machine which adds these binary numbers and leaves the answer on the tape in binary form is defined as

$$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}\}$

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

$\Gamma = \{0, 1, +, B, Y\}$

$F = \{q_{12}\}$

and δ transitions are defined in transition table 5.16.

The transition diagram of the resultant Turing machine is shown in fig. 5.30.

Table 5.16

States	0	1	+	Y	B
$\rightarrow q_0$	$(q_0, 0, R)$	$(q_0, 1, R)$	$(q_1, +, L)$		(q_{12}, B, L)
q_1	$(q_2, +, R)$	$(q_3, +, R)$			(q_{11}, B, R)
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	$(q_2, 0, R)$	(q_5, B, L)	(q_4, B, L)
q_3	$(q_3, 0, R)$	$(q_3, 1, R)$	$(q_3, 0, R)$	(q_6, B, L)	(q_5, B, L)
q_4	(q_7, B, R)	(q_8, B, R)			
q_5	(q_8, B, R)	(q_7, Y, R)			
q_6	(q_7, Y, R)	(q_8, Y, R)			
q_7					$(q_9, 0, L)$
q_8					$(q_9, 1, L)$
q_9				(q_{10}, Y, L)	(q_{10}, B, L)
q_{10}	$(q_{10}, 0, L)$	$(q_{10}, 1, L)$	$(q_1, +, L)$		
q_{11}	(q_{11}, B, R)		(q_{11}, B, R)	$(q_{12}, 1, R)$	(q_{12}, B, R)
q_{12}	Halt				

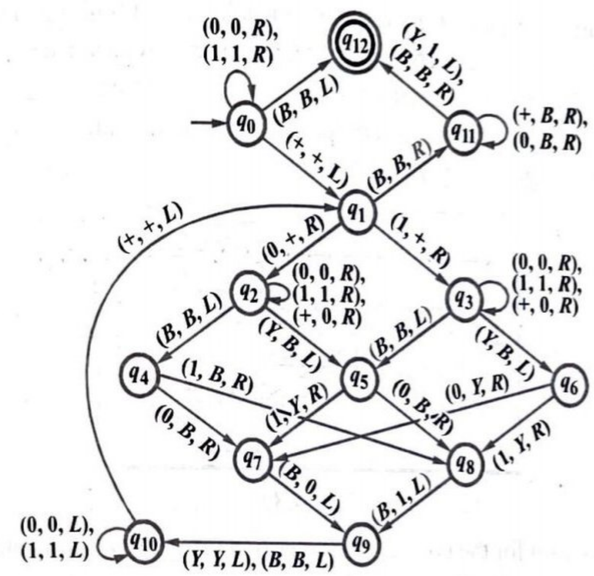


Fig. 5.30

Prob.24. Construct a turing machine to compute the function $f(w) = w^R$, where $w \in \{0, 1\}^+$. **(R.G.P.V., Dec. 2004)**

Sol. The turing machine M, which compute the function $F(\omega) = \omega^R$ for $\omega \in \{0, 1\}^+$ is defined as

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{0, 1\}, \{0, 1, X, B\}, \delta, q_0, B, \{q_6\})$$

where δ is defined in transition table 5.17.

Table 5.17 Transition Table

Present State	Input Tape Symbol			
	0	1	X	B
$\rightarrow q_0$	$(q_0, 0, R)$	$(q_0, 1, R)$		(q_1, B, L)
q_1	(q_2, X, R)	(q_4, X, R)	(q_1, X, L)	(q_5, B, R)
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	(q_2, X, R)	$(q_3, 0, L)$
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_1, X, L)	
q_4	$(q_4, 0, R)$	$(q_4, 1, R)$	(q_4, X, R)	$(q_3, 1, L)$
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_5, B, R)	
q_6	Accept			

Prob.25. Design a turing machine to compute the function $f(m, n) = m + n$ where m and n are non negative numbers. **(R.G.P.V., Dec. 2013)**

Sol. Assume that m and n are positive integers in unary representation.

The computation of $f(m, n)$ can be visualized at a high level by means of the diagram in fig. 5.31. The diagram depicts that we first use a comparing machine to determine whether or not $m \geq n$. If so, the comparer sends a start signal to the adder, which then computes $m + n$. If not, an erasing program is started that changes every 1 to a blank.

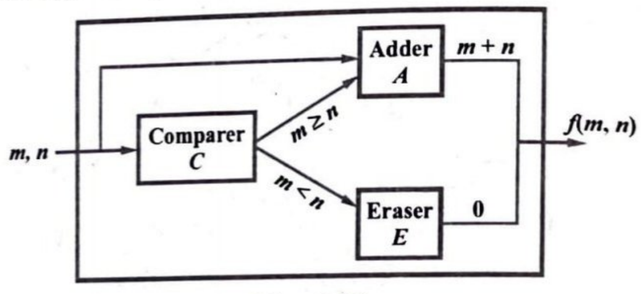


Fig. 5.31

The program for the comparer C is written using a Turing machine having states indexed with C. For the adder, we use states indexed with A. For the eraser E, we construct a Turing machine having states indexed with E. The computations to be done by C are –

$$q_{C,0}w(m)\,0w(n) \vdash^* q_{A,0}w(m)\,0w(n), \quad \text{if } m \geq n$$

and

$$q_{C,0}w(m)\,0w(n) \vdash^* q_{E,0}w(m)\,0w(n), \quad \text{if } m < n$$

If $q_{A,0}$ and $q_{E,0}$ are the initial states of A and E, respectively, we see that C starts either A or E.

The computations performed by the adder will be

$$q_{A,0}w(m)\,0w(n) \vdash^* q_{A,f}w(m+n)\,0,$$

and that of the eraser E will be

$$q_{E,0}w(m)\,0w(n) \vdash^* q_{E,f}\,0$$

The result is a single turing machine that combines the action of C, A and E as shown in fig. 5.31.

Prob.26. Proper subtraction $m \dot{-} n$ is defined to be $m - n$ for $m \geq n$ and zero $m < n$. Construct TM for this subtraction.

Or

Design a Turing machine that can compute proper subtraction i.e. ...m_n where m and n are positive integers –

$$m_n = m - n \quad \text{if } m > n$$
$$m_n = 0 \text{ if } m <= n$$

(R.G.P.V., Dec. 2010)

Or

Design a turing machine that computes a function $f(m, n) = m \dot{-} n$ i.e., proper subtraction of two integers defined as,

$$m \dot{-} n = m - n \quad \text{if } m > n$$
$$= 0 \quad \text{otherwise}$$

(R.G.P.V., June 2009)

Sol. Let TM

$$M = (\{q_0, q_1, \ldots\ldots, q_6\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_6\})$$

defined below started with 0^m10^n on its tape, halts with $0^{m \dot{-} n}$ 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^m10^n have all been 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's already have been changed. Then $n \geq m$, so $m \dot{-} n = 0$. M replaces all remaining 1's and 0's by B.

The function δ is described below –

(a) $\delta(q_0, 0) = (q_1, B, R)$

Begin the cycle. Replace the leading 0 by B.

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

$\delta(q_1, 1) = (q_2, 1, R)$

Search right, looking for the first (1).

(c) $\delta(q_2, 1) = (q_2, 1, R)$

$\delta(q_2, 0) = (q_3, 1, L)$

Search right pass 1's until encountering a 0. Change that 0 to 1.

(d) $\delta(q_3, 0) = (q_3, 0, L)$

$\delta(q_3, 1) = (q_3, 1, L)$

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

Move left to a blank. Enter state q_0 to repeat the cycle.

(e) $\delta(q_2, B) = (q_4, B, L)$

$\delta(q_4, 1) = (q_4, B, L)$

$\delta(q_4, 0) = (q_4, 0, L)$

$\delta(q_4, B) = (q_6, 0, R)$

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.

(f) $\delta(q_0, 1) = (q_5, B, R)$

$\delta(q_5, 0) = (q_5, B, R)$

$\delta(q_5, 1) = (q_5, B, R)$

$\delta(q_5, B) = (q_6, B, R)$

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.

A sample computation of M on input 0010 is –

$$q_0 0010 \vdash Bq_1 010 \vdash B0s_1 10 \vdash B01q_2 0 \vdash$$
$$B0q_3 11 \vdash Bq_3 011 \vdash q_3 B011 \vdash Bq_0 011 \vdash$$
$$BBq_1 11 \vdash BB1q_2 1 \vdash BB11q_2 \vdash BB1q_4 1 \vdash @@@@$$
$$BBq_4 1 \vdash Bq_4 \vdash B0q_6$$

On input 0100, M behaves as follows –

$$q_0 0100 \vdash Bq_1 100 \vdash B1q_2 00 \vdash Bq_3 110 \vdash$$
$$q_3 B110 \vdash Bq_0 110 \vdash BBq_5 10 \vdash BBBq_5 0 \vdash$$
$$BBBBq_5 \vdash BBBBBq_6$$

N-P COMPLETE PROBLEMS, DECIDABILITY AND RECURSIVELY ENUMERABLE LANGUAGES, DECIDABILITY, DECIDABLE LANGUAGES, UNDECIDABLE LANGUAGES, HALTING PROBLEM OF TURING MACHINE AND THE POST CORRESPONDENCE PROBLEM

Q.32. Define the term class P.

Or

Explain P-class problems with suitable example. (R.G.P.V., June 2015)

Ans. P is the class of problems which can be solve by a single tape deterministic turing machine in polynomial time.

So,

$$P = \bigcup_{k \geq 1} \text{DTIME}(n^k)$$

Example of P-class problem is – The PATH problem (Is there a path from s to t ?)

Q.33. Define the term class NP.

Or

Explain NP class problem with suitable example. (R.G.P.V., Dec. 2014)

Ans. NP is the class of problem which can be solve by a non-deterministic turing machine or it is a class of those languages which can be accepted by a non-deterministic turing machine in polynomial time.

So,

$$\text{NP} = \bigcup_{k > 1} \text{NTIME}(n^k)$$

The types of NP-class problems are given below –

(i) NP-complete (ii) NP-hard.

Some examples of NP class problems are as follows –

(i) Travelling salesman problem (ii) Graph colouring problem

(iii) Hamiltonian problem (iv) Vertex cover problem

(v) Subset sum problem.

Q.34. Define the relation between P and NP.

Ans. Every P-class problem is also the NP class problem but the reverse is fundamental unsolvable problem.

So, $\text{P} \subseteq \text{NP}$

P is the class of language for which membership can be decide quickly and NP is the class of language for which membership can be verified quickly whether $\text{NP} \subseteq \text{P}$ or $\text{P} = \text{NP}$ is the fundamental unsolvable problem in theory of computation. It is not known whether there exist some language in NP, which does not have a deterministic turing machine accepting it. Fig. 5.32 shows two possibilities of relationship between P and NP, in which only one is correct.

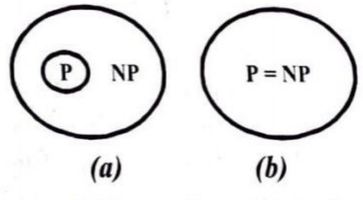


Fig. 5.32 Either (a) or (b) is Correct

Q.35. Explain P and NP problems. ***(R.G.P.V., June 2016)***

Or

Define P and NP problems. ***(R.G.P.V., Dec. 201.)***

Or

How P class problems different from NP class problem ? **(R.G.P.V., Dec. 2016)**

Ans. Refer to Q.32, Q.33 and Q.34.

Q.36. The set of all languages whose complements are in NP is called CO-NP. Prove that NP = CO-NP if and only if there is some NP-complete problem whose complement is in NP. **(R.G.P.V., Dec. 2010)**

Ans. **Proof –** The "only if" part is obvious. For the "if" part let S be an NP-complete problem, and suppose $\overline{S}$ were in NP. Since each L in NP is log-space reducible to S, each $\overline{L}$ is log-space reducible to $\overline{S}$. Thus, $\overline{L}$ is in NP.

Q.37. Write short note on NP-complete vs NP-hard problem. **(R.G.P.V., Dec. 2013)**

Or

Difference between NP complete vs NP hard problem. **(R.G.P.V., June 2016)**

Or

Explain NP complete and NP hard. **(R.G.P.V., May 2018)**

Or

Explain NP-complete and NP-hard problems with suitable examples. **(R.G.P.V., Nov. 2018)**

Or

Write short note on NP complete problems. **(R.G.P.V., Nov. 2019)**

Ans. The study of the relation between the complexity classes P and NP has generated particular interest among computer scientists. At the root, this is the question whether or not –

$$P = NP$$

This is one of the fundamental unsolved problem in the theory of computation. To explore it, computer scientists have introduced a variety of related concepts. One of them is the idea of an NP-complete problem. Loosely speaking, an NP-complete problem is one that is as hard as any NP problem and in some sense is equivalent to all of them.

A language L_1 is said to be ***polynomial-time reducible*** to some language L_2 if there exists a deterministic Turing machine by which any u_1 in the alphabet of L_1 can be transformed in polynomial time to a u_2 in the alphabet of L_2 in such a way that $u_1 \in L_1$ iff $u_2 \in L_2$.

We can see that if L_1 polynomial-time reducible to L_2, and if $L_2 \in P$, then $L_1 \in P$. Similarly, if $L_2 \in NP$, then $L_1 \in NP$.

A language $L \subseteq \{0, 1\}^*$ is ***NP-complete*** if

(i) $L \in NP$, and (ii) $L' \leq_p L$ for every $L' \in NP$.

If a language satisfies the property (ii), but not necessarily property (i), then L is ***NP-hard.***

It follows easily from these definitions that if some L_1 is NP-complete and polynomial-time reducible to L_2, then L_2 is also NP-complete. The implication of this definition is that if we can find a deterministic polynomial-time algorithm for any NP-complete language, then every language in NP is also in P, i.e.,

$$P = NP.$$

Q.38. Explain P, NP, NP complete problems with example.

Ans. Refer to Q.32, Q.33 and Q.37. **(R.G.P.V., Dec. 2012)**

Q.39. What do you understand by P, NP, NP complete and NP hard problems. **(R.G.P.V., June 2010)**

Or

Define the following –

(i) Polynomial problems
(ii) Non-deterministic polynomial (NP) problems
(iii) NP complete problems
(iv) NP hard problems.

Ans. Refer to Q.32, Q.33 and Q.37. **(R.G.P.V., Dec. 2010)**

Q.40. Discuss the relationship between class P, NP, NP complete and NP hard problems with example of each class.

Or

Explain relation between P, NP, NP hard and NP complete problems with diagram. **(R.G.P.V., Dec. 2011)**

Or

Draw and explain commonly believed relationship between class P, NP, NP-complete and NP-hard. **(R.G.P.V., Dec. 2015)**

Ans. Refer to Q.34 for P and NP.

A problem L is NP-hard if and only if satisfiability reduce to L (satisfiability $\propto L$). A problem is NP-complete if and only if L is NP-hard and $L \in NP$. All NP-complete problems are NP-hard but all NP-hard problems cannot be NP-complete. If an NP-hard problem can be solved in polynomial time then all NP-complete problems can also be solved in polynomial time.

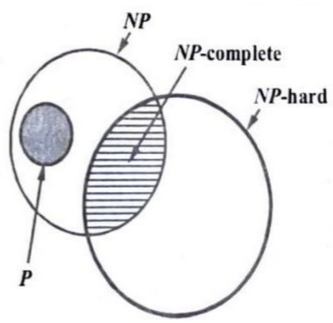


Fig. 5.33

Only a decision problem can be NP-complete. However, an optimization problem may be NP-hard. Furthermore if L_1 is a decision problem and L_2 an optimization

problem, then it is possible that $L_1 \propto L_2$. One can trivially show that the knapsack decision problem reduces to the knapsack optimization problem. For the clique problem one can easily show that the clique decision problem reduces to the clique optimization problem. In fact, one can also show that these optimization problems reduce to their corresponding decision problems. Fig. 5.33 shows the relationship among these classes.

Q.41. What is NP hard problems ? ***(R.G.P.V., June 2011, Dec. 2016)***

Or

What are NP hard problems ? Write a brief note on it. ***(R.G.P.V., Dec. 2009)***

Or

Explain what is NP-hard problem. ***(R.G.P.V., June 2015)***

Ans. Refer to Q.37 and Q.40.

Q.42. Define the following –

(i) Polynomial time reduction and NP-completeness ***(R.G.P.V., Dec. 2010)***

(ii) Cook's theorem. ***(R.G.P.V., Dec. 2010, 2011)***

Ans. (i) Polynomial Time Reduction and NP-completeness – Refer to Q.37.

(ii) Cook's Theorem – The statement that the satisfiability problem is *NP*-complete is known as Cook's theorem.

Q.43. Write short note on satisfiability problem. ***(R.G.P.V., June 2015, Dec. 2017)***

Ans. An expression is satisfiable if there is some assignment of 0's and 1's to the variables that gives the expression the value 1. The satisfiability problem is to determine, given a Boolean expression, whether it is satisfiable.

We may represent the satisfiability problem as a language L_{sat} as follows –

Let the variables of some expression be $x_1, x_2 \ldots\ldots x_m$ for some m. Code x_i as the symbol x followed by i written in binary. The alphabet of L_{sat} is thus.

$$\{\wedge, \vee, \neg, (,), x, 0, 1\}.$$

The length of the coded version of an expression of n symbols is easily seen to be no more than $[n \log_2 n]$, since each symbol other than a variable is coded by one symbol, there are no more than $[n/2]$ different variables in an expression of length n, and the code for a variable requires no more than $1 + [\log_2 n]$ symbols. We shall henceforth treat the word in L_{sat} representing an expression of length n as if the word itself ware of length n. Our results will not depend on whether we use n or $n \log n$ for the length of the word, since $\log (n \log n) \leq 2 \log n$, and we shall deal with log space reductions.

Q.44. Show that the satisfiability problem is NP-complete.

***Ans.* Proof** – To determine if an expression of length n is satisfiable, non-deterministically guess values for all the variables and then evaluate the expression. Thus L_{sat} is in *NP*.

To show that every language in *NP* is reducible to L_{sat}, for each NTM M that is time bounded by a polynomial $p(n)$, we give a log-space algorithm that takes as input a string x and produces a Boolean formula E_x that is satisfiable if and only if M accepts x. We now describe E_x.

Let $\# \beta_0 \# \beta_1 \# \ldots\ldots \# \beta_{p(n)}$ be a computation of M, where each β_i is an ID consisting of exactly $p(n)$ symbols. If acceptance occurs before the $p(n)$th move, we allow the accepting ID to repeat, so each computation has exactly $p(n) + 1$ ID's. In each ID we group the state with the symbol scanned to form a single composite symbol. Numbers are assigned to moves by arbitrarily ordering the finite set of choices that M may make given a state and tape symbol.

For each symbol that can appear in a computation and for each i, $0 \leq i < (p(n) + 1)^2$, we create a Boolean variable C_{ix} to indicate whether the ith symbol in the computations is X. The expression E_x that we shall construct will be true for a given assignment to the C_{ix}'s if and only if the C_{ix}'s that are true correspond to a valid computations.

The expression E_x states the following –

(i) The C_{ix}'s that are true correspond to a string of symbols in that exactly one C_{ix} is true for each i

(ii) The ID β_0 is an initial ID of M with input x

(iii) The last ID contains a final state

(iv) Each ID follows from the previous one by the move of M that is indicated.

The first formula, stating that for each i between 0 and $(p(n) + 1)^2 - 1$ exactly one C_{ix} is true is

$$\bigwedge_i \left[\bigvee_x C_{ix} \wedge \neg \left(\bigvee_{x \neq Y} (C_{ix} \wedge C_{iY}) \right) \right]$$

For a given value of i the term $\vee x C_{ix}$ forces at least one C_{ix} to be true and $\neg \vee_{x \neq Y} (C_{ix} \wedge C_{iY})$ forces at most one to be true.

Let $x = a_1, a_2, \ldots\ldots a_n$. The second formula expressing the fact that β_0 is an initial I_D is in turn the AND of the following terms.

(i) $C_{0\#} \wedge C_{p(n)+1\#}$. The symbols in positions 0 and $p(n) + 1$ are #

(ii) $C_{1Y1} \vee C_{1Y2} \vee \ldots\ldots \vee C_{1YK}$, where $Y_1, Y_2 \ldots\ldots Y_K$ are all the composite symbols that represent tape symbol a_1 the start state q_0, and the number of a legal move of M in state q_0 reading symbol a_1. This clause states that the first symbol of β_0 is correct

(iii) $\bigwedge_{2 \le i \le n} C_{ia_i}$. The 2nd through nth symbols of β_0 are correct

(iv) $\bigwedge_{n < i \le p(n)} C_{iB}$. The remaining symbols of β_0 are blank.

The 3rd formula says that the last ID has an accepting state. It can be written as

$$\bigvee_{p(n)(p(n)+1) < i < (p(n)+1)^2} \left(\bigvee_{X \text{ in } F} C_{ix} \right)$$

where F is the set of composite symbols that include a final state.

A predicate $f(W, X, Y, Z)$ that is true if and only if symbol Z could appear in position j of some ID given that W, X and Y are the symbols in positions $j-1$, j, and $j+1$ of the previous ID [W is # if $j = 1$ and Y is # if $j = p(n)$]. It is convenient also to declare $f(W, \#, X, \#)$ to be true, so we can treat the markers between ID's as we treat the symbols within ID's. We can now express the fourth formula as –

$$\bigwedge_{p(n)<j<(p(n)+1)^2} \left(\bigwedge_{\substack{W,X,Y,Z \text{ such} \\ \text{that } f(W,X,Y,Z)}} \left(C_{j-p(n)-2,W} \wedge C_{j-p(n)-1,x} \wedge C_{j-p(n),Y} \wedge C_{jZ} \right) \right)$$

Given an assignment of truth values making E_x true, the four formulas above guarantee that there is an accepting computation of M on x.

The formulas composing E_x are of length $O(p^2(n))$ and are sufficiently simple that a log-space TM can generate them given x on its input, the TM only needs sufficient storage to count upto $(p(n) + 1)^2$.

Since the logarithm of a polynomial in n is some constant times $\log n$, this can be done with $O(\log n)$ storage. We have thus shown that every language in NP is log-space reducible to L_{sat}, proving that L_{sat} is NP-complete.

Q.45. Discuss satisfiability problem. Prove that satisfiability is NP complete. ***(R.G.P.V., Dec. 2009)***

Ans. Refer to Q.43 and Q.44.

Q.46. Explain process of reducibility. ***(R.G.P.V., June 2016)***

Or

Write short note on reducibility. ***(R.G.P.V., Nov. 2019)***

Ans. In reducibility process, a problem is converted to another problem in such a way that a solution to second problem can be used to solve the first problem. Such reducibilities come up in everyday life even if we do not usually refer to them in this way. As an example, consider you want to find our way around a new city. It is easy to do if you had a map. Thus you can reduce the problem of finding your way around the city to the problem of obtaining a map. There are always two problems in reducibility, which we call A and B. If A reduces to B, we can use a solution to B to solve A. Reducibility says

nothing about solving A and B alone, but only about the solvability of A in the presence of a solution to B. Reducibility also occurs n mathematical problems. Reducibility plays an important role in classifying problems by decidability and later in complexity theory.

Q.47. Explain vertex cover problem.

Or ***(R.G.P.V., Dec. 2012)***

Write short note on vertex cover problem.

(R.G.P.V., Dec. 2013, Nov. 2018)

Or

Explain vertex cover problem with suitable example.

(R.G.P.V., June 2015, Dec. 2017)

Or

Define and discuss vertex cover problem. ***(R.G.P.V., Dec. 2015)***

Ans. Let $G = (V, E)$ be an (undirected) graph with set of vertices V and edges E. A subset $A \subseteq V$ is said to be vertex cover of G if for every edge (v, w) in E, at least one of v or w is in A.

The vertex cover problem is – given a graph G and integer $\bar{k}$, does G have a vertex cover of size k or less ?

To represent this problem as a language L_{vc}, consisting of strings of the form– k in binary, followed by a marker, followed by the list of vertices, where v_i is represented by v followed by i in binary, and a list of edges, where (v_i, v_j) is represented by the codes for v_i and v_j surrounded by parentheses. L_{vc} consists of all such strings representing k and G, such that G has a vertex cover of size k or less.

Q.48. Prove that the L_{vc}, the vertex cover problem, is NP-complete.

Or

Prove that vertex cover is NP-complete problem.

(R.G.P.V., June 2011, Dec. 2011)

Or

Prove that vertex cover problem is NP-complete problem.

(R.G.P.V., Dec. 2014)

Ans. **Proof** – Let guess a subset of k vertices and check that it covers all edges. This may be done in time proportional to the square of the length of the problem representation. L_{vc} is shown to be NP-complete by reducing 3-CNF satisfiability to L_{vc}.

Let $F = F_1 \wedge F_2 \wedge \ldots\ldots\ldots \wedge F_q$ be an expression in 3-CNF, where each F_i is a clause of the form $(\alpha_{i1} \vee \alpha_{i2} \vee \alpha_{i3})$ each α_{ij} being a literal. We construct an undirected graph $G = (V, E)$ whose vertices are pairs if integers (i, j), $1 \le i \le q$, $1 \le j \le 3$. The vertex (i, j) represents the jth literal of the ith clause. The edges of the graph are

(i) $[(i, j), (i, k)]$ provided $j \neq k$ and

(ii) $[(i, j), (k, l)]$ if $\alpha_{ij} = \neg \alpha_{kl}$.

Each pair of vertices corresponding to the same clauses are connected by an edge in (i). Each pair of vertices corresponding to a literal and its complement are connected by an edge in (ii).

G has been constructed so that it has a vertex cover of size $2q$ if and only if F is satisfiable. Assume F is satisfiable and fix an assignment satisfying F. Each clause must have a literal whose value is 1. Select one such literal for each clause. Delete the q vertices corresponding to these literal from V. The remaining vertices form a vertex cover of size $2q$. Clearly for each i, only one vertex of the form (i, j) is missing from the cover, and hence each edge in (i) is incident upon at least one vertex in the cover. Since edges in (ii) are incident upon two vertices corresponding to some literal and its complement, and since we would not have deleted both a literal and its complement, one or the other of these vertices is in the cover. Thus we indeed have a cover of size $2q$.

Conversely, assume we have a vertex cover of size $2q$, for each i the cover must contain all but one vertex of the form (i, j) for if two such vertices were missing, an edge $[(i, j), (i, k)]$ would not be incident upon any vertex in the cover. There can be no conflict because two vertices not in the cover cannot correspond to a literal and its complement, else there would be an edge in group (ii) not incident upon any vertex of the cover. For this assignment F has value 1. Thus F is satisfiable. We can essentially use the variables names in the formula F as the vertices of G, appending two bits for the j-component in vertex (i, j). Edges of type (i) are generated directly from the clauses, while those of type (ii) require two counters to consider all pairs of literals. Thus we conclude that L_{vc} is NP-complete.

Q.49. Describe vertex cover problem. ***(R.G.P.V., June 2016)***

Or

Explain vertex cover problem briefly. ***(R.G.P.V., Dec. 2016)***

Ans. Refer to Q.47 and Q.48.

Q.50. What is Hamilton circuit problem ? Prove that directed Hamilton circuit is NP-complete. ***(R.G.P.V., Dec. 2009)***

Or

What is Hamiltonian path problem ? Discuss with example. ***(R.G.P.V., June 2011)***

Or

Write short note on Hamiltonian path problem. ***(R.G.P.V., Dec. 2013, June 2015, Dec. 2017, Nov. 2018)***

Or

Describe Hamiltonian path problem. ***(R.G.P.V., June 2016)***

Or

What is Hamiltonian path problem ? Explain. ***(R.G.P.V., Dec. 2016)***

Ans. The Hamiltonian circuit problem is – Given a graph G, does G contain a path that visits each vertex exactly once and returns to its starting point ?

Proof – We first show that HAM-CYCLE belongs to NP. Given a graph $G = (V, E)$, our certificate is the sequence of $|V|$ vertices that make up the Hamiltonian cycle. The verification algorithm checks that this sequence contains each vertex in V exactly once and that with the first vertex repeated at that end, if forms a cycle in G. This verification can be performed in polynomial time.

We now prove that HAM-CYCLE is NP-complete by showing that 3 CNF-SAT $\leq p$ HAM-CYCLE. Given a 3-CNF boolean formula ϕ over variables $x_1, x_2, \ldots, x_n$ with clauses $C_1, C_2, \ldots, C_k$ each containing exactly 3 distinct literals, we construct a graph $G = (V, E)$ in polynomial time such that G has a Hamiltonian cycle if and only if ϕ is satisfiable. Our construction is based on ***widgets***, which are pieces of graphs that enforce certain properties.

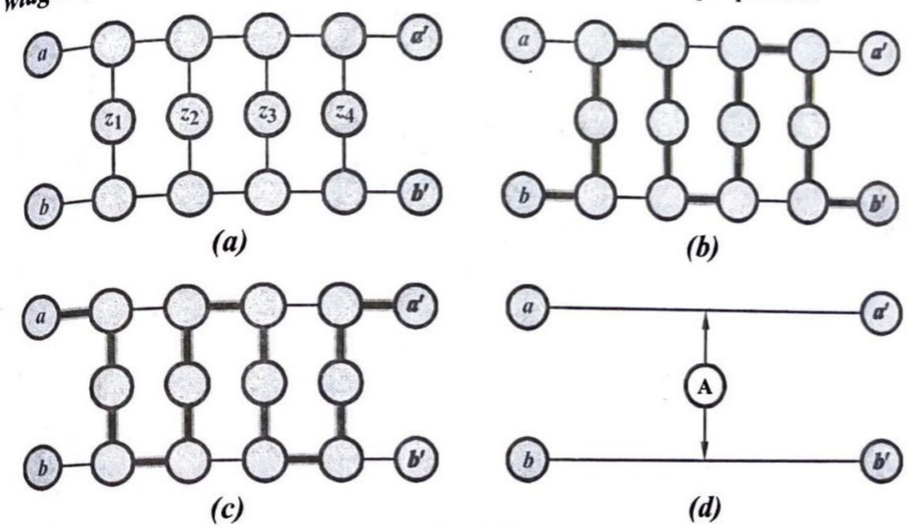


Fig. 5.34

Our first widget is the subgraph A shown in fig. 5.34 (a). Suppose that A is a subgraph of some graph G and that the only connections between A and the remainder of G are through the vertices a, a', b and b'. Furthermore, suppose that graph G has a Hamiltonian cycle. Since any Hamiltonian cycle of G must pass through vertices z_1, z_2, z_3 and z_4 in one of the ways shown in fig. 5.34 (b) and (c), we may treat subgraph A as if it were simply a pair of edges (a, a') and (b, b') with the restriction that any Hamiltonian cycle of G must include exactly one of these edges. We shall represent widget A as shown in fig. 5.34 (d).

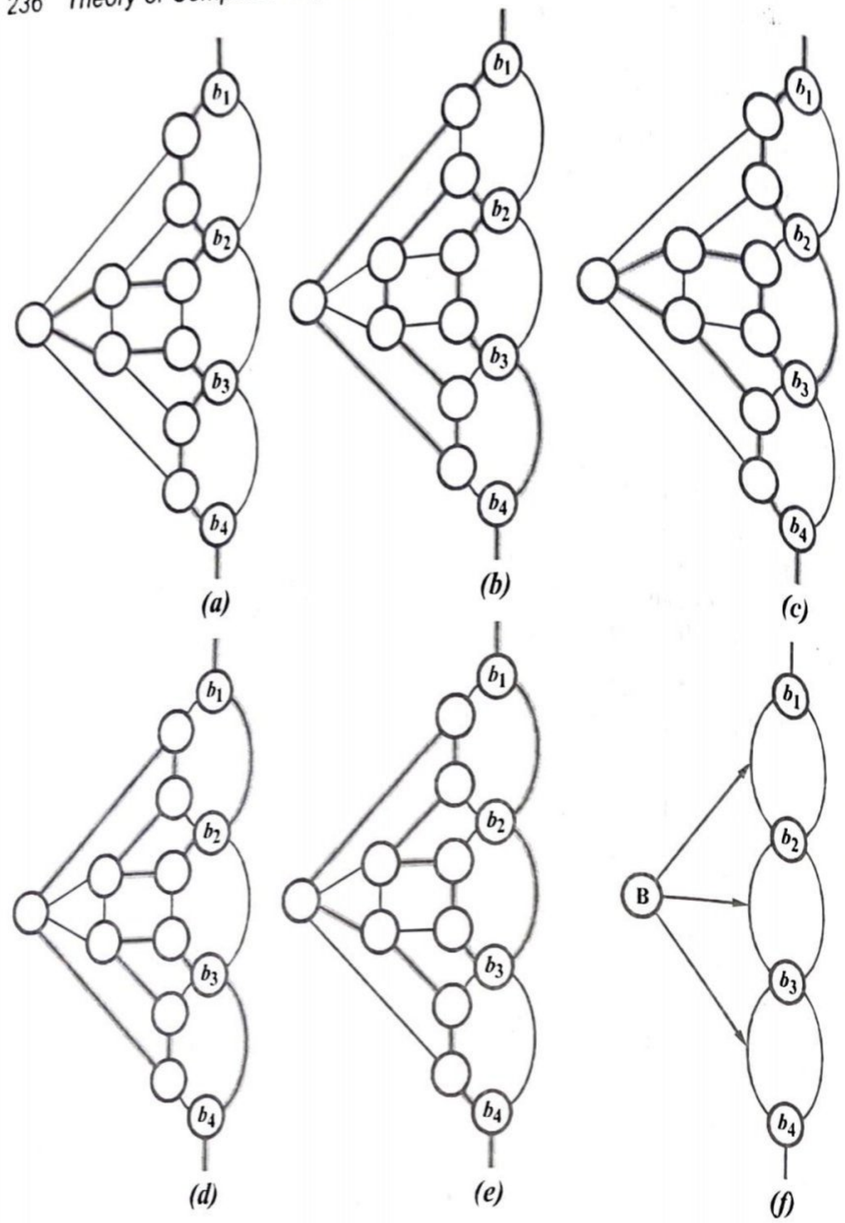


Fig. 5.35

The subgraph B in fig. 5.35 is our second widget. Suppose that B is a subgraph of some graph G and that the only connections from B to the remainder of G are through vertices b_1, b_2, b_3 and b_4. A Hamiltonian cycle of graph G cannot traverse all of the edges (b_1, b_2), (b_2, b_3) and (b_3, b_4), since then all vertices in the widget other than b_1, b_2, b_3 and b_4 would be missed. However, a Hamiltonian cycle of G may traverse any proper subset of these edges. Fig. 5.35 (a) to (e) show five such subsets; the remaining two subsets can be obtained by performing a top-to-bottom flip of parts (b) and (e). This widget

is represented as in fig. 5.35 (f), the idea being that at least one of the paths pointed to by the arrows must be taken by a Hamiltonian cycle.

The graph G that we shall construct consists mostly of copies of these two widgets. The construction is shown in fig. 5.36. For each of the k clauses in ϕ, we include a copy of widget B, and we join these widgets together in series as follows. Letting b_{ij} be the copy of vertex b_j in the ith copy of widget B, we connect $b_{i,4}$ to $b_{i+1,1}$ for $i = 1, 2, \ldots, k-1$.

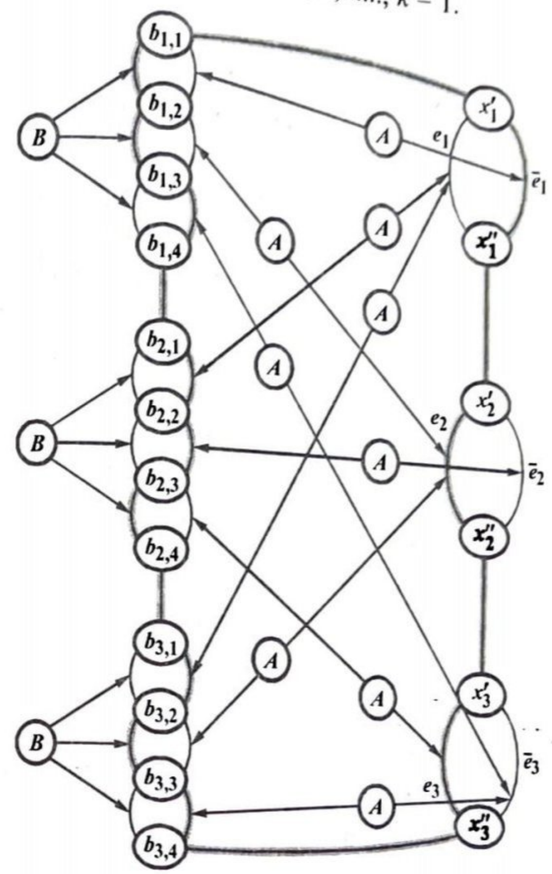


Fig. 5.36

Then for each variable x_m in ϕ, we include two vertices x'_m and x''_m. These two vertices are connected by means of two copies of the edge (x'_m, x''_m), which are denoted by e_m and $\bar{e}_m$. The idea is that if the Hamiltonian cycle takes edge e_m, it corresponds to assigning variable x_m the value 1. If the Hamiltonian cycle takes edge $\bar{e}_m$ the variable is assigned the value 0. Each pair of these edges forms a two-edge loop. These small loops are connected in series by adding edges (x'_m, x''_{m+1}) for $m = 1, 2, \ldots, n-1$. The left (clause) side of the graph is connected to the right (variable) side by means of two edges $(b_{1,1}, x'_1)$ and $(b_{k,4}, x''_n)$ which are the topmost and bottommost edges in fig. 5.36.

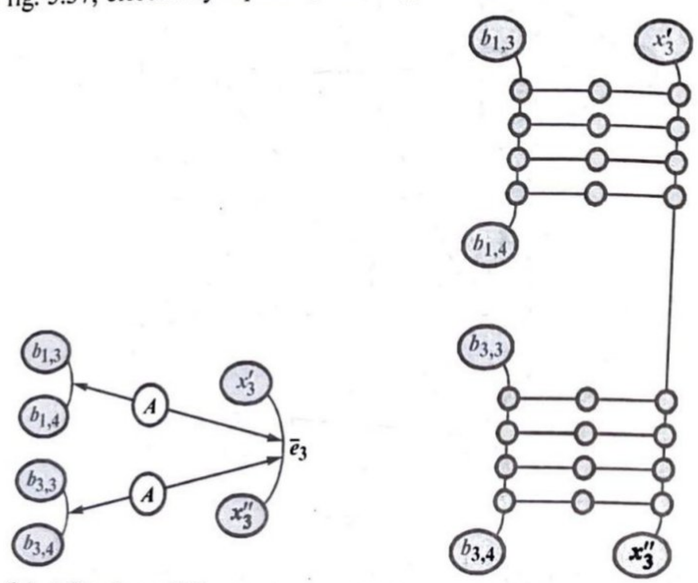
We are not yet finished with the construction of graph G, since we have yet to relate the variables to the clauses. If the jth literal of clause c_i is x_m, then we use an A widget to connect edge $(b_{ij}, b_{i,j+1})$ with edge e_m. If the jth literal of clause c_i is $\neg x_m$, then we instead put an A widget between edge $(b_{ij}, b_{i,j+1})$ and edge $\overline{e}_m$. In fig. 5.36, for example, because clause C_2 is $(x_1 \vee \neg x_2 \vee x_3)$, we place three A widgets as follows –

(i) between $(b_{2,1}, b_{2,2})$ and e_1

(ii) between $(b_{2,2}, b_{2,3})$ and $\overline{e}_2$

(iii) between $(b_{2,3}, b_{2,4})$ and e_3.

Note that connecting two edges by means of A widgets actually entails replacing each edge by the five edges in the top or bottom of fig. 5.34 (a) and, of course, adding the connections that pass through the z vertices as well. A given literal l_m may appear in several clauses (for example, $\neg x_3$ in fig. 5.36), and thus an edge e_m or $\overline{e}_m$ may be influenced by several A widgets (for example, edge $\overline{e}_3$). In this case, we connect the A widgets in series, as shown in fig. 5.37, effectively replacing edge e_m or $\overline{e}_m$ by a series of edges.



(a) A Portion of Fig. 5.36 *(b) The actual Subgraph Constructed*

Fig. 5.37

We claim that formula ϕ is satisfiable if and only if graph G contains a Hamiltonian cycle. We first suppose that G has a Hamiltonian cycle h and show that ϕ is satisfiable. Cycle h must take a particular form –

(i) First, it traverses edge $(b_{1,1}, x'_1)$ to go from the top left to the top right

(ii) It then follows all of the x'_m and x''_m vertices from top to bottom, choosing either edge e_m or edge $\overline{e}_m$, but not both

(iii) It next traverses edge $(b_{k,4}, x''_n)$ to get back to the left side

(iv) Finally, it traverses the B widgets from bottom to top on the left.

Given the Hamiltonian cycle h, we define a truth assignment for ϕ as follows. If edge e_m belongs to h, then we set $x_m = 1$. Otherwise, edge $\overline{e}_m$ belongs to h, and we set $x_m = 0$.

We claim that this assignment satisfies ϕ. Consider a clause C_i and the corresponding B widget in G. Each edge $(b_{i,j}, b_{i,j+1})$ is connected by an A widget to either edge e_m or edge $\overline{e}_m$, depending on whether x_m or $\neg x_m$ is the jth literal in the clause. The edge $(b_{i,j}, b_{i,j+1})$ is traversed by h if and only if the corresponding literal is 0. Since each of the three edges $(b_{i,1}, b_{i,2})$, $(b_{i,2}, b_{i,3})$, $(b_{i,3}, b_{i,4})$ in clause C_i is also in a B widget, all three cannot be traversed by the Hamiltonian cycle h. One of the three edges, therefore, must have a corresponding literal whose assigned value is 1, and clause C_i is satisfied. This properly holds for each clause C_i, $i = 1, 2, ..., k$, and thus formula ϕ is satisfied.

Conversely, let us suppose that formula ϕ is satisfied by some truth assignment. By following the rules from above, we can construct a Hamiltonian cycle for graph G – traverse edge e_m if $x_m = 1$, traverse edge $\overline{e}_m$ if $x_m = 0$, and traverse edge $(b_{i,j}, b_{i,j+1})$ if and only if the jth literal of clause C_j is 0 under the assignment. These rules can indeed be followed, since we assume that s is a satisfying assignment for formula ϕ.

Finally, we note that graph G can be constructed in polynomial time. It contains one B widget for each of the k clauses in ϕ. There is one A widget for each instance of each literal in ϕ, and so there are $3\,k$ A widgets. Since the A and B widgets are of fixed size, the graph G has $O(K)$ vertices and edges and is easily constructed in polynomial time. Thus, we have provided a polynomial-time reduction from 3-CNF-SAT to HAM-CYCLE.

Q.51. Explain vertex cover problem and Hamiltonian path problem. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.47 and Q.50.

Q.52. Discuss travelling salesman problem. ***(R.G.P.V., Dec. 2011)***

Or

Explain travelling salesman problem.(R.G.P.V., June 2010, May 2018)

Or

Show that the travelling salesman problem is NP-complete. ***(R.G.P.V., Dec. 2010, 2013)***

Or

Discuss and explain travelling salesman problem. (R.G.P.V., Dec. 2015)

Ans. The travelling salesman problem is – Given n cities, the distance between them and a number D, does there exist a tour programme for a salesman to visit all the cities so that the total distance travelled is atmost D ?

Proof – We first show that TSP (travelling-salesman problem) belongs to *NP*. Given an instance of the problem, we use as a certificate the sequence of n vertices in the tour. The verification algorithm checks that this sequence contains each vertex exactly once, sums up the edge costs, and checks whether the sum is at most k. This process can certainly be done in polynomial time.

To prove that TSP is *NP*-hard, we show that HAM-CYCLE $\leq_P$ TSP. Let $G = (V, E)$ be an instance of HAM-CYCLE. We construct an instance of TSP as follows. We form the complete graph $G' = (V, E')$, where $E' = \{(i,j): i,j \in V\}$, and we define the cost function c by

$$c(i, j) = \begin{cases} 0 & \text{if}(i, j) \in E, \\ 1 & \text{if}(i, j) \notin E. \end{cases}$$

The instance of TSP is then $(G', c, 0)$, which is easily formed in polynomial time.

We now show that graph G has a Hamiltonian cycle if and only if graph G' has a tour of cost at most 0. Suppose that graph G has a Hamiltonian cycle h. Each edge in h belongs to E and thus has cost 0 in G'. Thus, h is a tour in G' with cost 0. Conversely, suppose that graph G' has a tour h' of cost at most 0. Since the costs of the edges in E' are 0 and 1, the cost of tour h' is exactly 0. Therefore, h' contains only edges in E. We conclude that h is a Hamiltonian cycle in graph G.

Q.53. What is NP-complete problems ? Show that travelling salesman problem is NP-complete. ***(R.G.P.V., Dec. 2012)***

***Ans. NP*-complete Problems** – Refer to Q.37.

Travelling Salesman Problem – Refer to Q.52.

Q.54. Write short notes on –

(i) NP-hard problem (ii) Travelling salesman problem. ***(R.G.P.V., Dec. 2014)***

Ans. (i) NP-hard Problem – Refer to Q.37.

(ii) Travelling Salesman Problem – Refer to Q.52.

Q.55. What is the partition problem ?

Or

Discuss partition problem. ***(R.G.P.V., June 2011)***

Or

Explain partition problem. ***(R.G.P.V., May 2018)***

Ans. Given a list of integers $i_1, i_2 \ldots\ldots i_k$, does there exist a subset whose sum is exactly $1/2\ (i_1 + i_2 + \ldots. + i_k)$. Note that this problem appears to be in P until we remember that the length of an instance is not $i_1 + i_2 + \ldots\ldots + i_k$ but the sum of the lengths of the i_j's written in binary or some other fixed base.

Q.56. Explain the properties of recursive and recursive enumerable languages. ***(R.G.P.V., Dec. 2011, 2012)***

Or

Write a short note on recursive and recursively enumerable languages. ***(R.G.P.V., June 2004, 2017)***

Or

Discuss the properties of recursive and recursive enumerable languages. ***(R.G.P.V., June 2015)***

Or

Briefly describe the recursively enumerable and recursive languages. ***(R.G.P.V., Nov. 2018)***

Ans. **Recursively Enumerable Languages** – A language L is said to be ***recursively enumerable,*** if there exists a turing machine that accepts L.

This definition implies only that there exists a turing machine M, such that, for every $u \in L$.

$$q_0 u \vdash^*_M x_1\, q_f x_2,$$

with q_f a final state. The definition does not say anything about what happens for u not in L; it may be that the machine halts in a nonfinal state or that it never halts and goes into an infinite loop.

Recursive Languages – A language L on Σ is said to be recursive, if there exists a turing machine M that accepts L and that halts on every $u \in \Sigma^+$. In other words, we can say that a language is recursive iff there exists a membership algorithm for it or if there exist a TM that recognizes L.

If a langauge is recursive, then there exists an easily constructedM enumeration procedure. Suppose that M is a turing machine that determines membership in a recursive language L. First, we construct another turing machine, say M', that generates all strings in Σ^+, in proper order, say $u_1, u_2, \ldots$ As these strings are generated, they become the input to M, which is modified so that it writes strings on its tape only if they are in L.

There is also an enumeration procedure for every recursively enumerable langauge, but it is not easy to see. It is easy to see that every language for which an enumeration procedure exists is recursively enumerable. We simply compare the given input string against successive strings generated by the enumeraton procedure. If $u \in L$, then we will eventually find a match, and the process can be terminated.

Q.57. Explain the term recursively enumerable language. ***(R.G.P.V., Dec. 2014)***

Or

Explain recursively enumerable languages. ***(R.G.P.V., June 2009)***

Or

Write short note on recursively enumerable languages.
(R.G.P.V., June 2003, Dec. 2005, 2007, 2017)

Ans. Refer to Q.56.

Q.58. What do you understand by recursive enumerable languages ? Explain the turing machine with its properties. ***(R.G.P.V., Nov. 2019)***

Ans. Refer to Q.56 and Q.2.

Q.59. Prove that universal language is recursively enumerable.
(R.G.P.V., Dec. 2007)

Ans. We shall exhibit a three-tape TM M_1 accepting universal language L_u. The first tape of M_1 is the input tape, and the input head on that tape is used to look up moves of the TM M when given code $<M, w>$ as input. Note that the moves of M are found between the first two blocks of three 1's. The second tape of M_1 will simulate the tape of M.

The alphabet of M is $\{0, 1, B\}$, so each symbol of M's tape can be held in one tape cell of M_1's second tape. Observe that if we did not restrict the alphabet of M, we would have to use many cells of M_1's tape to simulate one of M's cells, but the simulation could be carried out with a little more work. The third tape holds the state of M, with q_i represented by 0^i. The behaviour of M_1 is as follows –

(i) Check the format of tape 1 to see that it has a prefix and that there are no two codes that begin with 0^i10^j1 for the same i and j.

(ii) Initialize tape 2 to contain w, the portion of the input beyond the second block of three 1's. Initialize tape 3 to hold a single 0, representing q_1. All three tape heads are positioned on the leftmost symbols.

(iii) If tape 3 holds 00, the code for the final state, halt and accept.

(iv) Let X_j be the symbol currently scanned by tape head 2 and let 0^i be the current contents of tape 3. Scan tape 1 from the left end to the second 111, looking for a substring beginning 110^i10^j1. If no such string is found, halt and reject. M has no next move and has not accepted. If such a code is found, let it be $0^i10^j10^k10^l10^m$. Then put 0^k on tape 3, print X_l on the tape cell scanned by head 2 and move that head in direction D_m.

It is straightforward to check that M_1 accepts $<M, w>$ if and only if M accepts w. It is also true that if M runs forever on w, M_1 will run forever on $<M, w>$, and if M halts on w without accepting, M_1 does the same on $<M, w>$.

Q.60. What do you mean by recursive language ? Prove that complement of a recursive language is recursive. ***(R.G.P.V., Dec. 2013)***

Ans. **Recursive Language** – Refer to Q.56.

Proof – Let L be a recursive language and M be a turing machine that halts on all inputs and accepts L. Let us construct M' from M so that if M enters a final state on input u, then M' halts without accepting. If M halts without accepting, M' enters a final state. Since one of these two events occurs, M' is an algorithm. Clearly $L(M')$ is the complement of L and thus the complement of L is a recursive language.

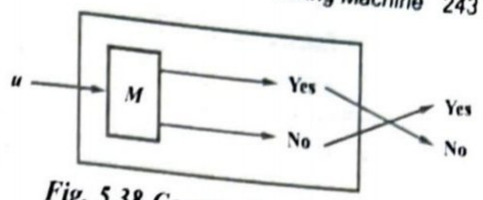


Fig. 5.38 Construction Showing that Recursive Languages are Closed under Complementation

The construction of M' from M is given in fig. 5.38.

Q.61. Prove that the if a language L and its complement $\overline{L}$ are both recursively enumerable, then both languages are recursive. If L is recursive, the $\overline{L}$ is also recursive, and consequently both are recursively enumerable.

Ans. **Proof** – If L and $\overline{L}$ are both recursively enumerable, then there exist Turing Machines M and $\hat{M}$ that serve as enumeration procedures for L and $\overline{L}$, respectively. The first will produce w_1, w_2 ..... in L, the second $\hat{w}_1, \hat{w}_2$..... in $\overline{L}$. Suppose now we are given any $w \in \Sigma^+$. We first Let M generate w_1 and compare it with w. If they are not the same, we let $\hat{M}$ generate $\hat{w}_1$ and compare again. If we need to continue, we next let M generate w_2, then $\hat{M}$ generate $\hat{w}_2$, and so on. Any $w \in \Sigma^+$ will be generated either by M or $\hat{M}$, so eventually we will get a match. If the matching string is produced by M, w belongs to L, otherwise it is in $\overline{L}$. The process is a membership algorithm for both L and $\overline{L}$, so they are both recursive.

For the converse, assume that L is recursive. Then there exists a membership algorithm for it. But this becomes a membership algorithm for $\overline{L}$ by simply complementing its conclusion. Therefore $\overline{L}$ is recursive. Since any recursive language is recursively enumerable.

Q.62. Describe the recursively enumerable language with example.
(R.G.P.V., May 2018)

Ans. Refer to Q.56 and Q.61.

Q.63. Give two properties of recursively enumerable set.
(R.G.P.V., Dec. 2016)

Ans. (i) Union of Two Recursively Enumerable – To prove the union of two recursively enumerable languages is recursively enumerable, let us consider L_1 and L_2 be two recursively enumerable languages accepted by algorithms M_1 and M_2.

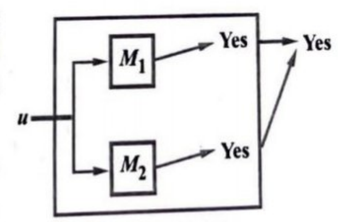


Fig. 5.39

We construct M, which simulate simultaneously M_1 and M_2 on separate tapes. If either accepts, then M accepts. The construction is shown in fig. 5.39.

(ii) Intersection of Two Recursively Enumerable – Suppose that $T_1 = (Q_1, \Sigma, \Gamma_1, q_1, \delta_1)$ and $T_2 = (Q_2, \Sigma, \Gamma_2, q_2, \delta_2)$ are TMs accepting L_1 and L_2, respectively. We wish to construct TMs accepting $L_1 \cap L_2$.

The two-tape machine $T = (Q, \Sigma, \Gamma, q_0, \delta)$ begins by copying the input string x onto tape 2 and inserting inserting the marker # at the beginning of each tape. The simultaneous simulation of T_1 on tape 1 and T_2 on tape 2 is accomplished by allowing every possible move

$$\delta((p_1, p_2), (a_1, a_2)) = ((q_1, q_2), (b_1, b_2), (D_1, D_2))$$

where, for both values of i, $\delta_i(p_i, a_i) = (q_i, b_i, D_i)$. The possible outcomes of the simulation are –

(a)Neither T_1 nor T_2 ever stops, in which case T never stops

(b) At least one of the two halts, in which case T halts

(c) Machine can crash as soon as either T_1 or T_2 crashes, and it can halt only after both T_1 and T_2 have halted. Hence, we may conclude that T accepts the language $L_1 \cap L_2$.

Q.64. Find the languages obtained from the following operations –

(i) Union of two recursive languages

(ii) Union of two recursively enumerable languages

(iii) If L and complement of L are recursively enumerable.

(R.G.P.V., Dec. 2014, June 2015)

Ans. (i) Proof – To prove the union of two recursive languages is recursive, let us conisder L_1 and L_2 be two recursive languages accepted by algorithms M_1 and M_2. We construct M, which first simulates M_1. If M_1 accepts, then M accepts. If M_1 rejects then M simulates M_2 and accepts iff M_2 accepts. Since both M_1 and M_2 are algorithms, M is guaranteed to halt. Clearly, M accepts $L_1 \cup L_2$. The construction of M is shown in fig. 5.40.

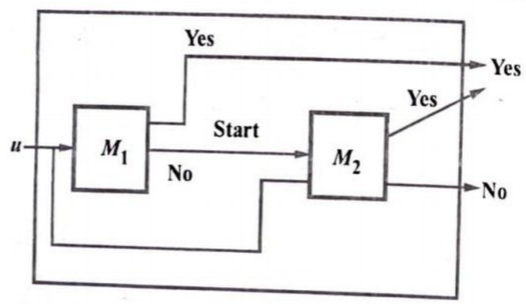


Fig. 5.40 Construction of Union

(ii) Proof – To prove L is recursive, if L and its complement $\overline{L}$ both are recursively enumerable, let us consider M_1 and M_2 accept L and $\overline{L}$, respectively.

We construct M, as given in fig. 5.41 to simulate simultaneously M_1 and M_2. M accepts u if M_1 accepts u and rejects u if M_2 accept u. Since u is in either L or $\overline{L}$, we know that exactly one of M_1 or M_2 will accept. Thus M will always say either "yes" or "no", but will never say both.

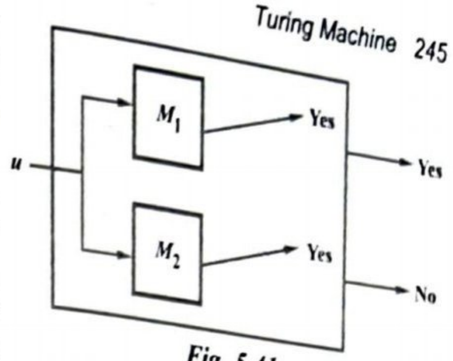


Fig. 5.41

It is noted that there is no priori limit on how long it may take before M_1 or M_2 accepts, but it is certain that one or the other will do so. Since M is an algorithm that accepts L, it follows that L is recursive.

(iii) Refer to Q.63 (i).

Q.65. Prove that every recursive language is recursively enumerable language.

(R.G.P.V., June 2008)

***Ans.* Proof** – Every language that is recursive is recursively enumerable. If T is a TM recognizing L, then a modified machine that crashes instead of leaving output 0 accepts L. If T is a TM accepting L, there may strings not in L for which T crashes and therefore never produces an answer.

If a language is recursive, then there exists an enumeration procedure. Suppose that M is a Turing machine that determines membership in a recursive language L. We first construct another turing machine, say $\hat{M}$, that generates all strings in Σ^+ in proper order, let us say $w_1, w_2, \ldots$ As these strings are generated, they become the input to M, which is modified so that it writes strings on its tape only if they are in L.

That there is also an enumeration procedure for every recursively enumerable language is not as easy to see. We cannot use the above argument as it stands, because if some w_j is not in L, the machine M, when started with w_j on its tape, may never halt and therefore never get to the strings in L that follow w_j in the enumeration. To make sure that this does not happen, the computation is performed in a different way. We first get $\hat{M}$ to generate w_1 and let M execute one move on it. Then we let $\hat{M}$ generate w_2 and let M execute one move on w_2, followed by the second move on w_2, the third step on w_1, and so on. From this it is clear that M will never get into an infinite loop. Since any $w \in L$ is generated by $\hat{M}$ and accepted by M in a finite number of steps, every string in L is eventually produced by M.

It is easy to see that every language for which an enumeration procedure exists is recursively enumerable. We simply compare the given input string against successive strings generated by the enumeration procedure. If $w \in L$, we will eventually get a match, and the process can be terminated.

Q.66. Prove that, there exists a recursively enumerable language whose complement is not a recursively enumerable.

Ans. Suppose $\Sigma = \{x\}$, and consider the set of all turing machines with this input alphabet. This set is countable, so we can associate an order M_1,

M_2,.....with its elements. For each turing machine M_i, there is an associated recursively enumerable language $L(M_i)$. Conversely, for each recursively enumerable language on Σ, there is some turing machine that accepts it.

Now, we consider a new language L defined as, for each $i \geq 1$, the string x^i is in L if and only if $x^i \in L(M_i)$. Clearly that the language L is well defined, since the statement $x^i \in L(M_i)$, and hence $x^i \in L$, must either be true or false. Next, we consider the complement of 9,

$$\overline{L} = \left\{x^i : x^i \notin L(M_i)\right\}, \quad \text{....(i)}$$

which is also well defined but, as we will show, is not recursively enumerable.

Now, we will show this by contradiction, starting from the assumption that $\overline{L}$ is recursively enumerable, then there must be some turing machine, say M_k, such that –

$$\overline{L} = L(M_k) \quad \text{....(ii)}$$

Consider the string x^k. Is it in L or in $\overline{L}$? Let $x^k \in \overline{L}$. By equation (ii), this implies that –

$$x^k \in L(M_k)$$

But by equation (i) it implies that

$$x^k \notin \overline{L}$$

Conversely, if we assume that x^k is in L, then $x^k \notin \overline{L}$ and equation (ii) implies that –

$$x^k \notin L(M_k)$$

But then from equation (i), we get

$$x^k \in \overline{L}$$

The contradiction is inescapable, and we must conclude that our assumption that $\overline{L}$ is recursively enumerable is false.

For showing the L is recursively enumerable, we can use the known enumeration procedure for turing machines. Given x^i, we first find i by counting the number of x's. Then we use the enumeration procedure for turing machines to find M_i. Finally, we give its descriptions along with x^i to a universal turing machine M_u that simulates the action of M on x^i. If x^i is in L, the computation carried out by M_u will eventually halt. The combined effects of this is a turing machine that accepts every $x^i \in L$. Therefore, by definition, L is recursively enumerable.

Q.67. What are decidable and undecidable problems ?

(R.G.P.V., Dec. 2016)

Ans. **Decidable Problems** – Those problems for which there exist an algorithm for that problem and the result of computation is either "yes" or "no". For example, let us consider a statement –

"A context free grammar is in Chomsky Normal Form". For some context free grammar this is false, for other it is true. If there exist any algorithm

which decides whether the above statement is true or false for any context free grammar then the problem is decidable problem.

Undecidable Problems – A problem is undecidable if there is no algorithm which correctly give a solution to given problem in form of "yes" or "no". Example of some undecidable problems are –

(i) The halting problem of turing machine

(ii) Post correspondence problem.

Q.68. Write short note on halting problem of turing machine.

(R.G.P.V., Nov. 2019)

Ans. A class of problems with two outputs (Yes/No) is said to be solvable (decidable) if there exists some definite algorithm which always terminates (halts) with two outputs (Yes/No). Otherwise the class of problems is said to be unsolvable.

Some examples of unsolvable problems are given below –

(i) To write a program that will search through all positive integers x, y, z and number n > 2 for a solution to the equation $x^n + y^n = z^n$ is easy. But to determine whether or not the program will halt if and when a solution is found, is a problem.

(ii) In any programming language, to determine whether or not

(a) A given program can loop for ever on some input.

(b) A given program ever produces an output.

(c) A given program eventually halts on the given input.

(iii) The halting problem of turing machine.

Now we shall prove that halting problem of a machine is unsolvable –

According to Church's thesis, a turing machine can be treated as the most general computing system. So finding an algorithm for Yes/No problem is equivalent to constructing a turing machine which can execute the program.

Theorem – The halting problem (H.P.) of turing machine over $\Sigma = \{0, 1\}$ is unsolvable (i.e., the problem of determining whether or not an arbitrary turing machine M over {0, 1} halts for an arbitrary input x in Σ^* is unsolvable).

Proof – We shall prove this theorem by contradiction. Let M be an arbitrary turing machine. Let d(M) be the encoded binary string representing M. Then the machine-string pair (M, x) will have d(M)*x as its encoded description. We assumed that H.P. is solvable. Hence there exists an algorithm P which decides the HP –

(a) If M halts for input x, then P reaches an accept halt.

(b) If M does not halt for input x, then P reaches a reject halt.

Let us construct a new algorithm Q based on P as follows –

(c) It takes d(M) as input and copies it to obtain d(M)*d(M) and then applies algorithm P to this input.

(d) Q loops for ever if P reaches an accept halt and Q halts if P reaches a reject halt.

By Church's thesis, there exists a turing machine say M', which can execute the algorithm Q. Since the algorithm P, similar to Q, works for an arbitrary machine M, so Q also works for the turing machine M'. So we can say that M = M'.

From (d) and (a) we can conclude that M' loops for ever if M' halts. From (d) and (b) we conclude that M' halts if M' loops for ever. Hence a conclusion is obtained that "M' halts if and only if M' loops". This is a contradiction and therefore, HP is unsolvable.

Even if the halting problem is unsolvable, we can run the machine with a given input. If it halts, then a specific case is answered. If it does not halt, it may be possible to conclude that it will never halt by detecting some pattern within the operation.

Q.69. Give short note on Post's correspondence problem.

(R.G.P.V., Dec. 2014)

Ans. The undecidability of the halting problem has many consequences of practical interest, particularly in the area of CFL. But in many instances it is cumbersome to work with the halting problem directly, and it is convenient to establish some intermediate results that bridge the gap between the halting problem and other problems. These intermediate results follow from the undecidability of the halting problem, but are more closely related to the problems we want to study and therefore make the augments easier. One such intermediate result is the post correspondence problem.

The post correspondence problem can be stated as follows. Given two sequence of n strings on some alphabet Σ say

$$A = w_1, w_2, \ldots, w_n$$

and $$B = v_1, v_2, \ldots, v_n$$

We say that there exist a post correspondence solution for pair (A, B) if there is a non-empty sequence of integers $i, j, \ldots, k$ such that

$$w_i w_j \ldots w_k = v_i v_j \ldots v_k$$

The post correspondence problem is to drive an algorithm that tell us, for any (A, B), whether or not there exists a post correspondence solution.

●●●

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2009
(New Scheme)
(Computer Science & Engg. Branch)
THEORY OF COMPUTATION
[CS–505(N)]

Note : Attempt *one* question from each Unit. All questions carry equal marks.

Unit–I

1. (a) Construct an NFA with $\in$ moves, corresponding to the regular expression a*b (a | bb)*, convert it to a DFA and minimize it. 10

(See Unit-II, Page 84, Prob.27)

(b) Construct a Moore machine equivalent to Mealy machine M given below –

Present State	Next State			
	a = 0		a = 1	
	State	Output	State	Output
q_1	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

(See Unit-I, Page 34, Prob.26) 10

2. (a) Construct DFA for – 10

(i) Binary integer divisible by 3

(ii) $a^n b \mid n \geq 0$.

(See Unit-I, Page 18, Prob.8)

(b) Construct a Mealy machine which is equivalent to the Moore machine given below –

Present State	Next State		Output
	a = 0	a = 1	
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

(See Unit-I, Page 31, Prob.22) 10

Unit-II

3. (a) Write CFG for the following sets – 10

(i) $L = \{a^n b^m c^k : n = m \text{ or } m \leq k, n \geq 0, m \geq 0, k \geq 0\}$

(ii) $L = \{a^{2n} b c \mid n \geq 1\}$.

(See Unit-III, Page 114, Prob.6)

(b) Reduce the following grammars to Greibach normal form – 10

(i) $S \to AO, A \to OB, B \to AO, B \to 1$

(ii) $S \to SS, S \to OS\ 1101$

Or

4. (a) Show that the following two grammars are equivalent – 5

Grammar-1	Grammar-2
$S \to abAB \mid ba$	$S \to abA\ aA \mid abAbb \mid ba$
$A \to aaa$	
$B \to aA \mid bb$	$A \to aaa$

(See Unit-III, Page 134, Prob.39)

(b) Show that $L = \{a^P \mid P \text{ is prime}\}$ is not a context-free language. 5

**

(c) Convert the following CFG to Greibach normal form (GNF) – 10

$S \to AB \mid 0$

$A \to BC \mid 1$

$B \to CD \mid 2$

$C \to AD \mid 0$

$D \to 1$

(See Unit-III, Page 146, Prob.55)

Unit-III

5. (a) For the given CFG, construct PDA that accepts the same language they generate – 10

$S \to Saa \mid aSa \mid aaS$ (See Unit-IV Page 175, Prob.19)

(b) Differentiate between "PDA by empty stack" and "PDA by final state". (See Unit-IV, Page 153, Q.5) 10

Or

6. (a) Write a brief note on 2-way PDA. (See Unit-IV, Page 159, Q.13) 10

(b) Using pumping lemma show that the following set are not regular –

(i) $\{a^n b^{2n} \mid n > 0\}$

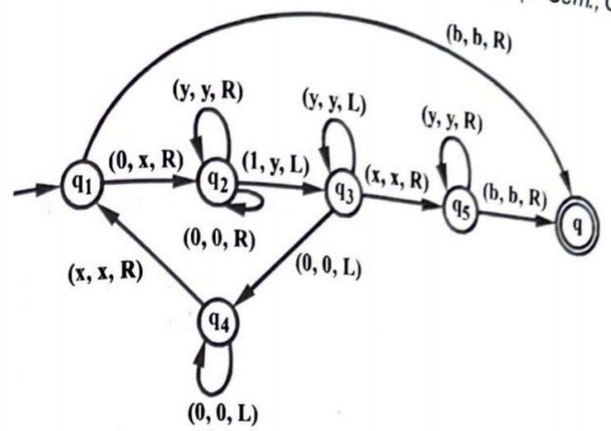
(ii) $\{a^n b^m \mid 0 < n < m\}$.

(See Unit-II, Page 96, Prob.41) 10

Unit-IV

7. (a) M is a turing machine represented by the transition system ahead. Obtain computation sequence of M for processing the input string 0011 – 10

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus



(Transition system M)

(b) Write a note on universal turing machine. (See Unit-V, Page 205, Prob.1) (See Unit-V, Page 199, Q.22) 10

Or

8. (a) Design a turing machine to recognize all strings consisting of even number of 1's. (See Unit-V, Page 206, Prob.2) 10

(b) Write brief notes on the following : 10

(i) Two-way infinite tape TM (See Unit-V, Page 200, Q.25)

(ii) Design of TM. (See Unit-V, Page 193, Q.12)

Unit-V

9. (a) Discuss satisfiability problem. Prove that satisfiability is NP complete. (See Unit-V, Page 232, Q.45) 10

(b) What are NP hard problems ? Write a brief note on it. 10

(See Unit-V, Page 230, Q.41)

Or

10. (a) Write a brief note on untractable problems. ** 10

(b) What is Hamilton circuit problem ? Prove that directed Hamilton circuit is NP-complete. (See Unit-V, Page 234, Q.50) 10

RGPV

B.E. (Fifth Semester) EXAMINATION, June 2010
(New Scheme)
(Computer Science & Engg. Branch)
THEORY OF COMPUTATION
[CS–505(N)]

Note : (i) Question paper is divided into five Units. There is internal choice in each Unit.

(ii) Attempt one question from each Unit.

(iii) All questions carry equal marks.

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

Unit–I

1. (a) Prove by induction –

$$\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$$ (See Unit-II, Page 100, Prob.49)

(b) Construct DFA over input alphabet $\Sigma = \{0, 1\}$ to accept string which contains no. of 0 is even and no. of 1 is even. (See Unit-I, Page 22, Prob.12)

(c) State and prove pumping lemma for regular set. (See Unit-II, Page 70, Q.30)

Or

(a) Construct an NFA with $\in$-move corresponding to the R.E. $(a+b)^*abb$, convert it to a DFA and minimize it. (See Unit-II, Page 82, Prob.26)

(b) Formally define the following (with example) :
(i) Mealy machine (ii) Moore machine. (See Unit-I, Page 10, Q.11)

Unit-II

2. (a) Construct a CFG generating all integers (with sign). (See Unit-III, Page 112, Prob.1)

(b) Show that the following grammar is ambiguous :
$E \rightarrow E + E|E*E|2|3|4$ (See Unit-III, Page 126, Prob.28)

(c) Define CNF and GNF. (See Unit-III, Page 109, Q.18)

Or

(a) Convert the following CFG into CNF :
$S \rightarrow \sim S \mid [S \supset S] \mid p/q$ (See Unit-III, Page 136, Prob.43)

(b) Explain rule for simplication of CFG. (See Unit-III, Page 107, Q.14)

(c) Explain leftmost derivation, rightmost derivation and parse with suitable example. (See Unit-III, Page 105, Q.11)

Unit-III

3. (a) State and prove pumping lemma for CFG.

(b) Construct a PDA equivalent to the following grammar –

$$S \rightarrow aAA$$
$$A \rightarrow aS|bS|a$$

(See Unit-IV, Page 177, Prob.22)

(c) Deterministic and non deterministic models of PDA are not equivalent. Justify with example. (See Unit-IV, Page 158, Q.11)

Or

(a) Construct PDA for the following set –
$L = \{0^n1^n2^n | n \geq 1\}$

(b) Show the procedure of converting a PDA into equivalent CFG using suitable example. (See Unit-IV, Page 159, Q.14)

Unit-IV

4. (a) Turing machine models the computing capacity of a general purpose

computer. Justify with complete example. (See Unit-V, Page 187, Q.5)

(b) Design turing machine to accept –
$L = \{a^nb^n | n \geq 1\}$ (See Unit-V, Page 208, Prob.7)

Or

(a) Discuss how turing machine can be designed to –
(i) n! (ii) $\log_2 n$

(b) Give definition of a turing machine. What type of language can it accept ? Types of turing machine. (See Unit-V, Page 201, Q.27)

Unit-V

5. (a) What do you understand by P, NP, NP complete and NP hard problems. (See Unit-V, Page 229, Q.39)

(b) Explain travelling salesman problem. (See Unit-V, Page 239, Q.52)

Or

Write short notes on any *four* of the following –

(a) Equivalent of DFA and NDFA (See Unit-II, Page 40, Q.7)
(b) Myhill-Nerode theorem (See Unit-II, Page 68, Q.27)
(c) Two-way finite automata (See Unit-II, Page 74, Q.35)
(d) Chomsky classification of grammar (See Unit-III, Page 101, Q.1)
(e) Regular expression (See Unit-II, Page 60, Q.18)
(f) Automata. (See Unit-I, Page 3, Q.2)

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2010
(New Scheme)
(Computer Science & Engg. Branch)
THEORY OF COMPUTATION
[CS–505(N)]

Note : The question paper is divided into five Units. Each Unit carries an internal choice. Attempt one question from each Unit. Thus attempt five questions in all. All questions carry equal marks. Assume suitable data whenever necessary.

Unit-I

1. (i) Find the regular grammar for the language – 10
(a) $L = \{w \in \{a, b\}^* : w \text{ does not contain the substring } aa\}$
(b) $L = \{a^nb^n : n \geq 0\}$
(See Unit-III, Page 117, Prob.16)

(ii) Regular expression $(aa^*)|(bb^*)$ is given. Construct NFA for the expression and convert this NFA to DFA. (See Unit-II, Page 86, Prob.28) 10

Or

2. (i) Prove that $\{a^nba^mba^{m+n} : n, m \geq 1\}$ is not regular. 8

(ii) Construct a finite automation that will accept those strings of a binary number that are divisible by three. (See Unit-I, Page 19, Prob.9) 7

(iii) Construct a Mealy machine which can output EVEN, ODD according as the total number of 1's encountered is even or odd. The input symbols are 0 or 1. (See Unit-I, Page 25, Prob.17) 5

Unit-II

3. (i) Construct a context-free grammar for the languages –

$L(G_1) = \{a^i b^{2i}, i > 0\}$ and $L(G_2) = \{a^n b\, a^n, n > 0\}$ 10

(See Unit-III, Page 115, Prob.10)

(ii) What do you mean by ambiguity ? Show that the grammar 10

$S \to aB \mid ab$
$A \to a$
$B \to C|b$
$C \to D$
$D \to E$
$E \to a$

is ambiguous. (See Unit-III, Page 127, Prob.29)

Or

4. (i) Find a Greibach normal form equivalent to the following CFG – 10

$S \to AB \mid a$
$A \to BS \mid b$
$B \to SA \mid c$

(See Unit-III, Page 147, Prob.56)

(ii) The language is –

$L = \{a^n b^n\} \cup \{a^n b^{2n}\}, n \geq 0$

Show that –

(a) L is context-free

(b) L is not deterministic context-free.

Unit-III

5. (i) Use closure under union to show that the following languages are context free – 10

(a) $\{a^m b^n : m \neq n\}$

(b) $\{a, b\}^* - \{a^n b^n : n \geq 0\}$

(c) $\{w \in \{a, b\}^* : w = w^R\}$

(ii) Construct a PDA accepted empty stack in the language – 10

(a) $a^n b^{2n} \mid n \geq 1$ (See Unit-IV, Page 167, Prob.4)

(b) $a^{2n} b^n \mid n \geq 1$ (See Unit-IV, Page 168, Prob.5)

Or

6. (i) Prove that the class of language accepted by PDA is exactly class of CFG, hence construct PDA for the following grammar : 10

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

What language is represented by it ? (See Unit-IV, Page 162, Q.16)

(ii) State and prove pumping lemma for CFL's. Also prove that $\{a^n b^m a^n b^m : n, m \geq 1\}$ is not context-free. ** 10

Unit-IV

7. (i) All natural numbers are either even or odd. Construct a Turing machine that computes the parity function for natural numbers, that is, that computes – 20

$$F(n) = \begin{cases} 0, & \text{if n is even} \\ 1, & \text{if n is odd} \end{cases}$$

(See Unit-V, Page 215, Prob.15)

(ii) Design a Turing machine that can compute proper subtraction i.e. ... m_n where m and n are positive integers – 10

m_n = m – n if m > n

m_n = 0 if m < = n

(See Unit-V, Page 225, Prob.26)

Or

8. (i) If L_1 is recursive and L_2 is recursive enumerable, show that $L_1 - L_2$ is recursively enumerable. Is $L_2 - L_1$ recursive ? Is it recursively enumerable ? 10

(ii) Design a Turing machine M to recognize the language $\{ww : w \in \{a, b\}^*\}$. (See Unit-V, Page 220, Prob.21) 10

Unit-V

9. (i) Define the following – 10

(a) Polynomial problems

(b) Non-deterministic polynomial (NP) problems

(c) NP complete problems

(d) NP hard problems.

(See Unit-V, Page 229, Q.39)

(ii) Show that travelling salesman problem is NP-complete. 10

(See Unit-V, Page 239, Q.52)

Or

10. (i) Define the following –

(a) Polynomial time reduction and NP-Completeness

(b) Cook's theorem

(See Unit-V, Page 230, Q.42)

(ii) The set of all languages whose complements are in NP is called CO – NP. Prove that NP = CO-NP if and only if there is some NP-complete problem whose complement is in NP. 10

(See Unit-V, Page 228, Q.36)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

RGPV

B.E. (Fifth Semester) EXAMINATION, June 2011
(Computer Science & Engg. Branch)
THEORY OF COMPUTATION
[CS–505(N)]

Note : Attempt any *five* questions. All questions carry equal marks. Attempt *one* question from each Unit.

Unit-I

1. (a) Show that the set $L = \{b^{i^2} / i \geq 1\}$ is not regular. 10
(See Unit-II, Page 95, Prob.38)

(b) Construct a minimum state automaton for the following DFA – 10

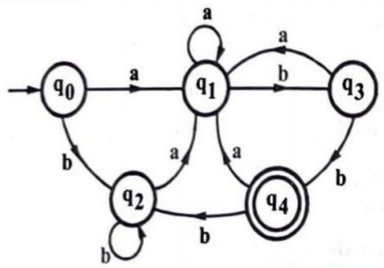


Fig. 1 (See Unit-II, Page 56, Prob.15)

Or

2. (a) Convert the following NFA with $\in$ to equivalent DFA. 10

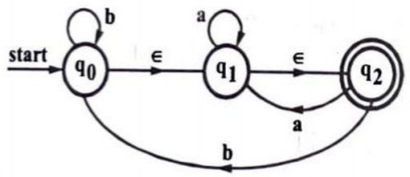


Fig. 2 (See Unit-II, Page 44, Prob.2)

(b) Explain Myhill-Nerode theorem. (See Unit-II, Page 68, Q.27) 10

Unit-II

3. (a) Remove the $\in$ production from the following CFG : 10

$S \rightarrow XYX$
$X \rightarrow OX / \in$
$Y \rightarrow 1Y / \in$

(See Unit-III, Page 128, Prob.31)

(b) Show that the grammar $S \rightarrow a/Sa/bSS/SSb/SbS$ is ambiguous. 10
(See Unit-III, Page 125, Prob.25)

Or

4. (a) Explain LMD and RMD with example.
(See Unit-III, Page 104, Q.10) 10

(b) What do you mean by context free ? Explain the types of grammar.
(See Unit-III, Page 103, Q.4) 10

Unit-III

5. (a) Design PDA for language $\{L = a^{2n} b^n | n \geq 1\}$. 10
(See Unit-IV, Page 168, Prob.5)

(b) Obtain CFG for the PDA given as below – 10

$A = (\{q_0, q_1\}, \{0, 1\}, \{A, z\}, \delta, z, \{q_1\})$

where δ is as given below :

$\delta(q_0, 0, z) = (q_0, Az)$
$\delta(q_0, 1, A) = (q_0, AA)$
$\delta(q_0, 0, A) = (q_1, \in)$

(See Unit-IV, Page 181, Prob.29)

Or

6. (a) Design DPDA for $L = \{0^n 1^m 0^n | m, n \geq 1\}$. 10
(See Unit-IV, Page 170, Prob.11)

(b) Explain *three* closure properties of CFL. ** 10

Unit-IV

7. (a) Differentiate between composite and iterated TM. 10
(See Unit-V, Page 203, Q.30)

(b) Explain Church hypothesis. (See Unit-V, Page 201, Q.28) 10

Or

8. (a) Explain TM and its parameter. (See Unit-V, Page 186, Q.3) 10

(b) Show that the language $L = \{a^i b^j c^k / i < j \text{ and } j < k\}$ is not context free language. ** 10

Unit-V

9. (a) What is NP hard problem ? (See Unit-V, Page 230, Q.41) 10
Write the name of problem and explain with example.

(b) Discuss partition problem. (See Unit-V, Page 240, Q.55) 10

Or

10. (a) Prove that vertex cover is NP complete problem. 10
(See Unit-V, Page 233, Q.48)

(b) What is Hamiltonian path problem ? Discuss with example. 10
(See Unit-V, Page 234, Q.50)

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2011
(Computer Science & Engg. Branch)
THEORY OF COMPUTATION
(CS–505)

Note : Attempt any *five* questions. All questions carry equal marks. Attempt *one* question from each Unit.

Unit-I

1. (a) Design FA which accepts even no. of 0's and even no. of 1's. 10
(See Unit-I, Page 22, Prob.12)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

(b) Find the regular expression accepted by following deterministic finite automation. 10

Or

2. (a) Prove that $a^n \subset b^n | n \geq 1$ is not regular. (See Unit-II, Page 99, Prob.48) 10

(b) Check whether the given grammar is ambiguous or not – 10

$S \rightarrow iC + S$
$S \rightarrow iC + ScS$
$S \rightarrow a$
$C \rightarrow b$

(See Unit-III, Page 128, Prob.30)

Unit-II

3. (a) Reduce the grammar into CNF and GNF : 10

$G = (\{S, A, B\}, \{0, 1\}, \{S \rightarrow |A| 0B, A \rightarrow 0S | 0, B \rightarrow 1S | 1, S)$.

(See Unit-III, Page 143, Prob.51)

(b) How many types of grammar are there ? Explain each of them. 10

(See Unit-III, Page 101, Q.1)

Or

4. (a) Define the following : 10

(i) Chomsky normal form
(ii) Greibach normal form.

(See Unit-III, Page 109, Q.18)

(b) Convert the following grammar to CNF – 10

$S \rightarrow bA | aB$
$A \rightarrow bAA | aS | a$
$B \rightarrow aBB | bSbb$

(See Unit-III, Page 141, Prob.49)

Unit-III

5. (a) Design PDA for the language : 10

$L = \{w | w \in (a + b)^* \text{ and } n_a(w) > n_b(w)\}$

(See Unit-IV, Page 170, Prob.13)

(b) Derive the string "aabbabba" for leftmost derivation and rightmost derivation using a CFG given by : 10

$S \rightarrow aB | bA$
$A \rightarrow a | aS | bAA$
$B \rightarrow b | bS | aBB$

And also draw the derivation tree.

(See Unit-III, Page 120, Prob.20)

Or

6. (a) Construct PDA for the following grammar – 10

$S \rightarrow AB$

$A \rightarrow CD$
$B \rightarrow b$
$C \rightarrow a$
$D \rightarrow a$

(See Unit-IV, Page 178, Prob.25)

(b) Explain three closure properties of CFL. ** 10

Unit-IV

7. (a) Construct a TM for language of even no. of 1's and even no. of 0's over $\Sigma = \{0, 1\}$. (See Unit-V, Page 216, Prob.16) 10

(b) Explain the properties of recursive and recursive enumerable language. (See Unit-V, Page 241, Q.56) 10

Or

8. (a) Explain multitape and multi-head turing machine. 10

(See Unit-V, Page 198, Q.21)

(b) Design a TM for language $\{L = a^n b^n | n \geq 1\}$. 10

(See Unit-V, Page 208, Prob.7)

Unit-V

9. (a) Explain relation between P, NP, NP hard and NP complete problem with diagram. (See Unit-V, Page 229, Q.40) 10

(b) Explain Cook's theorem. (See Unit-V, Page 230, Q.42) 10

Or

10. (a) Prove that vertex cover is NP complete problem. 10

(See Unit-V, Page 233, Q.48)

(b) Discuss travelling salesman problem. (See Unit-V, Page 239, Q.52) 10

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2012
THEORY OF COMPUTATION
(CS-505)

Note : (i) Attempt *one* question from each Unit.
(ii) All questions carry equal marks.

Unit-I

1. (a) Construct an DFA accepting the set of all string over the alphabet {0, 1}, such that number of 0's divisible by 5 and number of 1's divisible by 3. (See Unit-I, Page 24, Prob.15)

(b) State and prove Myhill-Nerode theorem. (See Unit-II, Page 68, Q.27)

Or

2. (a) Explain pumping lemma for regular sets. Also prove following set is not regular.

$L = \{0^m 1^m 0^{m+n} | m \geq 1 \text{ and } n \geq 1\}$

(See Unit-II, Page 98, Prob.47)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

(b) Convert the following Mealy machine into its equivalent Moore machine –

Present State	Next State			
	a = 0		a = 1	
	State	Output	State	Output
q_0	q_1	n	q_2	n
q_1	q_1	y	q_2	n
q_2	q_1	n	q_2	y

(See Unit-I, Page 32, Prob.24)

Unit-II

3. (a) Convert the following Grammar into Greibach normal form.

$S \to AA \mid 0$

$A \to SS \mid 1$

(See Unit-III, Page 145, Prob.53)

(b) Write CFG for set of all words consisting of an equal number of a's and b's.

For example : a a b b, a b a b, a b b b a a.

(See Unit-III, Page 113, Prob.3)

Or

4. (a) Let G be the grammar.

$S \to aB \mid bA$

$A \to a \mid aS \mid bAA$

$B \to b \mid bs \mid aBB$

For the string a a a b b a b b b a find a

(i) Leftmost Derivation (ii) Rightmost Derivation (iii) Parse tree.

(See Unit-III, Page 121, Prob.22)

(b) Convert the following CFG into CNF

$S \to ABA$

$A \to aA \mid \in$

$B \to bB \mid \in$

(See Unit-III, Page 135, Prob.42)

Unit-III

5. (a) Construct PDA that accepts language

$L = \{WW^R \mid W \text{ in } (0 + 1)^*\}$. (See Unit-IV, Page 172, Prob.16)

(b) Give a grammar for languge N(m) where –

$M = (\{q_0, q_1\}, \{0, 1\}, (X, Z_0\}, \delta, q_0, Z_0, \phi)$

Where δ given by –

$\delta(q_0, 0, Z_0) = (q_0, XZ_0)$, $\delta(q_1, 1, X) = (q_1, \in)$

$\delta(q_0, 0, X) = (q_0, XX)$, $\delta(q_1, \in, X) = (q_1, \in)$

$\delta(q_0, 1, X) = (q_1, \in)$, $\delta(q_1, \in, Z_0) = (q_1, \in)$

(See Unit-IV, Page 180, Prob.28)

Or

6. (a) Explain pumping lemma for CFLs. Show that –

$L = \{a^n b^n c^n \mid n \geq 1\}$ is not a context free language. **

(b) Construct a PDA equivalent to following grammar –

$S \to aAA$

$A \to aS \mid bS \mid a$

(See Unit-IV, Page 177, Prob.22)

Unit-IV

7. (a) Design a turing machine to accept the language –

$L = \{0^n 1^n 0^n \mid n \geq 1\}$ (See Unit-V, Page 212, Prob.10)

(b) Explain properties of recursive and recursively enumerable languages.

(See Unit-V, Page 241, Q.56)

Or

8. (a) Design a Turing machine to compute factorial of a number.

(b) Explain Church's hypothesis. (See Unit-V, Page 201, Q.28)

Unit-V

9. (a) Explain P, NP, NP complete problems with example.

(See Unit-V, Page 229, Q.38)

(b) Write brief note on untractable problems. **

Or

10. (a) What is NP complete problems ? Show that travelling salesman problem is NP complete. (See Unit-V, Page 240, Q.53)

(b) Explain vertex cover problem. (See Unit-V, Page 233, Q.47)

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2013

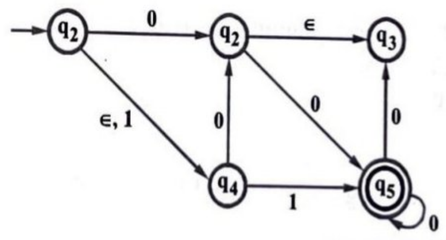
THEORY OF COMPUTATION

(CS-505)

Note : Attempt one question from each unit. All questions carry equal makrs.

Unit-I

1. (a) Consider the FA below and construct the, smallest DFA which accepts the same language. 7



(See Unit-II, Page 51, Prob.11)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

(b) Give Mealy and Moore machine for the input from $(0 + 1)^*$ if the input ends is '000', output A; if the input ends in '111', output B; otherwise output C. (See Unit-I, Page 35, Prob.27) 7

2. (a) Explain pumping lemma for regular sets. Prove by pumping lemma the following set is not regular.
$L = \{W \subset W^R | W \in (0 + 1)^*\}$ 7

(b) Design a DFA that accepts the string such that numer of zero visible by five and number of one divisible by three. 7
(See Unit-I, Page 24, Prob.15)

Unit-II

3. (a) Write the CFG for the following language –
(i) $L = \{0^i 1^j 2^k | i = j \text{ or } j = k\}$ 7
(ii) $L = \{0^n 1^n | n \geq 1\}$ (See Unit-III, Page 113, Prob.2)

(b) Convert the following grammar G into Chomsky normal form. 7
$S \rightarrow ABAC$
$A \rightarrow aA | \in$
$B \rightarrow bB | \in$
$C \rightarrow c$
(See Unit-III, Page 140, Prob.48)

Or

4. (a) Convert the following grammar into GNF. 7
$A_1 \rightarrow A_2A_3$
$A_2 \rightarrow A_3A_1 | b$
$A_3 \rightarrow A_1A_2 | a$
(See Unit-III, Page 148, Prob.57)

(b) Find regular grammars for the following languages on (a, b). 7
(i) $L = \{w | n_a(w) \text{ and } n_b(w) \text{ are both even}\}$
(ii) $L = \{w | (n_a(w) - n_b(w)\} \bmod 3 = 1\}$
(See Unit-III, Page 117, Prob.15)

Unit-III

5. (a) Design a PDA which accepts the language $L = \{W \in (a, b)^*/W$ has the equal number of a's and b's$\}$. (See Unit-IV, Page 166, Prob.1) 7

(b) Explain closure properties of CFL's. ** 7

Or

6. (a) Design PDA corresponding to given CFG – 7
$S \rightarrow aSA$
$S \rightarrow bSb$
$S \rightarrow a$
(See Unit-IV, Page 176, Prob.20)

(b) Explain pumping lemma for CFL. Prove that following language is CFL or not ? 7
$L = \{a^n b^n c^n | n \geq 1\}$

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

Unit-IV

7. (a) What do you mean by recursive language ? Prove that complement of a recursive language is recursive. (See Unit-V, Page 242, Q.60) 7

(b) Build a Turing machine that accepts the language. 7
$L = \{a^n b^{2n}\}$
(See Unit-V, Page 210, Prob.8)

Or

8. (a) Design a turing machine to compute the function $f(m, n) = m + n$ where m and n are non negative numbers. 7
(See Unit-V, Page 224, Prob.25)

(b) What do you mean by turing machine ? Explain multiple tapes turing machine. (See Unit-V, Page 193, Q.15) 7

Unit-V

9. (a) Show that travelling salesman problem is NP-complete.
(See Unit-V, Page 239, Q.52)

(b) Write short notes on following (any two) –
(i) NP hard Vs NP complete (See Unit-V, Page 228, Q.37)
(ii) Vertex cover problem (See Unit-V, Page 233, Q.47)
(iii) Hamiltonian path problem. (See Unit-V, Page 234, Q.50)

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2014
THEORY OF COMPUTATION
(CS-505)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.
(ii) All parts of each questions are to be attempted at one place.
(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.
(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) Write the definition of DFA. (See Unit-II, Page 38, Q.2) 10

(b) Construct a finite automata for the language $\{0^n \mid n \bmod 3 = 2, n \geq 0\}$.
(See Unit-I, Page 18, Prob.7) 10

(c) Prove $L = \{a^P \mid P \text{ is a prime}\}$ is not regular using pumping lemma.
(See Unit-II, Page 96, Prob.39) 10

(d) Find out the regular expression from given DFA.

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

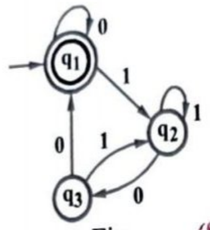


Fig. (See Unit-II, Page 88, Prob.32)

Or

Construct DFA equivalent to the NFA.
M = ({p, q, r, s}, {0, 1}, δ, p, {q, s})
where δ is defined in the following table.

δ	0	1
p	{q, s}	{q}
q	{r}	{q, r}
r	{s}	{p}
s	–	{p}

(See Unit-II, Page 45, Prob.4)

Unit-II

2. (a) Find the CFG for the regular expression (110 + 11)*(10)*.
(See Unit-III, Page 116, Prob.12) 10

(b) Show that the grammar S → a|abSb|aAb, A → bS | aAAb is ambiguous.
(See Unit-III, Page 126, Prob.26) 10

(c) Construct the reduced grammar equivalent to grammar –
S → aAa
A → Sb | bCC | DaA
A → abb | DD
E → aC
D → aDA
(See Unit-III, Page 134, Prob.40) 10

(d) Convert the following CFG to CNF
S → ABA
A → aA | ∈
B → bB | ∈
(See Unit-III, Page 135, Prob.42) 10

Or

Convert the following CFG to GNF –
A → BC
B → CA | b
C → AB | a
(See Unit-III, Page 148, Prob.57) 10

Unit-III

3. (a) Give the definition of Pushdown Automata with the help of diagram.
(See Unit-IV, Page 151, Q.1) 10

(b) Write the closure properties of CFL's. ** 10

(c) Is it true that non deterministic PDA is more powerful than that of deterministic PDA ? Justify your answer. 10
(See Unit-IV, Page 157, Q.10)

(d) Construct PDA for the following language –
$L = \{a^m b^n c^{m+n} \mid m, n \geq 1\}$
(See Unit-IV, Page 169, Prob.7) 10

Or

Show that the language $\{a^{n2} \mid n \geq 1\}$ is not context free. ** 10

Unit-IV

4. (a) Explain the term recursively enumerable language.
(See Unit-V, Page 241, Q.57) 10

(b) Give short note on Post's correspondence problem.
(See Unit-V, Page 248, Q.69) 10

(c) What are the features of Universal turing machine ?
(See Unit-V, Page 200, Q.23) 10

(d) Construct a turning machine for a language having equal number of a's and b's in it over the input set Σ = {a, b}.
(See Unit-V, Page 207, Prob.6)

Or

Find the languages obtained from the following operations –
(i) Union of two recursive languages.
(ii) Union of two recursively enumerable languages.
(iii) If L and complement of L are recursively enumerable.
(See Unit-V, Page 244, Q.64)

Unit-V

5. (a) What is tractable and interactable problem ? **

(b) Explain NP class problem with suitable example.
(See Unit-V, Page 227, Q.33)

(c) Give the non-deterministic algorithm for sorting elements in non decreasing order. **

(d) Prove that vertex cover problem is NP complete problem.
(See Unit-V, Page 233, Q.48)

Or

Write short note –
(i) NP-hard Problem
(ii) Travelling salesman problem.
(See Unit-V, Page 240, Q.54)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

RGPV

B.E. (Fifth Semester) EXAMINATION, June 2015
THEORY OF COMPUTATION
(CS-505)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.
(ii) All parts of each questions are to be attempted at one place.
(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.
(iv) Except numericals, Derivation, Design and Drawing etc.

1. (a) Differentiate between transition graph and transition matrix. (See Unit-I, Page 5, Q.6)
(b) Explain the term alphabet and string. (See Unit-II, Page 58, Q.13)
(c) What do you understand by DFA (Deterministic Finite Automata) and how is it represented ? (See Unit-II, Page 38, Q.2)
(d) What is Moore machine ? How Finite Automates can be converted into Moore machine ? Explain with the help of explain. (See Unit-I, Page 12, Q.12)

Or

State the Myhill-Nerode theorem and its applications. (See Unit-II, Page 68, Q.27)

2. (a) Show that the grammar S → a|sa|bss|ssb|sbs| is ambiguous. (See Unit-III, Page 125, Prob.25)
(b) How can be construct regular grammar from regular expression ? (See Unit-III, Page 103, Q.7)
(c) Describe context free and context sensitive grammar. (See Unit-III, Page 102, Q.2)
(d) Convert the following CFG to CNF –

$S \rightarrow ABA$
$A \rightarrow aA/\in$
$B \rightarrow bB/\in$

(See Unit-III, Page 135, Prob.42)

Or

Convert the grammar S → AB, A → BS|b, B → SA|a into Greibach normal form. (See Unit-III, Page 148, Prob.57)

3. (a) What is the additional feature PDA has when compared with NFA ? (See Unit-IV, Page 153, Q.3)

(b) Is it true that deterministic PDA and non-deterministic PDA are equivalent in the sense of language of acceptances ? Justify your answer. (See Unit-IV, Page 158, Q.11)
(c) Prove that CFLs are not closed under intersection. **
(d) Construct a PDA for the following language –

$L = \{a^m b^n c^{m+n} | m,n \geq 1\}$ (See Unit-IV, Page 169, Prob.7)

Or

Show that the language $\{a^{n^2} | n \geq 1\}$ is not context free. **

4. (a) Explain the concept of Turing machine model. (See Unit-IV, Page 185, Q.2)
(b) Construct TM accepting language –
L = {w/w has even no. of 2} (See Unit-V, Page 206, Prob.3)
(c) Discuss the properties of recursive and recursive enumerable languages. (See Unit-V, Page 241, Q.56)
(d) Construct a turing machine for checking the palindrome of a string of odd palindrome for Σ = {0, 1}. (See Unit-V, Page 218, Prob.19)

Or

Find the languages obtained from the following operations –
(i) Union of two recursive languages
(ii) Union of two recursively enumerable languages
(iii) If L and complement of L are recursively enumerable. (See Unit-V, Page 244, Q.64)

5. (a) What is tractable and untractable problem ? **
(b) Explain P-class problems with suitable example. (See Unit-V, Page 226, Q.32)
(c) Explain what is NP-hard problem ? (See Unit-V, Page 230, Q.41)
(d) Explain vertex cover problem with suitable example. (See Unit-V, Page 233, Q.47)

Or

Write short notes –
(i) Satisfiability problem (See Unit-V, Page 230, Q.43)
(ii) Hamiltonian path problem. (See Unit-V, Page 234, Q.50)

****Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus**

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2015
THEORY OF COMPUTATION
(CS-505)

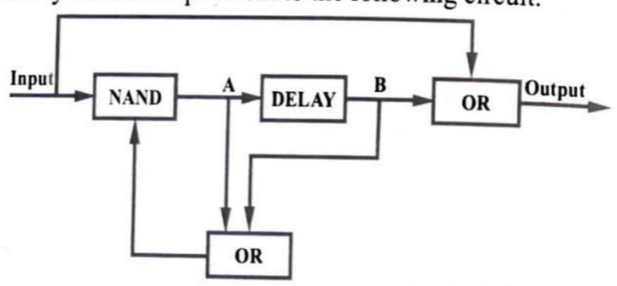
Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.
(ii) All parts of each questions are to be attempted at one place.
(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.
(iv) Except numericals, Derivation, Design and Drawing etc.

1. (a) Define DFA. List three household applications of finite automata. (See Unit-II, Page 39, Q.3)

(b) What is a trap state in FA ? State and explain the properties of transition functions. (See Unit-I, Page 8, Q.8)

(c) Design deterministic fine automation accepting the following languages over the alphabet (0, 1) –
(i) The set of all words ending in 00.
(ii) The set of all words except $\in$.
(iii) The set of all words that begin with 0.
(See Unit-I, Page 16, Prob.4)

(d) What do you mean by automata with output capability ? Draw a Mealy machine equivalent to the following circuit.



(See Unit-I, Page 26, Prob.19)

Or

What do you mean by closure by properties of regular languages ? State these properties. State pumping Lemma and show that $L = \{a^i b^j | i >= 1\}$ is not a regular language. (See Unit-II, Page 97, Prob.43)

2. (a) Show that the following grammar is ambiguous
$S \rightarrow aSbS|bSaS|\in$ (See Unit-III, Page 126, Prob.27)

(b) What are leftmost and rightmost derivations ? Explain with suitable example. (See Unit-III, Page 104, Q.10)

(c) Why CFG is not considered adequate for describing natural language ? Explain with suitable example. (See Unit-III, Page 103, Q.5)

(d) What do you mean by normal forms ? Reduce the grammar G with following productions to CNF.
$S \rightarrow ASA|bA$
$A \rightarrow B|S$
$B \rightarrow c$

(See Unit-III, Page 137, Prob.44)

Or

What do you mean by useless production ? Consider the grammar G = (V, T, P, S) where V, T, P, S are given as –
V = {S, A, B, C, E}
T = {a, b, c}
S = {S} and
P consists of
$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b$
$B \rightarrow C$
$E \rightarrow c$
Eliminate useless symbols and productions from the above grammar. (See Unit-III, Page 130, Prob.34)

3. (a) What is PDA ? Explain instantaneous description of PDA. (See Unit-IV, Page 152, Q.2)

(b) State the difference between PDA and the FA. (See Unit-IV, Page 153, Q.4)

(c) Design a PDA to accept the language $\{x \in (a, b)^* | n_a(x) > n_b(x)\}$. (See Unit-IV, Page 170, Prob.13)

(d) Consider the grammar –
$S \rightarrow aA$
$A \rightarrow aABC|bB|a$
$B \rightarrow b$
$C \rightarrow c$
Construct PDA corresponding to this grammar. Also provide moves of the PDA and the leftmost derivation for any string in the language defined by the grammar. (See Unit-IV, Page 179, Prob.26)

Or

The intersection of two context-free languages may or may not be context-free. **

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

Also write an algorithm for a given any context-free grammar to determine whether or not it can generate any words.

4. (a) Differentiate the purpose of the study of Turing machine with Finite Automata/Pushdown Automata. (See Unit-V, Page 186, Q.4)

(b) What is Turing-computable function ? Define recursive function. (See Unit-V, Page 192, Q.11)

(c) How UTM overcomes the limitation of Turing machine ? Also define UTM. (See Unit-V, Page 199, Q.22)

(d) Present a Turing machine that inserts symbol # in the beginning of a string on the turing tape. Assume $\Sigma = \{a, b\}$. (See Unit-V, Page 216, Prob.17)

Or

Design a Turing machine that adds two numbers presented in binary notation and leaves the answer on the tape in binary form. (See Unit-V, Page 222, Prob.23)

5. (a) Define P and NP problems. (See Unit-V, Page 227, Q.35)

(b) Discuss tractable and intractable problem. **

(c) Draw and explain commonly believed relationship between class P, NP, NP-complete and NP-hard. (See Unit-V, Page 229, Q.40)

(d) Define and discuss vertex cover problem. (See Unit-V, Page 233, Q.47)

Or

Discuss and explain travelling salesman problem. (See Unit-V, Page 239, Q.52)

RGPV

B.E. (Fifth Semester) EXAMINATION, June 2016
THEORY OF COMPUTATION
(CS-505)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

(ii) All parts of each questions are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

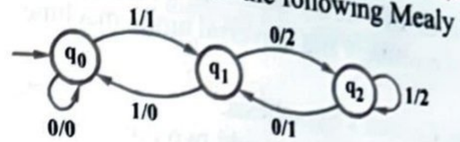
Unit-I

1. (a) Design DFA that accepts all strings with at most 3 a's. (See Unit-I, Page 15, Prob.2)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

(b) Design a NFA for $\{cbab^n \mid n >= 0\}$. (See Unit-II, Page 44, Prob.1)

(c) Construct Moore machine for the following Mealy machine.



(d) Write and explain Myhill-Nerode theorem. (See Unit-I, Page 33, Q.25)

Or

Construct NFA for the following grammar – (See Unit-II, Page 68, Q.27)

$S \rightarrow Ab|ab, A \rightarrow Ab|Bb, B \rightarrow Ba|a$

(See Unit-II, Page 44, Prob.3)

Unit-II

2. (a) Give CFG for R.E. $(011 + 1)^* (01)^*$. (See Unit-III, Page 115, Prob.11)

(b) Explain GNF conversion steps. (See Unit-III, Page 111, Q.22)

(c) Explain ambiguous grammar problem. (See Unit-III, Page 105, Q.12)

(d) Convert the following CFG to CNF

$S \rightarrow ASB \mid \in$
$A \rightarrow aAS \mid a$
$B \rightarrow SbS \mid A \mid bb$

(See Unit-III, Page 138, Prob.46)

Or

Convert the following grammar G into GNF

$S \rightarrow XA \mid BB$
$B \rightarrow b \mid SB$
$X \rightarrow b$
$A \rightarrow a$

(See Unit-III, Page 145, Prob.54)

Unit-III

3. (a) Explain PDA. (See Unit-IV, Page 151, Q.1)

(b) Explain how many way's PDA can accept (final out null store). (See Unit-IV, Page 154, Q.6)

(c) Explain pumping lemma for CFL. **

(d) Design pushdown automata which accepts $L = \{0^n 1^{2n} \mid n \geq 1\}$. (See Unit-IV, Page 167, Prob.4)

Or

Design a push down automata which accepts set of balanced parentheses. $(\{\{(\ )\}\})$ (See Unit-IV Page 171, Prob.14)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

Unit-IV

4. (a) Explain ID of a turing machine. (See Unit-V, Page 190, Q.8)
   (b) Explain multitape and universal turing machine. (See Unit-V, Page 200, Q.24)
   (c) Explain Church's hypothesis. (See Unit-V, Page 201, Q.28)
   (d) Design turing machine to add two numbers a and b. (See Unit-V, Page 221, Prob.22)

Or

Design turning machine for accepting strings of the language defined as $\{\omega\omega\gamma/\omega\in(0+1)^*\}$.

Unit-V

5. (a) Explain P and NP problems. (See Unit-V, Page 227, Q.35)
   (b) Difference between NP complete Vs NP hard problem. (See Unit-V, Page 228, Q.37)
   (c) Explain process of reducibility. (See Unit-V, Page 232, Q.46)
   (d) Describe hamiltonian path problem. (See Unit-V, Page 234, Q.50)

Or

Describe vertex cover problem. (See Unit-V, Page 234, Q.49)

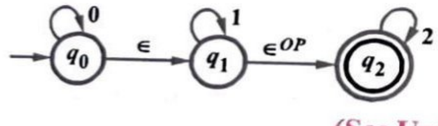
RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2016
THEORY OF COMPUTATION
(CS-505)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.
(ii) All parts of each questions are to be attempted at one place.
(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.
(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) Define two way finite automata. (See Unit-II, Page 74, Q.35)
   (b) Define regular expressions. (See Unit-II, Page 60, Q.18)
   (c) Design DFA accepting the language over the alphabet 0, 1 that have the set of all strings ending in 00. (See Unit-I, Page 17, Prob.5)
   (d) Determine the DFA equivalent to the given NFA.



(See Unit-II, Page 50, Prob.10)

Or

Explain Myhill Nerode theorem with example. (See Unit-II, Page 68, Q.27)

Unit-II

2. (a) Define ambiguity. (See Unit-III, Page 105, Q.12)
   (b) Define CFG. (See Unit-III, Page 102, Q.3)
   (c) Find L(G) for the grammar $S \to aCa$, $C \to aCa/b$. (See Unit-III, Page 116, Prob.13)
   (d) Prove that the given grammar is ambiguous.
   (i) $S \to aSb/SS/\in$
   (ii) $S \to AB/aaB$, $A \to a/Aa$, $B \to b$ (See Unit-III, Page 124, Prob.24)

Or

Eliminate unit, useless and $\in$-productions from the grammar $S \to aA/aBB$, $A \to aaA/\in$, $B \to bB/bbC$, $C \to B$. (See Unit-III, Page 131, Prob.36)

Unit-III

3. (a) Give the formal definition of PDA. (See Unit-IV, Page 151, Q.1)
   (b) Define deterministic PDA. (See Unit-IV, Page 156, Q.8)
   (c) Explain the pumping Lemma of context free languages. **
   (d) Design PDA to accept the language $L(G) = \{a^n b^m a^n/m, n\geq 1\}$. (See Unit-IV, Page 170, Prob.10)

Or

Design PDA to accept the language $L(G) = \{ww^R/w\in(0, 1)^*$ and w^R is the reverse of word w$\}$. (See Unit-IV, Page 172, Prob.16)

Unit-IV

4. (a) What are decidable and undecidable problems ? (See Unit-V, Page 246, Q.67)
   (b) Give two properties of recursively enumerable set. (See Unit-V, Page 243, Q.63)
   (c) Explain the types of turing machine. (See Unit-V, Page 201, Q.26)
   (d) Design turing machine for the language $L(G) = \{a^n b^n/n\geq 1\}$ (See Unit-V, Page 208, Prob.7)

Or

Design turing machine for the language $L(G) = \{a^n b^m a^{n+m}/n\geq 1, m\geq 1\}$. (See Unit-V, Page 214, Prob.14)

Unit-V

5. (a) What is NP hard problems ? (See Unit-V, Page 230, Q.41)
   (b) How P class problems different from NP class problem ? (See Unit-V, Page 227, Q.35)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

(c) What is Hamiltonian path problem ? Explain. (See Unit-V, Page 234, Q.50)

(d) Differentiate between tractable and untractable problems. **

Or

Explain vertex cover problem briefly. (See Unit-V, Page 234, Q.49)

RGPV

B.E. (Fourth Semester) EXAMINATION, June 2017
Choice Based Credit System (CBCS)
THEORY OF COMPUTATION
(CS-227)

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) Construct a finite automaton accepting all string over {0, 1}.
(i) Having odd number of 0's
(ii) Having even number of 0's and even number of 1's.
(See Unit-I, Page 22, Prob.13)

(b) Explain finite automaton and its various types. (See Unit-II, Page 40, Q.5)

2. (a) Construct an NFA for the regular expression $(0^*(10 + 01)^*11)^*$.
(See Unit-II, Page 77, Prob.21)

(b) State and prove the pumping lemma for regular sets.
(See Unit-II, Page 70, Q.30)

3. Find a grammar in GNF equivalent to the following grammar G.
$S \rightarrow AA|a$
$A \rightarrow SS|b$
(See Unit-III, Page 144, Prob.52)

4. Explain with example chomsky normal form and GNF forms ?
(See Unit-III, Page 109, Q.18)

5. (a) Construct a PDA for accepting language –
$L = \{a^n b^n \; n \geq 1\}$ by the Null stack.
(See Unit-IV, Page 167, Prob.3)

(b) Construct a PDA equivalent to the following grammar.
$S \rightarrow 0BB$
$B \rightarrow 0S|1S|0$
Test whether 010^4 is in N(A). (See Unit-IV, Page 176, Prob.21)

6. Construct the CFG corresponding to PDA,

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

$A = (\{q_0, q_1\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \phi)$ and δ is given by
$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, a)$
$\delta(q_1, b, a) = (q_1, a)$
$\delta(q_1, a_1, a) = (q_1, n)$
$\delta(q_1, n_1, z_0) = (q_1, n)$

7. (a) Write the closure property of regular expression. (See Unit-IV, Page 183, Prob.31)

(b) Explain melay and moore machines. (See Unit-II, Page 71, Q.32)

8. Write short notes on (any two) – (See Unit-I, Page 10, Q.11)
(a) Tractable and untractable problem. **
(b) Turing machine (See Unit-V, Page 186, Q.3)
(c) Recursive and recursively enumerable language. (See Unit-V, Page 241, Q.56)

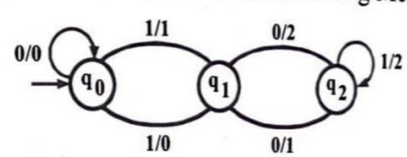
RGPV

CS-227 (CBCS)
B.E. IV Semester
EXAMINATION, Dec. 2017
Choice Based Credit System (CBCS)
THEORY OF COMPUTATION

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) Design DFA that accepts all strings with a most 3a's.
(See Unit-I, Page 15, Prob.2)

(b) Construct Moore machine for the following Mealy machine –



(See Unit-I, Page 33, Prob.25)

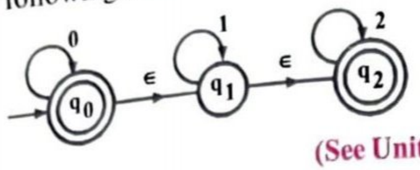
2. (a) Write and explain Myhill-Nerode theorem.
(See Unit-II, Page 68, Q.27)

(b) Define two-way finite automata with example.
(See Unit-II, Page 75, Q.36)

3. (a) Prove that the following is not regular language $\{0^n/n$ is perfect square$\}$.
(See Unit-III, Page 118, Prob.17)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

(b) Conver the following NFA with ∈-move to equivalent DFA.



(See Unit-II, Page 50, Prob.10)

4. (a) What do you mean by context free ? Explain the type of grammar. (See Unit-III, Page 103, Q.4)

(b) Convert the following grammar to CNF

$S \rightarrow bA \mid aB$

$A \rightarrow bAA \mid aS \mid a$

$B \rightarrow aBB \mid bSbb$

(See Unit-III, Page 141, Prob.49)

5. (a) Design a PDA to accept the language $L = a^n b^{2n} | n \geq 1$. (See Unit-IV, Page 167, Prob.4)

(b) Prove that every context free grammar is equivalent to PDA. (See Unit-IV, Page 163, Q.18)

6. (a) Construct turing machine for language $L = 0^n 1^n 2^n | n \geq 1$. (See Unit-V, Page 212, Prob.11)

(b) What do you mean by turing machine ? Explain different types of turing machine. (See Unit-V, Page 193, Q.14)

7. (a) Explain vertex cover problem with suitable example. (See Unit-V, Page 233, Q.47)

(b) What is tractable and untractable problem ? **

8. (a) Write short notes (any three) –
   (a) Satisfiability problem (See Unit-V, Page 230, Q.43)
   (b) Recursively enumerable language. (See Unit-V, Page 241, Q.57)
   (c) Greibach normal form (See Unit-III, Page 111, Q.20)
   (d) Transition graph Vs transition matrix (See Unit-I, Page 5, Q.6)
   (e) Hamiltonian path problem. (See Unit-V, Page 234, Q.50)

RGPV

CS-4005 (CBGS)
B.E. IV Semester
EXAMINATION, May 2018
Choice Based Grading System (CBGS)
THEORY OF COMPUTATION

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) Explain deterministic and non-deterministic finite automata with example. (See Unit-II, Page 39, Q.4) 7

****Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus**

(b) Explain applications of pumping lemma. (See Unit-II, Page 71, Q.31) 7

2. (a) Construct a NDFA accepting all string in {a, b} with either two consecutive a's or two consecutive b's. (See Unit-I, Page 25, Prob.16) 7

(b) Find a grammar in Chomsky normal form equivalent to $S \rightarrow aAD$; $A \rightarrow aB|bAB$; $B \rightarrow b$, $D \rightarrow d$. (See Unit-III, Page 142, Prob.50) 7

3. Explain the the following – 14
   (a) Regular grammars (See Unit-II, Page 60, Q.17)
   (b) Context free grammars (See Unit-III, Page 102, Q.3)
   (c) Derivation trees. (See Unit-III, Page 104, Q.9)

4. (a) Construct a grammar in GNF which is equivalent to the grammar $S \rightarrow AA|a$, $A \rightarrow SS|b$. (See Unit-III, Page 144, Prob.52) 7

(b) Define deterministic push down automata (DPDA). It is true that DPDA and PDA are equivalent in the sense of langauge acceptance is concern ? Justify your answer. (See Unit-IV, Page 159, Q.12) 7

5. (a) Demonstrate the working of your TM with an example. (See Unit-V, Page 188, Q.6) 7

(b) Explain in detail note on universal turing machines with example. (See Unit-V, Page 199, Q.22) 7

6. (a) Describe the recursively enumerable language with example. (See Unit-V, Page 243, Q.62) 7

(b) Construct a PDA for set of palindrome over the alphabet {a, b} $L(M) = \{WcW^R\}$. (See Unit-IV, Page 173, Prob.17) 7

7. (a) Give a detailed description of ambiguity in context free grammar. (See Unit-III, Page 105, Q.12) 7

(b) Explain vertex cover problem and Hamiltonian path problem. (See Unit-V, Page 239, Q.51) 7

8. Explain the following – 14
   (a) NP complete and NP hard (See Unit-V, Page 228, Q.37)
   (b) Traveling salesman problem (See Unit-V, Page 239, Q.52)
   (c) Partition problem. (See Unit-V, Page 240, Q.55)

RGPV

CS-4005 (CBGS)
B.E. IV Semester
EXAMINATION, November 2018
Choice Based Grading System (CBGS)
THEORY OF COMPUTATION

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) Construct a DFA M' equivalent to –

NFA M = ({q_0, q_1, q_2, q_3}, {a, b}, δq_0, {q_3} where δ is given by table 1.

Table 1: State Transition Table for NFA M

State/Σ	a	b
→ q_0	q_0, q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_3
q_3		q_2

(See Unit-II, Page 49, Prob.9)

(b) Define Mealy and Moore machines. Construct a Mealy machine which can output EVEN, ODD according as the total number of 1's encountered is even or odd. The input symbols are 0 and 1.

(See Unit-I, Page 26, Prob.18)

2. (a) State Arden's theorem. Determine the regular expression equivalent to finite automata given in figure 1.

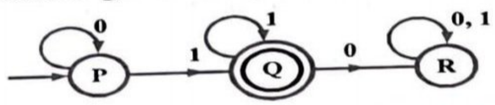


Figure 1 (See Unit-II, Page 88, Prob.31)

(b) State pumping lemma for regular sets and apply it to prove that L = $\{a^{j^2} | j \geq 1\}$ is not regular. (See Unit-II, Page 95, Prob.38)

3. (a) Consider the following grammar –

$S \rightarrow abSb \mid aAb \mid a$

$A \rightarrow bS \mid aAAb$

For the string abababb, illustrate –

(i) LMD (ii) RMD (iii) Derivation tree.

Also show that the given grammar is ambiguous.

(See Unit-III, Page 123, Prob.23)

(b) What do we eliminate during simplification of a context-free grammar (CFG) ? Simplify the following CFG –

$S \rightarrow aA \mid aBB$

$A \rightarrow aAA \mid \varepsilon$

$B \rightarrow bB \mid bbC$

$C \rightarrow B$

(See Unit-III, Page 132, Prob.37)

4. (a) Reduce the following grammar G into GNF –

$S \rightarrow XA \mid BB$

$B \rightarrow sB \mid b$

$X \rightarrow b$

$A \rightarrow a$

(See Unit-III, Page 145, Prob.54)

(b) Briefly describe the closure properties for context-free languages. Give proof for any one closure property. **

5. (a) Design a pushdown automata which accepts equal number of a's and b's over Σ = {a, b}. Also illustrate the moves of PDA for any input string.

(See Unit-IV, Page 166, Prob.1)

(b) Construct a PDA equivalent to the CFG given in Q.4(a). Illustrate the moves of PDA for input string 'babbab'.

6. (a) Briefly elaborate the followings –

(i) Representation of turing machine by instantaneous descriptors (IDs) (See Unit-V, Page 190, Q.8)

(ii) Turing machines as enumerators. (See Unit-V, Page 203, Q.31)

(b) Briefly describe the followings –

(i) Recursively enumerable and recursive langauges (See Unit-V, Page 241, Q.56)

(ii) Universal turing machine. (See Unit-V, Page 199, Q.22)

7. (a) Design a turing machine to accept the language L = $\{a^n b^n \mid n \geq 1\}$. Also illustrate the moves of turing machine for any input string in L.

(See Unit-V, Page 208, Prob.7)

(b) Explain NP-complete and NP-hard problems with suitable examlples.

(See Unit-V, Page 228, Q.37)

8. Write short notes on any two –

(a) Decision problems for CFLs **

(b) Church's hypothesis (See Unit-V, Page 201, Q.28)

(c) Vertex cover problem (See Unit-V, Page 233, Q.47)

(d) Hamilton path problem. (See Unit-V, Page 234, Q.50)

RGPV

CS-501 (CBGS)
B.Tech., V Semester
EXAMINATION, November 2019
Choice Based Grading System (CBGS)
THEORY OF COMPUTATION

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

1. (a) What do you mean by closure properties of regular languages ? Define some important closer properties. (See Unit-II, Page 73, Q.33)

   (b) Obtain the regular expression for the following set –

   (i) Set of all strings over $\Sigma = (a, b)$ with exactly two a.

   (ii) $L = \{a^x \mid x \text{ is divisible by 3 or 5}\}$

   (See Unit-II, Page 76, Prob.17)

2. Prove the following two properties –

   (a) Regular set are closed over the intersection operation.

   (b) Let A and B are context free language over the same alphabet Σ then, prove that $A \cup B$ (A union B) is also context free.

   (See Unit-II, Page 73, Q.34)

3. (a) Find the CNF (Chomsky Normal Form) for the following CFG –

   $S \rightarrow ABA$

   $A \rightarrow aA|\varepsilon$

   $B \rightarrow bB|\varepsilon$

   (See Unit-III, Page 135, Prob.42)

   (b) Convert the following grammar to GNF –

   $A_1 \rightarrow A_2A_3$

   $A_2 \rightarrow A_3A_1|b$

   $A_3 \rightarrow A_1A_2|a$

   (See Unit-III, Page 148, Prob.57)

4. Explain the Mealy and Moore machines in brief. Also describe what are the differences between them. (See Unit-I, Page 10, Q.11)

5. Design a PDA that accepts the language $L = \{a^n b^m a^n : m, n > 0\}$.

   (See Unit-IV, Page 170, Prob.12)

6. What do you understand by recursive enumerable languages ? Explain the turing machine with its properties. (See Unit-V, Page 242, Q.58)

7. Describe the NFA and DFA in brief. Also explain the construction of DFA for the following regular expression –

   $r = (0 + 1)^* (00 + 11) (0 + 1)^*$ (See Unit-II, Page 81, Prob.25)

8. Write short notes on any three –

   (i) Reducibility . (See Unit-V, Page 232, Q.46)

   (ii) Epsilon transition (See Unit-II, Page 44, Q.12)

   (iii) Halting problem of turing machine (See Unit-V, Page 247, Q.68)

   (iv) NP complete problems. (See Unit-V, Page 228, Q.37)

FOLLOW US ON...

@brocodeengineeringofficial

Brocode Engineering

@brocodeengineering

Brocode Engineering