# *Long questions ( UNIT 1)*

->Construct DFA

-> Minimize DFA

-> Algebraic laws of Regular expressions

-> Convert Automata to regular expressions

-> Differentiate DFA and NFA

-> Convert NFA to DFA

oufastupdates.com

# INTRODUCTION TO FINITE AUTOMATA

The finite automata is a mathematical model of a system (digital computer) with discrete inputs and outputs. Theory of finite automata is useful design tool for these systems. Finite automaton *"finite"* becomes the number of possible states and number of letters in the alphabet are both finite and *"automaton"* precise definition given below and it comes from the Greek but correct plural is *"Automata"*.

## DEFINITIONS

A finite automata is a collection of,

(1) A finite set of states, one of which is designated as the initial state, called the start state and some of which are designated as final states/accepting states.

(2) An alphabet $\Sigma$ of possible input letters, from which are formed strings, that are to be read one letter at a time.

(3) A finite set of transitions that tell for each state and for letter of the input alphabet which state to go next.

This definition functionality is being presented with an input string of letters that it reads letter by letter starting at the left most letter. Beginning at start state the letters determine a sequence of states. The sequence ends when the last input letter has been read.

There are three different classes of finite automata. They are,

(1) Deterministic Finite Automata (DFA).

(2) Non-Deterministic Finite Automata (NFA).

(3) Non-deterministic Finite Automata with $\varepsilon$-moves ($\varepsilon$-NFA).

# AN INFORMAL PICTURE OF FINITE AUTOMATA

Finite automaton is a mathematical model of a system with discrete inputs and outputs. Such a system can be in any one of the finite number of internal configurations or 'states' and each state of the system provides sufficient information concerning the past inputs so that the behaviour of the system could be studied on the provision of subsequent inputs.

There are many *examples* of finite automaton like, the control mechanism of an elevator, the controller for an automatic door, human brain, controllers of various household appliances, digital watches, calculators and many examples in computer science.

Finite automata (FA) is mainly used for modeling the *reactive systems.*

A reactive system is a system that changes its actions, outputs and conditions/status in response to stimuli from within or outside it. A reactive system is an event driven or control driven system continuously having to react to external and/or internal stimuli.

oufastupdates.com

In general, finite automaton are useful models to describe dynamic behaviours of reactive systems. When we use a finite automata to describe a system, we are not bother about the physical implementation of finite automata because finite automaton are very formal and rigorous to be implemented. But we can simulate the behavior of finite automaton by computer programs.

To model a reactive system with finite automaton, first the states and operations of that system are identified. These states become the states of the finite automaton that models it. Then the transitions between the states triggered by events and conditions (external or internal to the system) are identified and they become arcs in the transition diagram of the finite automaton. In addition to the above, actions that may take place in those states can also be added to the model to make the model clearer. For example, consider the following very simplified version of login process to a computer from the computer point of view. Let us assume that computer accepts a single user at a time.

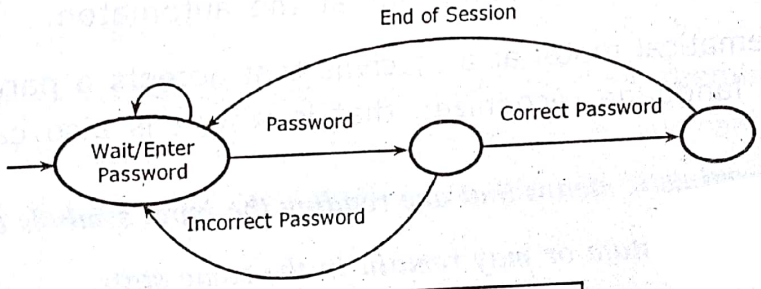



**Fig. 2.2.1** Login Process

Initially the computer waits for a user name to be typed in. This is one state of the system. When a name is typed in, it checks whether or not the name is valid. If it is valid, then it asks for and then waits for the password, which is another state. If the user name typed in is not valid, it loops into the initial state.

We could make it go to a different state and count the number of login attempts for security purpose. But let us make it simple. When a password is typed in and it is correct, it accepts the user and starts a session. When the session terminates, it goes back to the initial state and waits for another login.

We have several things to be done in designing and implementation of a reactive system. One important and initial thing is, what is the behaviour of a machine? If we want to describe the behaviour, there should be some effective way. An effective way is using some kind of modeling system. Finite automaton are basic and simplest model to be used for this purpose. "Automata theory is used to describe the behaviour of automatic machines mathematically", in other words, "A computational model or a recognition model can be described by some kind of automata, it is philosophy of automatic machines mathematically".

In computer Science field, the word "AUTOMATA" is used for recognizing or recognizer and word "AUTOMATON" is its plural. An automata understands a set of words or strings and rejects other. It is also known as acceptor One application of automata theory in Computer Science is in designing of lexical analyzer. which is a part of compiler.

oufastupdates.com

All machines have their internal configurations and these change time to time. The meaning is that at particular moment the machine resides in a configuration and next moment on next input the machine changes its configuration and enters the next configuration. Such types of internal configurations are known as states.

Let us think about storage of an automatic machine. In practice, it may be in large size but still it is finite. In theory, the storage may be of infinite capacity as we consider the term infinity in mathematics. The number of storage may be more than one. One question arise here that the storage is read-only storage or read-write storage? We consider both the types.

## DETERMINISTIC FINITE AUTOMATA (DFA)

A deterministic automaton is one in which each move is uniqually determined by the current configuration. If the internal state, input and contents of the storage are known, it is possible to predict the future behaviour at the automaton.

A DFA is a mathematical motel at a machine that accepts a particular language over $\Sigma$. It can be used as language recognizer, that is why it is also called an acceptor.

**NOTE :** *The term deterministic means that one reading the input symbols the automaton changes its current state to some other state or may remain in the same state.*

A DFA consists of following three things,

(1) **A Finite Set of States :** One of which is designed as the initial state, called the starting state and some (may be none) of which are designed as final stages.

(2) **Input Tape :** An input alphabet $\Sigma$.

(3) **A Transition System :** That tells for each state and each letter of the input alphabet, the state to which to go next.

### Definition of a Deterministic Finite Automaton

A deterministic finite automaton (or) DFA (or) FA is defined by the quintuple,

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

Where,

$Q$ = Finite non-empty set of states.

$\Sigma$ = Finite non-empty set of input symbols.

$\delta$ = Transition system,

$$\delta \in Q \times \Sigma \rightarrow Q$$

$q_0$ = Initial state and $q_0 \in Q$.

$F$ = A set at accepting states (of final states) and $F \subseteq Q$.

oufastupdates.com

## How a DFA Processes Strings

Suppose $a_1, a_2, a_3, \ldots, a_n$ is a sequence of input symbols. We start out with deterministic finite automata having $q_0, q_1, q_2, q_3, \ldots, q_n$ states where $q_0$ is the initial state and $q_n$ is the final state and transition function and processed as,

$$\delta(q_0, a_1) = q_1$$

$$\delta(q_1, a_2) = q_2$$

$$\delta(q_2, a_3) = q_3$$

$$\vdots \qquad \vdots$$

$$\vdots \qquad \vdots$$

$$\delta(q_{n-1}, q_n) = q_n$$

Input $a_1, a_2, \ldots, a_n$ is said to be "accepted" since $q_n$ is the member of the final states, and if not then it is "rejected". By adopting this approach we can define the language of a particular DFA and can also check that particular string is accepted or rejected by given FA.

**NOTE :** *So a string is said to be accepted by the given FA only when it is processed by transition function $\delta$ in such a manner that it ends at the final state.*

**Construct DFA for L = {w/w is in the form of 'x01y' for some strings x and y consisting of 0's and 1's}.**

**SOLUTION**

According to the given language, we must design a DFA that accepts all strings of the form w = x01y where, x and y are the strings consisting of 0's and 1's.

**Example :** DFA must accept strings like 11001001, 0011, 1000100.

Let us first design the DFA that fulfills the condition of the string containing 01.

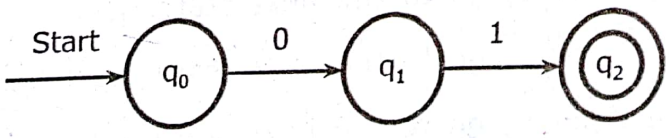



**Fig. 2.3.1** DFA for String 01

Since, a string should begin with some string x and end with some string y, we need to add two states to the above DFA, one as the initial state and another as the final state, as shown in Fig. 2.3.2.

oufastupdates.com

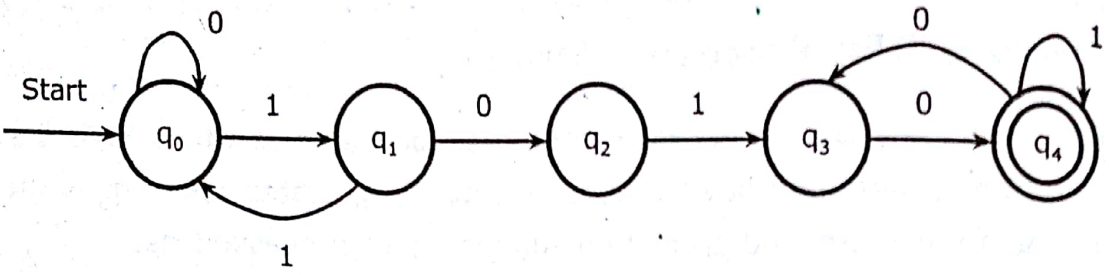



**Fig. 2.3.2** DFA for String x01y.

This is the required DFA.

## Simpler Notations for DFA's

Specifying a DFA has a five-tuples with a detailed description of the δ transition function is both tedious and very difficult to read. For describing DFA in two different notations are,

(1) DFA representation using, transition diagram/graph (Pictorial representation).

(2) DFA representation using, transition table.

## Transition Diagrams

A transition diagram/graph is a finite directed labeled graph in which each vertex (or) node represents a state and the directed edges indicates the transition of the state and the edges are labelled with the input alphabet.

A transition graph is shown in Fig. 2.3.3. In the Fig. 2.3.3 the initial is represented by a ring (or) circle with an arrow pointing towards it, final state by two concentric (or) circle and the other states are represented by just a ring (or) circle. The edges are labeled by input.

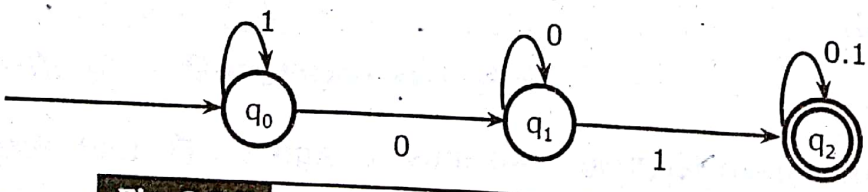



**Fig. 2.3.3** Transition Diagram for the DFA

**Example :** In the Fig. 2.3.3 state $q_0$ is an initial state, $q_2$ is a final state and $q_1$ is another state, and the transition performed by these states are follows,

From state $q_0$ and input 0, go to the next state $q_1$

From state $q_0$ and input 1, go to the next state $q_0$

From state $q_1$ and input 0, go to the next state $q_1$

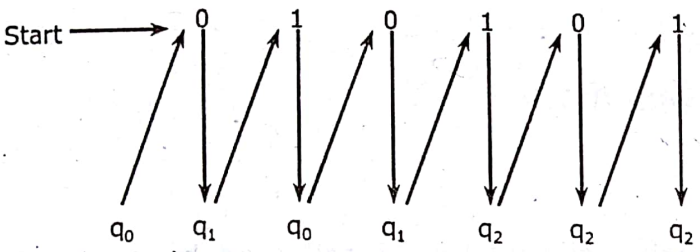
From state $q_1$ and input 1, go to the next state $q_2$

From state $q_2$ and input 0, go to the next state $q_2$

From state $q_2$ and input 1, go to the next state $q_2$

oufastupdates.com

(or)




The $q_2$ is a final state, hence the input is accepted.

**NOTE :** ***A transition diagram may contain more than one initial state.***

## Transition Tables

A DFA may be conveniently represented by a transition table. The transition table of Fig. 2.3.3 is represented in Table 2.3.1.

**Table 2.3.1** Input Alphabets

| States | 0 | 1 |
|---|---|---|
| $\rightarrow q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_2$ | $q_2$ |

The table represents the transition graph of $\delta$ i.e., to find the value of present state q and input alphabet x we have to look at the row labeled q and the column labeled x. The initial state is marked by an $\rightarrow$ and final states are marked by single rings (or) circle.

## Extending the Transition Function to Strings

Given a DFA $M = (Q, \Sigma, \delta\ q_0, F)$ the extended transition function $\delta^* : Q \times \Sigma^* \rightarrow Q$ is defined as follows,

$$\delta^*(p, \varepsilon) = p \qquad \text{... (1)}$$

$$\delta^*(p, wa) = \delta(\delta^*(p, w), a) \qquad \text{... (2)}$$

Where,

$a \in \Sigma$ and $a \in \Sigma^*$

It is immediate that $\delta^*(p, a) = \delta(p, a)$ for $a \in \Sigma$. The meaning of $\delta^*(p, w)$ is that it is the state reached from state p following the path from p specified by w. $\delta(p, \varepsilon) = (p, \varepsilon)$ means there is no input/empty string, the machine will not move to any new state, instead, it stays in the same state p.

It is also easy to show that,

$$\delta^*(p, wv) = \delta^*(\delta^*(p, w), v)$$

We can now define how a DFA accepts or rejects a string.

oufastupdates.com

**Obtain DFA to accept language,**

**L = {w/w of even length and begins with 01}.**

**SOLUTION**

It is clear the problem that starting string should be 01 followed by even number of 0's or 1's. So the DFA to accept corresponding language is shown in Fig. 2.3.4.

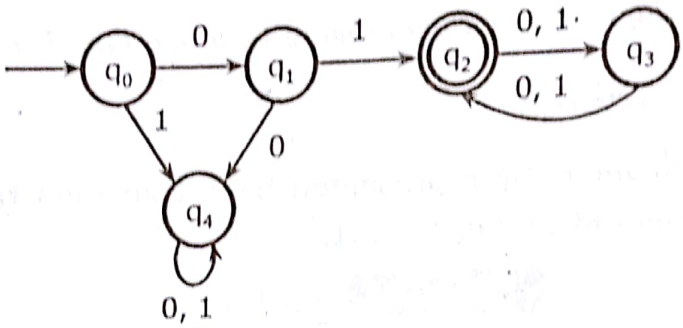



**Fig. 2.3.4** Transistion Diagram for L

Clearly state $q_2$ is the only accepting state. The DFA can thus be given as,

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

Where,

$\delta$ = The transition function.

We can also represent this DFA by a transiton table. Fig. 2.3.4 shows this table.

Suppose the input is 011101. Since this string starts with 01 and is of even length we expect it is in the langauge. Thus, we expect that $\hat{\delta}(q_0, 011101) = q_2$, since $q_2$ is the only accepting state.

| $\delta$ | **0** | **1** |
|---|---|---|
| $*q_0$ | $q_1$ | $q_4$ |
| $q_1$ | $q_4$ | $q_2$ |
| $\rightarrow q_2$ | $q_3$ | $q_3$ |
| $q_3$ | $q_2$ | $q_2$ |
| $q_4$ | $q_4$ | $q_4$ |

**Fig. 2.3.5** Transition Table for the Above DFA

However, we are not just concerned with the design of this DFA, we use to illustrate the construction of extended transitions ($\delta\times$) from its transition function $\delta$. The input string is 011101,

oufastupdates.com

$$\delta \times (q_0, 011101) = \delta \times (\delta (q_0, 0) 11101)$$
$$= \delta \times (\delta (q_1, 1) 1101)$$
$$= \delta \times (\delta (q_2, 1) 101)$$
$$= \delta \times (\delta (q_3, 1) 01)$$
$$= \delta \times (\delta (q_2, 0) 1)$$
$$= \delta \times (\delta (q_3, 1))$$
$$= \delta \times (\delta (q_2, \epsilon)$$
$$= q_2 \text{ is final state.}$$

## The Language of a DFA

Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, the language L(M) accepted (or) recognized by M is the language,

$$L(M) = \{w \in \Sigma^* \mid \delta^* (q_0, w) \in F\}$$

Thus, a string $w \in \Sigma^*$ is accepted if the path from $q_0$ on input w ends in a final state,

$$\overline{L(M)} = \{w \in \Sigma^* \mid \delta^* (q_0, w) \notin F\}$$

Thus, a string $W \in \Sigma^*$ is accepted if the path from $q_0$ on input w ends in some other state i.e., not a final state. Therefore DFA will not be in final state and so the string is rejected.

**Design a DFA that accepts set of all strings that contain 0's or 1's and end in 00.**

**SOLUTION**

We are required to design a DFA for the regular expression $r = (0 + 1)^*00...$ In other words the DFA should be designed to accept the language of r.

i.e., L(M) = {00, 100, 1100, 0000, 111000, ...}

Where,

M is a DFA.

Consider,

$$\Sigma = \{0, 1\}$$
$$Q = \{q_0, q_1, q_2\}$$
$$q_0 = \text{Initial state}$$
$$q_2 = \text{Final state and } \delta : Q \times \Sigma \rightarrow Q$$

oufastupdates.com

**Transition Diagram**

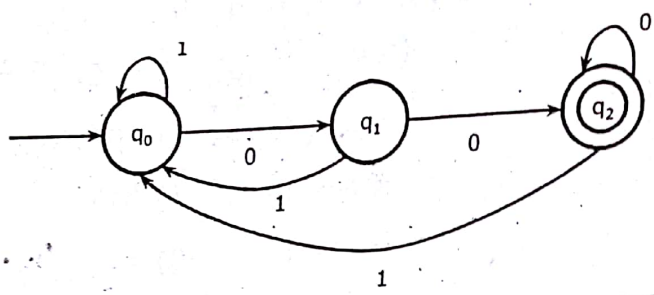



**Fig. 2.3.6** State Transition to Represent L(M) = {W ∈ Σ* | W ends with 00}

**Transition Table**

**Table 2.3.2** State Transition Table

| Input / State | Input Symbol 0 | Input Symbol 1 |
|---|---|---|
| → $q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_2$ | $q_0$ |
| * $q_2$ | $q_2$ | $q_0$ |

**DFA Action for the Input String**

(i) To show that the string 100 is accepted by DFA.

Using sequence state diagram,

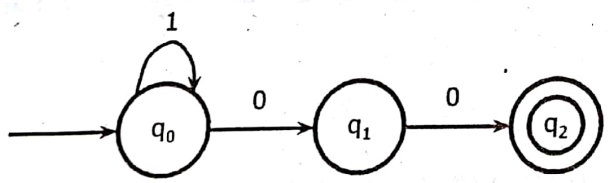



**Fig. 2.3.7** State Transition Diagram for 100

Since we encounter the end of the input and we are in the final state, we say that string is accepted by machine M. Thus 100 is in L(M).

Using extended transition function,

Consider,

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

$$\delta(q_0, 10) = \delta(\delta(q_0, 1), 0)$$

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

$$\delta(q_0, 100) = \delta(\delta(q_0, 10), 0)$$

$$= \delta(q_1, 0) = q_2$$

oufastupdates.com

Since after scanning the entire string, we reach at the final state $q_2$, the given string 100 is thus accepted by the DFA M. Thus 100 is in L(M).

Using vdash function($\vdash$),

An input string a is accepted by $\vdash$ M iff ($W \in \Sigma^* | \vdash (q, e) \in F$),

$$\vdash : Q \times \Sigma \rightarrow Q$$

(q, e) means 'reached end of input'.

Consider,

$$(q_0, 100) \vdash M\ (q_0, 00)$$
$$\vdash M\ (q_1, 0)$$
$$\vdash M\ (q_2, e)$$

Since we reached the end of input (e) and we are in the final state $q_2$, hence 100 is accepted by M. Thus 100 is in L(M).

(ii) To show that the string 010 is rejected by DFA.

Using sequence state diagram,

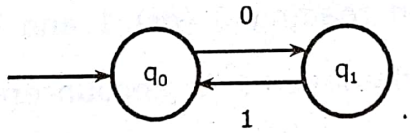



**Fig. 2.38** State Transition Diagram for 010

Since we encounter the end of input and $q_1$ is not the final state, we say that the string 010 is rejected by the machine.

Using extended transition function,

$$\delta(q_0, 0) = q_1$$
$$\delta(q_0, 01) = \delta(\delta(q_0, 0), 1)$$
$$= \delta(q_1, 1) = q_0$$
$$\delta(q_0, 010) = \delta(\delta(q_0, 01), 0)$$
$$= \delta(q_0, 0) = q_1$$

Since after scanning the entire, we did not reach the final state $q_2$, hence the string 010 is rejected by DFA.

Using vdash function,

$$(q_0, 010) \vdash M\ (q_1, 10)$$
$$\vdash M\ (q_0, 0)$$
$$\vdash M\ (q_1, e)$$

Since we reached the end of input (e) and we are not in final state $q_2$, hence 010 is rejected by M. Thus 010 is not in L(M).

oufastupdates.com

**Design a DFA to accept the language given below,**

**L = {W/W is of even length and begins with O's}.**

**SOLUTION**

The language begins with letter O and must be even length. So, the set of language look like as follows,

{00, 01, 0001, 011101 ....}

To design a DFA machine use the following steps,

(i) Let us consider an initial state i.e., $q_0$.

(ii) $q_1$ state entered on reading 0 in state $q_0$.

(iii) $q_2$ state entered on reading 0 (or) 1 and so far is of odd length.

(iv) $q_3$ state entered on reading 0 (or) 1 and even length.

(v) $q_4$ state entered whenever 1 is encountered in state $q_0$.

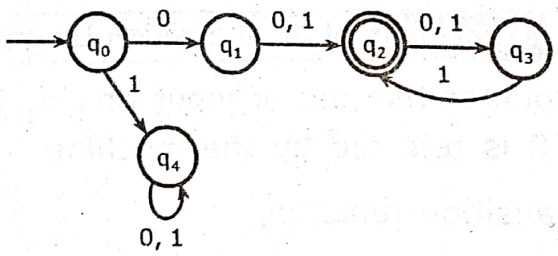



**Fig. 2.3.9** Transition Diagram for DFA of L

The DFA can be defined as,

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

**Table 2.3.3** Transition of the DFA

| States | Input Symbols | |
|---|---|---|
| | 0 | 1 |
| → $q_0$ | $q_1$ | $q_4$ |
| $q_1$ | $q_2$ | $q_2$ |
| → $q_2$ | $q_3$ | $q_3$ |
| $q_3$ | $q_2$ | $q_2$ |
| $q_4$ | $q_4$ | $q_4$ |

oufastupdates.com

**Draw the transition diagram for the DFA with the following transition table,**

| δ | 0 | 1 |
|---|---|---|
| → $q_0$ | $q_2$ | $q_1$ |
| *$q_1$ | $q_1$ | $q_1$ |
| $q_2$ | $q_2$ | $q_1$ |

**SOLUTION**

The transition diagram for DFA is consider,

$S = \{0, 1\}$

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

$q_0$ = Initial state.

$q_1$ = Final state.

$\delta = Q \times S \rightarrow Q$

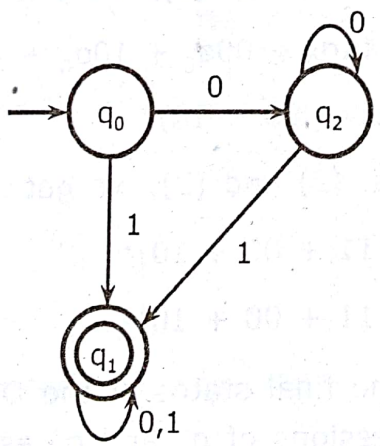



**Fig. 2.3.10**

The language of the DFA is,

$$q_0 = \epsilon$$

$$q_2 = 0q_0 + 0q_2 = 0q_0 + 0^* = 0\epsilon + 0^* = 0 + 0^*$$

$$q_1 = 0q_1 + 1q_1 + 1q_0 + 1q_2$$

$$q_1 = (0 + 1)\, q_1 + 1\epsilon + 1\,(0 + 0^*)$$

$$q_1 = (0 + 1)^* + 1 + 1\,(0 + 0^*)$$

Since, $q_1$ is a final state of DFA the language of the DFA is,

$$L = (0 + 1)^* + 1 + 1\,(0 + 0^*).$$

oufastupdates.com

**Find the language accepted by following finite automaton,**

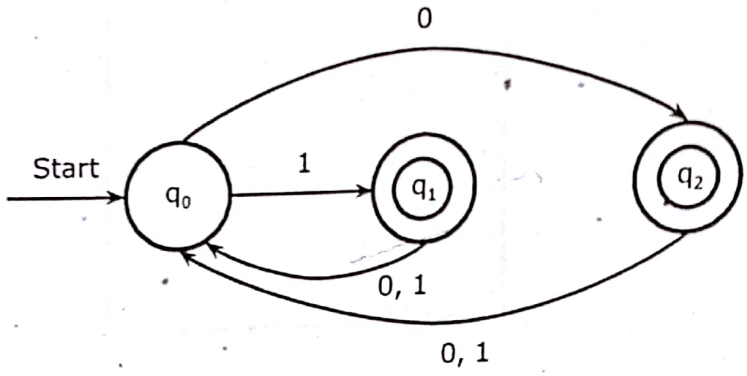



**Fig. 2.3.11** State Transition Diagram

**SOLUTION**

The language for the given DFA can be obtained as follows,

$$q_0 = 0q_1 + 1q_1 + 0q_2 + 1q_2 + \epsilon \quad \text{... (1)}$$

$$q_1 = 1q_0 \quad \text{... (2)}$$

$$q_2 = 0q_0 \quad \text{... (3)}$$

Substituting Eqs. (2) and (3) in Eq. (1), we get,

$$q_0 = 01q_0 + 11q_0 + 00q_0 + 10q_0 + \epsilon = (01 + 11 + 00 + 10)\, q_0 + \epsilon$$

$$q_0 = (01 + 11 + 00 + 10)^* \quad \text{... (4)}$$

Substituting Eq. (4) in Eqs. (2) and (3), we get,

$$q_1 = 1(01 + 11 + 00 + 10)^*$$

$$q_2 = 0(01 + 11 + 00 + 10)^*$$

Since $q_1$ and $q_2$ are the final states of the DFA, the language is given by the union of the regular expressions of $q_1$ and $q_2$ as,

$$L = r.e(q_1) + r.e(q_2)$$

$$= 1(01 + 11 + 00 + 10)^* + 0(01 + 11 + 00 + 10)^*.$$

**Design a DFA that accepts the set $\{a^n\} \cup \{b|n \geq 1\}$.**

**SOLUTION**

This is a language that generates strings containing one or more number of a's or a string containing a single b. That is we have to design a DFA that accepts the strings like a, b, aa, aaa, aaaa etc.

oufastupdates.com

The DFA accepting the given language is shown in Fig. 2.3.12.

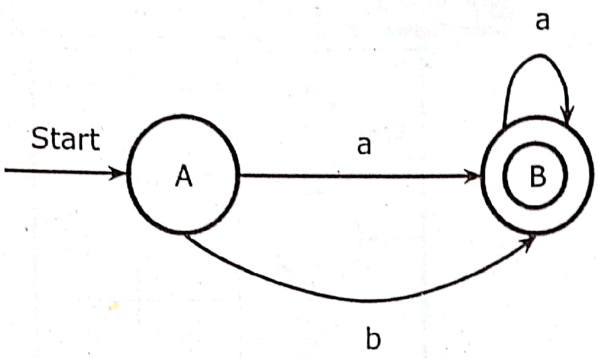



**Fig. 2.3.12** DFA for $a^n \cup b$

**Design a DFA that,**

**(i) Starts with 0 and has odd number of 0's.**

**(ii) Starts with 1 and has even number of 1's.**

**SOLUTION**

We are required to design a DFA for the regular expression,

$$r = (0 + 1).((0 + 1).(0 + 1))^*$$

i.e., L(M) = {01100, 110011, ...}.

Consider,

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

$q_0$ = Initial state.

$q_1$ = Final state and $\delta$ is given by,

**Transition Diagram**

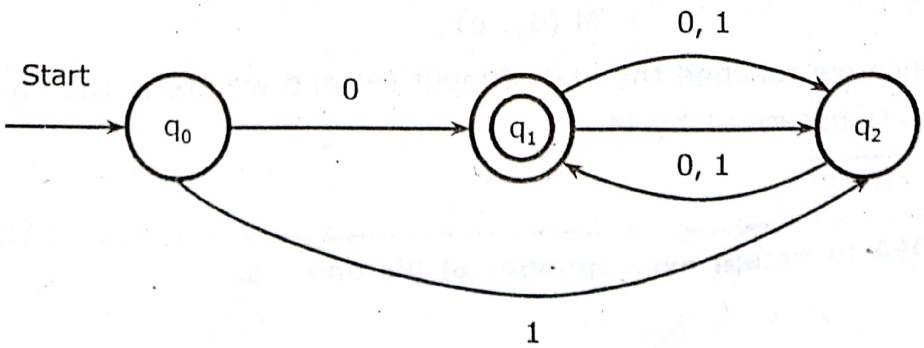



**Fig. 2.3.13** State Transition Diagram

{W | W starts with 0 and has odd 0's, W starts with 1 and has even 1's}

oufastupdates.com

**Transition Table**

**Table 2.3.4** State Transition Table

| State \ Input | Input Symbol | |
|---|---|---|
| | 0 | 1 |
| → $q_0$ | $q_1$ | $q_2$ |
| * $q_1$ | $q_2$ | $q_2$ |
| $q_2$ | $q_1$ | $q_1$ |

**DFA Action for the Input String**

(i) To show that the string 01100 is accepted by DFA.

Consider,

$$(q_0, 01100) \vdash M(q_1, 1100)$$
$$\vdash M(q_2, 100)$$
$$\vdash M(q_1, 00)$$
$$\vdash M(q_2, 0)$$
$$\vdash M(q_1, e)$$

Since we reached the end of input (e) and we are in the final state $q_1$, hence 01100 is accepted by L(M).

(ii) To show that 110011 is accepted by DFA.

Consider,

$$(q_0, 110011) \vdash M(q_2, 10011)$$
$$\vdash M(q_1, 0011)$$
$$\vdash M(q_2, 011)$$
$$\vdash M(q_1, 11)$$
$$\vdash M(q_2, 1)$$
$$\vdash M(q_1, e)$$

Since we reached the end of input (e) and we are in the final state $q_1$, hence 110011 is accepted by M.

oufastupdates.com

Consider,

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

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

$q_0$ = Initial and final state.

$\delta$ is given by,

**Transition Diagram**

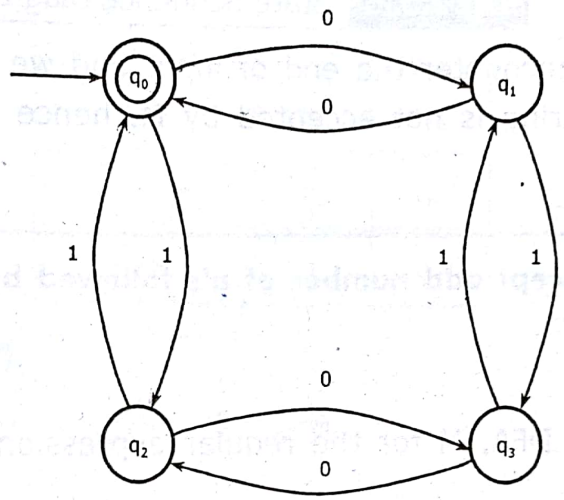



**Fig. 2.3.14** State Transition Diagram

$\{W \in \Sigma^* | \text{ W has even length of 0's and 1's}\}$

**Transition Table**

**Table 2.3.5** State Transition Table

| Input / State | Input Symbol 0 | Input Symbol 1 |
|---|---|---|
| → * $q_0$ | $q_1$ | $q_2$ |
| $q_1$ | $q_0$ | $q_3$ |
| $q_2$ | $q_3$ | $q_0$ |
| $q_3$ | $q_2$ | $q_1$ |

**DFA Action for the Input String**

(i) To show that 0011 is accepted by DFA,

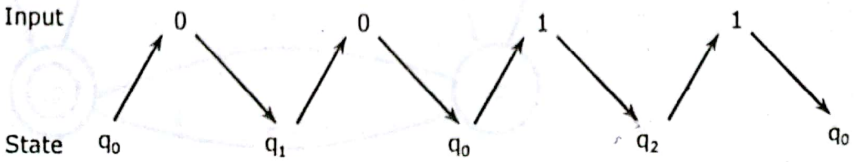



**Fig. 2.3.15** State Sequence Diagram for 0011

oufastupdates.com

Since we encounter the end of input and we are in the final state, we say that the string is accepted by M.

(ii) To show that 00110 is rejected by DFA,

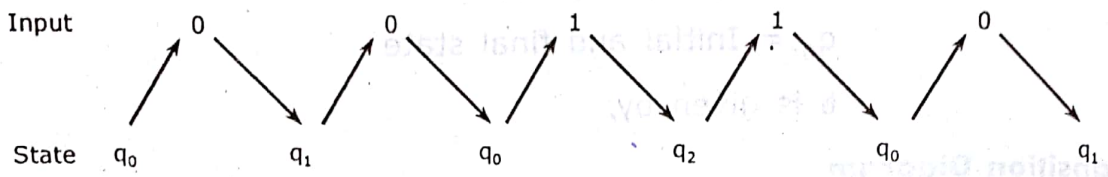



**Fig. 2.3.16** State Sequence Diagram for 00110

Since we encounter the end of input and we are not in the final state, we say that the string is not accepted by M, hence 00110 is not in L(M).

## SOLVED PROBLEM 7

**Design a DFA to accept odd number of a's followed by b's.**

**SOLUTION**

This is to design a DFA, M for the regular expression r = (aa)* a(bb)*b

i.e L(M) = {ab, aaabbb, abbbbb, ababab, aabbab, ...}.

Consider,

$\Sigma$ = {a, b}

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

$q_0$ = Initial state

$q_3$ = Final state and $\delta$ is given by,

***Transition Diagram***

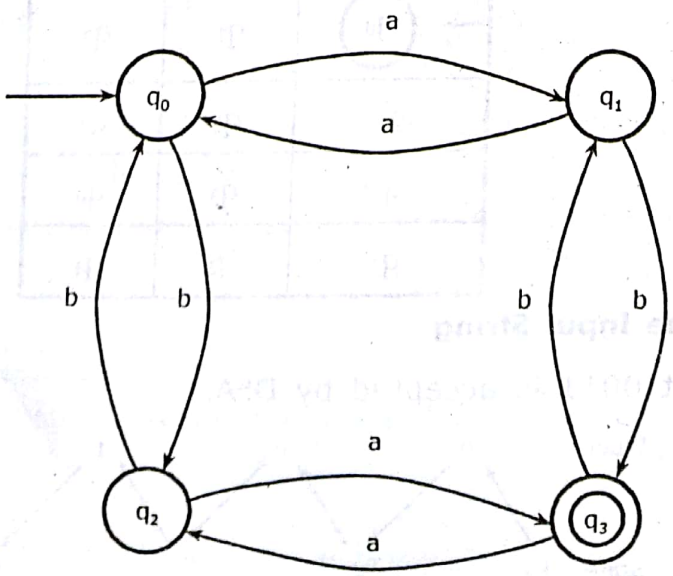



**Fig. 2.3.17** State Transition Diagram for L(M)

oufastupdates.com

**Transition Table**

**Table 2.3.6** State Transition Table

| State \ Input | Input Symbol | |
|---|---|---|
| | a | b |
| → $q_0$ | $q_1$ | $q_2$ |
| $q_1$ | $q_0$ | $q_3$ |
| $q_2$ | $q_3$ | $q_0$ |
| * $q_3$ | $q_2$ | $q_1$ |

**DFA Action for Input String**

(i) To show that ababab is accepted by DFA.

Consider,

$$[\ _0, \text{ababab}) \vdash M\ (q_1, \text{babab})$$
$$\vdash M\ (q_3, \text{abab})$$
$$\vdash M\ (q_2, \text{bab})$$
$$\vdash M\ (q_0, \text{ab})$$
$$\vdash M\ (q_1, \text{b})$$
$$\vdash M\ (q_3, \text{e})$$

Since we reached the end of input(e), and we are in the final state, hence ababab is accepted by M.

(ii) To show that bbabb is rejected by DFA.

Consider,

$$(q_0, \text{bbabb}) \vdash (q_2, \text{babb})$$
$$\vdash M\ (q_0, \text{abb})$$
$$\vdash M\ (q_1, \text{bb})$$
$$\vdash M\ (q_3, \text{b})$$
$$\vdash M\ (q_1, \text{e})$$

Since we reached the end of input(e) and not in the final state, hence bbabb is rejected by DFA.

oufastupdates.com

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

**SOLUTION**

This is design a DFA, M for the regular expression r = (aa)* a.(bb)*.

i.e L(M) = {aaa, aaabb, a, abb, abbbb, aaabbbb, ...}.

Consider,

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

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

$q_0$ = Initial state

$q_1$ = Final state and $\delta$ is given by,

**Transition Diagram**

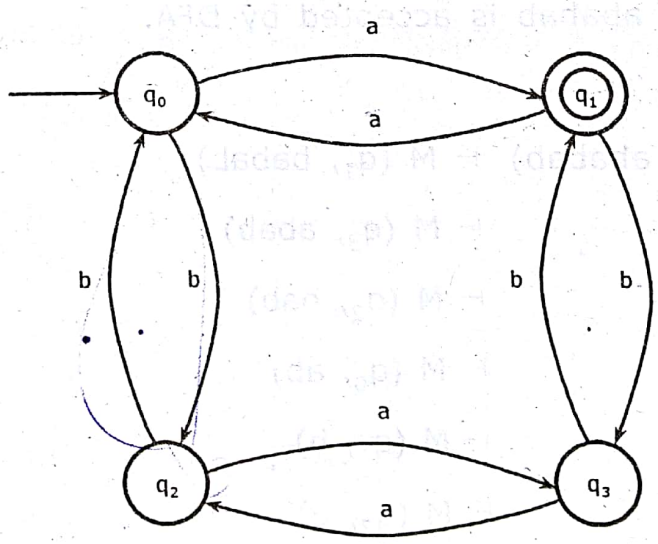



**Fig. 2.3.18** State Transition Diagram

{W ∈ Σ*| W has odd number of a's and even no of b's}

**Transition Table**

**Table 2.3.7** State Transition Table

| Input / State | Input Symbol | |
|---|---|---|
| | a | b |
| → $q_0$ | $q_1$ | $q_2$ |
| * $q_1$ | $q_0$ | $q_3$ |
| $q_2$ | $q_3$ | $q_0$ |
| $q_3$ | $q_2$ | $q_1$ |

oufastupdates.com

**DFA Action for the Input String**

(i) To show that ababa is accepted by DFA,

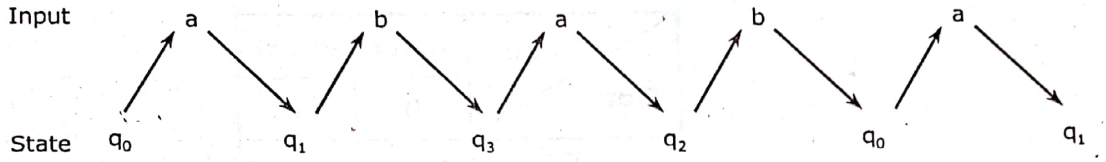



**Fig. 2.3.19** State Sequence Diagram for ababa

Since we encounter the end of input, which is in the final state, we say that the string is accepted by M.

(ii) To show that ababaa is rejected by DFA,

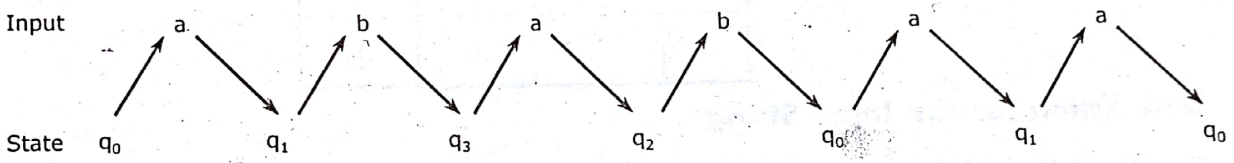



**Fig. 2.3.20** State Sequence Diagram for ababaaa

Since we encounter the end of input and which is not in the final state, we say that the string is rejected by M.

## SOLVED PROBLEM 9

**Design a DFA that contains set of all strings ending with 3 consecutive zeros, over the alphabet {0, 1}.**

**SOLUTION**

This is to design a DFA, M for the regular expression r = (0 + 1)* 000 or r = 1*(01)*(001)*(0001)* 000

Consider,

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

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

$q_0$ = Initial state

$q_3$ = Final state and $\delta$ is given by,

**Transition Diagram**

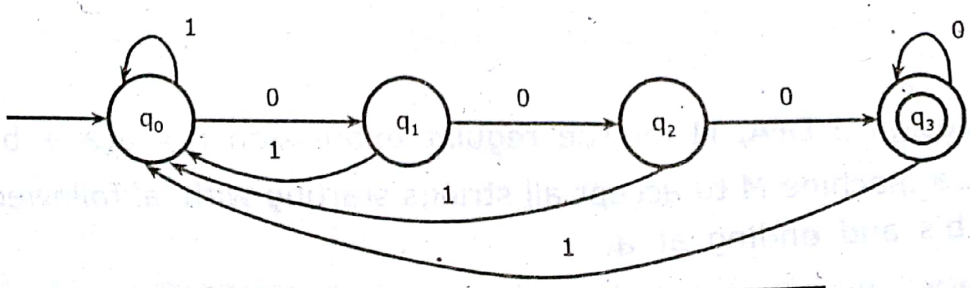



**Fig. 2.3.21** Transition Diagram for L(M)

oufastupdates.com

**Table 2.3.8** State Transition Table

| State \ Input | Input Symbol 0 | Input Symbol 1 |
|---|---|---|
| → $q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_2$ | $q_0$ |
| $q_2$ | $q_3$ | $q_0$ |
| $*q_3$ | $q_3$ | $q_0$ |

**DFA Action for the Input String**

To show that 01011000 is accept by DFA.

Consider,

$$(q_0, 01011000) \vdash M\ (q_1, 1011000)$$
$$\vdash M\ (q_0, 011000)$$
$$\vdash M\ (q_1, 11000)$$
$$\vdash M\ (q_0, 1000)$$
$$\vdash M\ (q_0, 000)$$
$$\vdash M\ (q_1, 00)$$
$$\vdash M\ (q_2, 0)$$
$$\vdash M\ (q_3, e).$$

Since we reached the end of input(e) and are in the final state $q_3$, hence 01011000 is accepted by M.

## SOLVED PROBLEM 10

**Design a DFA, to accept the Language L = {aWa | W ∈ (a + b)*} over alphabet Σ = {a, b}.**

**SOLUTION**

This is to design a DFA, M for the regular expression r = a(a + b)* a or r = ab* (ab)*a* i.e. a machine M to accept all strings starting with 'a' followed by any number of a's and b's and ending at a.

i.e., L(M) = {aa, aba, aaaa, abbbaaa, ...}.

oufastupdates.com

Consider,

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

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

$q_0$ = Initial state

$q_2$ = Final state and $\delta$ is given by

**Transition Diagram**

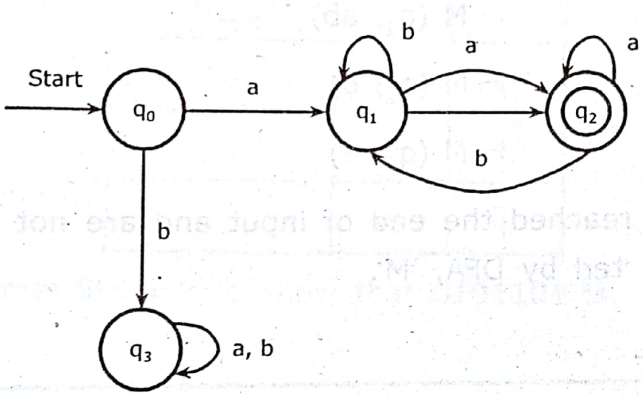



**Fig. 2.3.22** State Transition Diagram for L(M)

**Transition Table**

**Table 2.3.9** State Transition Table

| State \ Input | Input Symbol a | Input Symbol b |
|---|---|---|
| → $q_0$ | $q_1$ | $q_3$ |
| $q_1$ | $q_2$ | $q_1$ |
| * $q_2$ | $q_2$ | $q_1$ |
| $q_3$ | $q_3$ | $q_3$ |

**DFA Action for the Input String**

(i) To show that ababa is accepted by DFA.

Consider,

$$(q_0, ababa) \vdash M\ (q_1, baba)$$
$$\vdash M\ (q_1, aba)$$
$$\vdash M\ (q_2, ba)$$
$$\vdash M\ (q_1, a)$$
$$\vdash M\ (q_2, e)$$

Since we reached the end of input (e) and are in the final state, hence ababa is accepted.

oufastupdates.com

(ii) To show that ababab is rejected by DFA.

Consider,

$$(q_0, \text{ababab}) \vdash M\ (q_1, \text{babab})$$
$$\vdash M\ (q_1, \text{abab})$$
$$\vdash M\ (q_2, \text{bab})$$
$$\vdash M\ (q_1, \text{ab})$$
$$\vdash M\ (q_2, \text{b})$$
$$\vdash M\ (q_1, \text{e})$$

Since we reached the end of input and are not in the final state, hence ababab is rejected by DFA, 'M'.

## SOLVED PROBLEM 11

**Design a DFA, over alphabet $\Sigma = \{0, 1\}$, that contains set of strings of 0's or 1's except those containing substring 110.**

**SOLUTION**

We need to design DFA, M for the regular expression $r = 0^* + 0^* (10)^* + 0^* (10)^* 11$

i.e., $L(M) = \{00, 11, 0101, 00010010011, ....\}$.

Consider,

$$\Sigma = \{0, 1\}$$
$$Q = \{q_0, q_1, q_2, q_3\}$$
$$q_0 = \text{Initial state}$$

$F = \{q_0, q_1, q_2\}$ as final state and $\delta$ is given by,

**Transition Diagram**

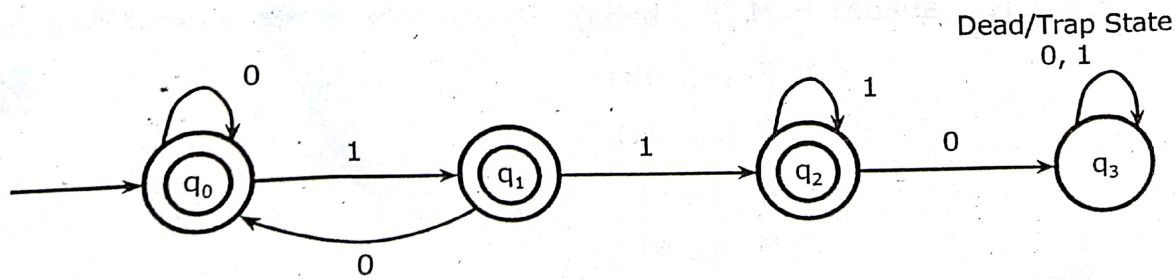



**Fig. 2.3.23** State Transition Diagram for L(M)

oufastupdates.com

**Transition Table**

**Table 2.3.10** State Transition Table

| State \ Input | Input Symbol 0 | Input Symbol 1 |
|---|---|---|
| → $q_0$ * | $q_0$ | $q_1$ |
| * $q_1$ | $q_0$ | $q_2$ |
| * $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_3$ | $q_3$ |

**DFA Action for the Input String :** To show that 0101101 is rejected by DFA.

Consider,

$$
\begin{aligned}
(q_0, 0101101) &\vdash M(q_0, 101101) \\
&\vdash M(q_1, 01101) \\
&\vdash M(q_0, 1101) \\
&\vdash M(q_1, 101) \\
&\vdash M(q_2, 01) \\
&\vdash M(q_3, 1) \\
&\vdash M(q_3, e).
\end{aligned}
$$

Since we reached the end of input (e) and are not in any of the final states $(q_0, q_1, q_2)$, thus 0101101 is rejected by M.

## SOLVED PROBLEM 12

**Construct a DFA to accept the set of all strings of the form $0^{2n}11$, $n \geq 0$.**

**SOLUTION**

Let,

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

Such that,

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

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

$q_0 = q_0$

$F = \{q_4\}$

oufastupdates.com

The transition δ is shown in Fig. 2.3.24.

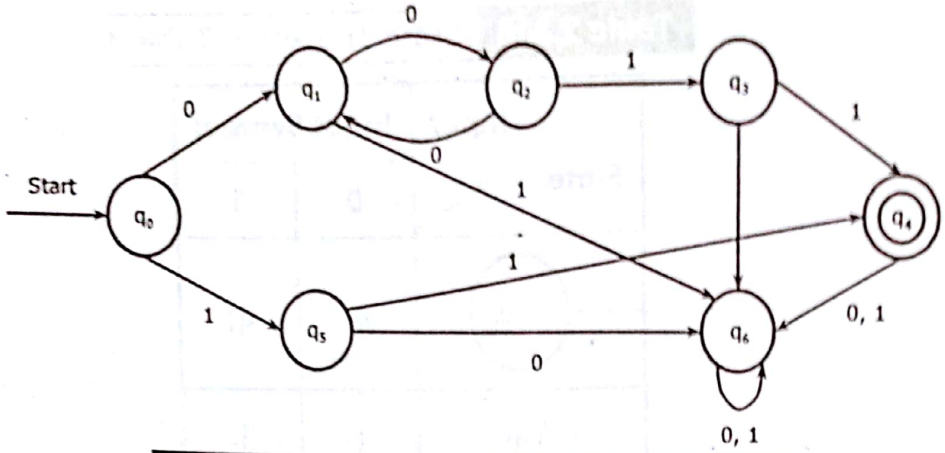



**Fig. 2.3.24** DFA to Accept $\{0^{2n}1 \mid n \geq 0\}$

## SOLVED PROBLEM 13

**Construct an automaton to accept only the word ∈ over the alphabet Σ = {a, b}, given Q $_0$, $q_1$}.**

**SOLUTION**

Let,

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

Such that,

$Q = \{q_0, q_1\}$

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

$q_0 = q_0$ and $F = \{q_0\}$.

The transition δ, for the DFA to accept only the word ∈, is shown in Fig. 2.3.25.

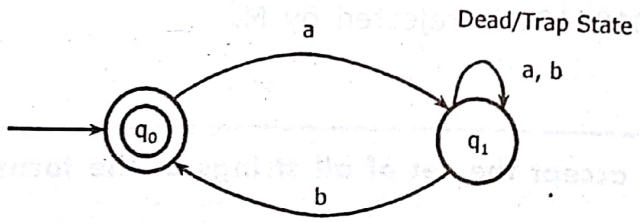



**Fig. 2.3.25** State Transition Diagram for L(M) = {∈}, r = ∈

## SOLVED PROBLEM 14

***Construct a DFA to accept strings of 0's, 1's and 2's, beginning with a '0' followed by odd number of 1's and ending with a '2'.***

**SOLUTION**

Let,

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

oufastupdates.com

Such that,

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

$\Sigma = \{0, 1, 2\}$

$q_0 = q_0$

$F = \{q_3\}$

The transition 'δ', for the DFA is shown in Fig. 2.3.26.

For L(M) = {11, 1110, 1001, ....} or L(M) = {W ∈ Σ*| W is divisible by 3}

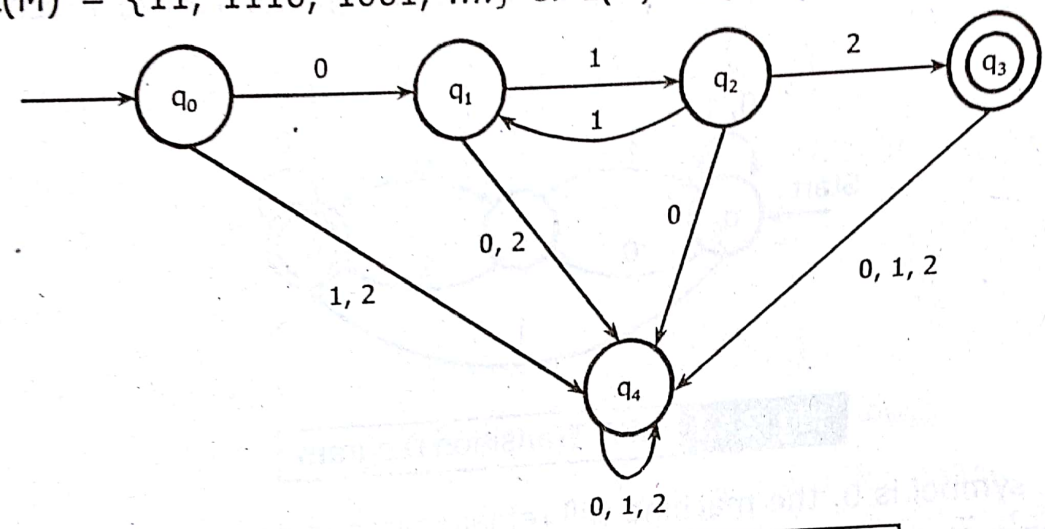



**Fig. 2.3.26** State Transition Diagram for L(M)

## NON-DETERMINISTIC FINITE AUTOMATA (NFA)

A *"Non-Deterministic"* Finite Automata (NFA) has capability to traverse more than one state at a time. Finite accepters are more complicated if we allow them to act Non-deterministically. We normally think of computers as completely deterministic and the element of choice seems out of place. Nevertheless, non-deterministic is a useful notation.

### ADVANTAGES

The advantages of non-determinism are,

(1) NFA can be smaller, easier to construct and understand than a DFA that accepts the same language.

(2) It is useful for proving some theorems.

(3) It gives a good introduction to non-determinism in more powerful computational models, where non-determinism plays an important role.

(4) Non-determinism is a kind of parallel computation where several processes can be running concurrently.

When NFA splits to follow several choices, that corresponds to a process 'forking' into several children, each proceeding separately, then the entire computation accepts, if atleast one of these accepts.

oufastupdates.com

## An Informal View of NFA

Like the DFA, an NFA also has a finite set of states as well as finite set of input alphabets (symbols). One is the start state and more than one state of accepting states. It also contains a transition function, which we call commonly as $\delta$. Except $\delta$, other quin tuples are same in both DFA as well as NFA. In the NFA, $\delta$ function takes a state and argument as input symbol but it returns a set of zero, one or more states where as in the DFA it returns exactly one state.

Consider the NFA transition diagram shown in the Fig. 2.4.1. The machine will start in $q_0$ state.

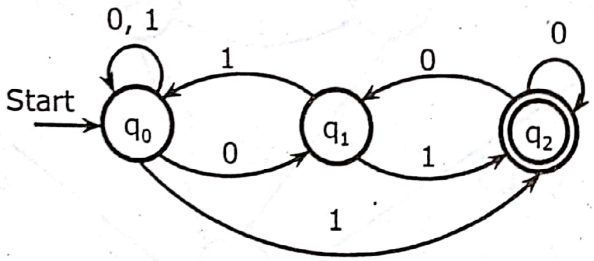



**Fig. 2.4.1** NFA Transition Diagram

If the input symbol is 0, the machine will remain same state as well as machine enters into $q_1$ state. Since there are two transitions on input symbol 0 from $q_0$ state. If the input symbol is 1 from table $q_0$ the machine will be remains same state $q_0$. Similarly from $q_1$ state, there is a multiple transitions on input symbol 1 i.e., $q_0$ and $q_2$ means machine can either enter into state $q_0$ or state $q_2$. From this discussion it is not possible to determine exactly in which state machine will be so, we say it is non-deterministic finite automaton (NFA).

**Table 2.4.1** Transition Table for the M/c Shown in Fig. 2.4.1

| Current | Representation of Input | |
|---|---|---|
| | 0 (Next State) | 1 (Next State) |
| $q_0$ | $\delta(q_0, 0) = \{q_0, q_1\}$ | $\delta(q_0, 1) = \{q_0, q_2\}$ |
| $q_1$ | $\delta(q_1, 0) = \{\phi\}$ | $\delta(q_1, 1) = \{q_0, q_2\}$ |
| $q_2$ | $\delta(q_2, 0) = q_1$ | $\delta(q_2, 1) = \{\phi\}$ |

Table 2.4.1 shows transition table i.e., processes inputs for the machine shown in Fig. 2.4.1.

Note that $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}q_0$ is the start state, $q_1$ is the final state and $\delta$ is the transition are as follows,

oufastupdates.com

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

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

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

$\delta(q_1, 1) = \{q_0, q_2\}$ [Group of subsets of Q i.e., $2^Q$ i.e., power set]

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

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

$\delta : \Sigma \rightarrow 2^Q$.

## Definition of NFA

A non-deterministic finite automata is a five-tupel machine,

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

Where,

Q = A finite set of states i.e., $(q_0, q_1, q_2, \ldots)$.

$\Sigma$ = A finite set of input symbols including the input symbol $\in$.

$\delta$ = A transition function that defines the rules for change of the state. It takes two arguments a state and the input symbol and returns a subset of output states i.e., more than one output state.

$q_0$ = An initial state or start state, $q_0 \in Q$.

F = The set of final states or accepting states, $F \subseteq Q$.

Like DFA, NFA is also represented as a transition graph.

The following NFA can be specified formally,

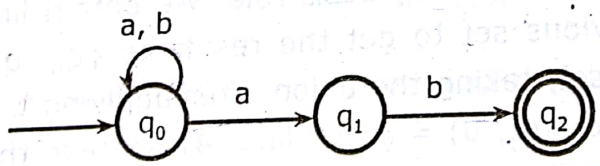



**Fig. 2.42** NFA Accepting all Strings that End with ab

As $(\{q_0, q_1, q_2\}, \{a, b\}\delta, q_0 \{q_2\})$ where the transition function $\delta$ is given the transition table as follows,

**Table 2.42** Transition Table

| State | Input ($\Sigma$) | |
|---|---|---|
| | a | b |
| $\rightarrow q_0$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| $q_1$ | $\phi$ | $\{q_2\}$ |
| $*q_2$ | $\phi$ | $\phi$ |

oufastupdates.com

## The Extended Transition Function

As for DFA's we need to extend the transition function $\delta$ of an NFA to a function $\delta^*$ that takes a state q and a string of input symbol w, and returns the set of states that the NFA is in if it starts in state q and process the string w. Formally, we define $\delta^*$ for an NFA's $M = (Q, \Sigma, \delta, q_0, F)$. We would like to define the language accepted by M, and for this, we need to extend the transiting function $\delta : Q \times \Sigma \rightarrow 2^Q$ to a function,

$\delta^* : Q \times \Sigma \rightarrow 2^Q$ is defined as follows,

$$\delta(q, \varepsilon) = \delta^*(q, \varepsilon) = q$$

$$\delta(q, wa) = \delta(\delta^* (q, w), q) = p_i$$

$$\delta(q, aw) = \delta(\delta^* (q, q), q) = p_j$$

Where,

$q \in Q$ $a \in S$, $w \in \Sigma^*$ and $p_i$ and $p_j$ are the set of states which are reachable from q.

**Example :** Let us use $\delta^*$ to describe the processing of input 00101 by the NFA of Fig. 2.4.2 whole summary for the following steps,

(1) $\delta^*(q, \varepsilon) = \{q\}$

(2) $\delta^*(q_0, 0) = \delta\{q_0, 0\} = \{q_0, q_1\}$

(3) $\delta^*(q_0, 00) = \delta\{q_0, 0\} \cup \delta \{q_1, 0\} = \{q_0, q_1\} \cup \{\phi\} = \{q_0, q_1\}$

(4) $\delta^*(q_0, 001) = \delta\{q_0, 1\} \cup \delta \{q_1, 0\} = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

(5) $\delta^*(q_0, 0010) = \delta\{q_0, 0\} \cup \delta \{q_2, 0\} = \{q_0, q_1\} \cup \{\phi\} = \{q_0, q_1\}$

(6) $\delta^*(q_0, 00101) = \delta\{q_0, 1\} \cup \delta \{q_1, 1\} = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

In the line (1) in the induction by basis rule. We obtain line (2) by applying $\delta$ to the single state $q_0$, i.e., previous set to get the result as $\{q_0, q_1\}$ line (3) is obtained by considering the previous set, taking the union after applying $\delta$ to them with input 0. i.e., $\delta (q_0, 0) = \{q_0, q_1\}$ where $\delta(q_1, 0) = \phi$. For line (4) we take the union of $\delta(q_0, 1) = \{q_0\}$ and $\delta (q_1, 1) = \{q_2\}$. The line (5) and (6) are similar way of line number (3) and (4).

## The Language of an NFA

A sequence of input symbols obtained from $\Sigma$ is called string. The set of all string recognized by an automaton is called language. Acceptance of language L for NFA $M = (Q, \Sigma, \delta, q_0, F)$.

The language (string w) is accepted by NFA can be defined in the formal notation as,

$$L(M) = \{w \mid \delta^* (q_0, w) \cap F \neq 0\}$$

That is, L(M) is the set of strings w w in $\Sigma^*$ such that $\delta^* (q_0, w)$ contains at least one accepting state.

oufastupdates.com

## Equivalent of DFA and NFA

For every non-deterministic finite automata (NFA) there exists an equivalent deterministic automata (DFA). The equivalence is defined in terms of language acceptance. Since a NFA is nothing but a finite automata in which zero, one or more transiting on an input symbol is permitted. We can always construct a DFA which will simulate all the moves of NFA on particular input symbol in parallel to get a DFA in which there will be exactly one transiting input symbol. Hence it will be a DFA equivalent to NFA.

Since the DFA equivalent to NFA is to simulate the moves of NFA in parallel, every state of DFA will be a combination of one or more states of NFA. Hence every state of DFA will be represented by some subset of set of states of NFA and therefore transformation of NFA to DFA is normally called as subset construction. Therefore if subset have n sates then the number of states of DFA will be equivalent to $2^n$, with initial state corresponding to the subset $\{q_0\}$.

Therefore transformation of NFA to DFA involves, finding all possible subsets of set of states of NFA and then considering each subset to be state of DFA an finding transition form it on every input symbol. But out of all states of DFA obtained in this way, if any states is not reachable form initial state on any possible input sequence, then such state does not play any role to decide the language to get accepted by a DFA. Such states are eliminated from the set of states.

This can be done as follows,

Let,

$M = (Q, \Sigma, \delta, q_0, F)$ be a NFA accepting language L.

We construct a DFA M' that can also accept the same language L.

Then a DFA M' is defined as follows,

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

Where,

$Q' = 2^Q$ (Power set of Q or all subset of Q) and all states of Q' are denoted by $[q_1, q_2, q_3, \ldots, q_n]$.

$\Sigma$ = Similar to NFA.

F' = Set of all subsets of Q' containing an element of F(i.e., final state of NFA).

$\delta'$ = Transition system.

oufastupdates.com

To define transition system, use the following steps,

**STEP 1 :** Draw an initial state (or) vertex with label $[q_0]$ of transition diagram $\delta'$.

**STEP 2 :** Take this state (i.e., $[q_0]$) and identify all next states from Q' on all differen given input symbols. Let us consider the next states as $[q_i, q_j, \ldots, q_k]$ on input symbo $0 \in \Sigma$ from state $[q_0]$.

Create a state labeled $[q_i, q_j, \ldots, q_k]$ and add to transition digraph $\delta'$ an edge from $[q_0]$ to $[q_i, q_j, \ldots, q_k]$ with labeled 0.

**STEP 3 :** Take any state of step 2 (or) previously defined state and find new next state (i.e., it does not exist already) from Q' on all different given input symbols.

Let us consider the next state as $[q_i, q_j, \ldots, q_k]$ on input symbol $0 \in \Sigma$ from state $[q_i, q_j, \ldots, q_k]$.

Compute $\delta'(q_i, 0), \delta'(q_j, 0), \ldots, \delta'(q_k\ 0)$ and then form a union of all $\delta'$, yielding the set of states $[q_i, q_j, \ldots, q_n]$.

Create a state with label $[q_i, q_j, \ldots, q_n]$ and add to transition digraph $\delta'$ and edge from $[q_i, q_j, \ldots, q_k]$ to $[q_i, q_j, \ldots, q_n]$ labeled 0.

**STEP 4 :** Repeat the step 3 until no new states are generated.

**STEP 5 :** Draw all states as a final states of, if it contain the final state of NFA.

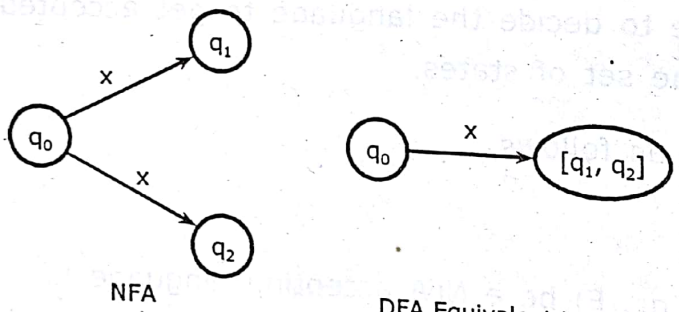
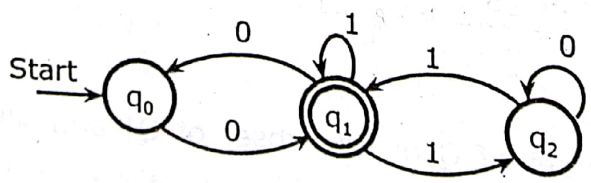



**Fig. 2.43** The DFA constructed from the NFA

## EXAMPLE PROBLEM 2.4

**Convert the following NFA to its equivalent DFA,**




**SOLUTION**

**STEP 1 :** $q_0$ is the start state of DFA.

So,

$$Q_D = \{[q_0]\}$$

oufastupdates.com

**STEP 2 :** Obtain new states from each state in $Q_D$ and corresponding transitions as follows,

Consider the state $[q_0]$, when input,

$$a = 0$$

$$\delta_D(q_0, 0) = \delta_N([q_0], 0)$$

$$= [q_1] \quad \text{... (2)}$$

When input,

$$a = 1$$

$$\delta_D(q_0, 1) = \delta_N([q_0], 1)$$

$$= \phi$$

Since the state obtained Eq. (2) is not in $Q_D$. So add this state $Q_D$ so that,

$$Q_D = \{[q_0], [q_1]\}$$

The corresponding transitions on a = 0 and a = 1 are shown below,

| | δ | Σ | |
|---|---|---|---|
| | | 0 | 1 |
| ↑ Q ↓ | $[q_0]$ | $q_1$ | $\phi$ |
| | $[q_1]$ | | |

Consider the state $[q_1]$, when input,

$$a = 0$$

$$\delta_D(q_1, 0) = \delta_N(q_1, 0)$$

$$= [q_0] \quad \text{... (3)}$$

When input,

$$a = 1$$

$$\delta_D(q_1, 1) = \delta_N(q_1, 1)$$

$$= [q_1, q_2] \quad \text{... (4)}$$

Since the state obtained Eq. (4) is not defined in $Q_D$. So add this state $Q_D$ so that,

$$Q_D = \{[q_0], [q_1], [q_1, q_2]\}$$

oufastupdates.com

And add the transitions on a = 0 and a = 1 as shown below,

| $\delta$ | $\Sigma$ | |
|---|---|---|
| | 0 | 1 |
| $[q_0]$ | $[q_1]$ | $[\phi]$ |
| $[q_1]$ | $[q_0]$ | $[q_1, q_2]$ |
| $[q_1, q_2]$ | | |

Consider the state $[q_1, q_2]$, when input,

$$a = 0$$

$$\delta_D([q_1, q_2]\ 0) = \delta_N([q_1, q_2], 0)$$

$$= \delta_N(q_1, 0) \cup \delta_N(q_2, 0)$$

$$= q_0 \cup q_2$$

$$= [q_0, q_2] \quad \text{... (5)}$$

When input,

$$a = 1$$

$$\delta([q_1, q_2]\ 1) = \delta_N([q_1, q_2], 1)$$

$$= \delta_N(q_1, 1) \cup \delta_N(q_2, 1)$$

$$= q_2 \cup q_1$$

$$= [q_1, q_2] \quad \text{... (6)}$$

Since the state obtained Eq. (5) is not defined in $Q_D$. So add this state $Q_D$ so that,

$$Q_D = \{[q_0], [q_1], [q_1, q_2], [q_0, q_2]\}$$

And add the transitions on a = 0 and a = 1 as shown below,

| | $\delta$ | $\Sigma$ | |
|---|---|---|---|
| | | 0 | 1 |
| | $[q_0]$ | $[q_1]$ | $[\phi]$ |
| ↑ | $[q_1]$ | $[q_0]$ | $[q_1, q_2]$ |
| Q | $[q_1, q_2]$ | $[q_0, q_2]$ | $[q_1, q_2]$ |
| ↓ | $q_0, q_2$ | | |

oufastupdates.com

Consider the state $[q_0, q_2]$ when input,

$$a = 0$$

$$\delta_D([q_0, q_2], 0) = \delta_N([q_0, q_2]), 0)$$

$$= \delta_N(q_0, 0) \cup \delta_N(q_2, 0)$$

$$= q_1 \cup q_2$$

$$= [q_1, q_2] \quad \ldots (7)$$

When input,

$$a = 1$$

$$\delta([q_0, q_2]\ 1) = \delta_N([q_0, q_2]), 1)$$

$$= \delta_N(q_0, q_1) \cup \delta_N(q_2, 1)$$

$$= \phi \cup q_1$$

$$= [q_1] \quad \ldots (8)$$

Since the state obtained are not an new states (already present in $Q_D$) so do not add these states into $Q_D$. But, the transitions on a = 0 and a = 1 should be added to the transition table as follows. This is the final transition table and the final DFA is shown in Fig. 2.4.4.

| δ | Σ | |
|---|---|---|
| | 0 | 1 |
| → $q_0$ | $[q_1]$ | $\phi$ |
| *$[q_1]$ | $[q_0]$ | $[q_1, q_2]$ |
| *$[q_1, q_2]$ | $[q_0, q_2]$ | $[q_1]$ |
| $[q_0, q_2]$ | $[q_1, q_2]$ | $[q_1]$ |

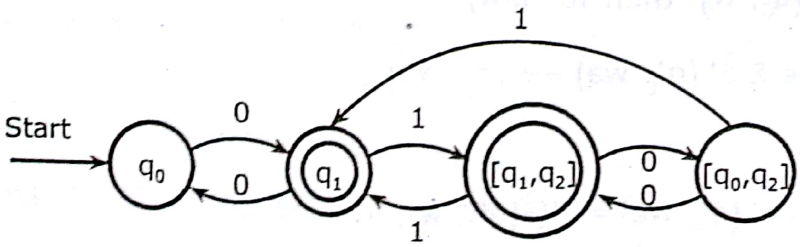



Fig. 2.4.4 The Final DFA

oufastupdates.com

## Differences between NFA and DFA

**Table 2.4.3** Differences between NFA and DFA

| S.No. | Non-Deterministic Finite Automata (NFA) | Deterministic Finite Automata (DFA) |
|---|---|---|
| (1) | More than one transitions are possible from each state on the same input symbol. | There is almost one transition from each state on any input symbol. |
| (2) | It can accept the input symbol $\epsilon$ i.e., empty input symbol. | It does not accept empty input symbol. |
| (3) | It takes more time to recognize input string. | It is faster than NFA. |
| (4) | A NFA can be smaller than DFA. | A DFA can be much bigger than NFA. |
| (5) | An NFA accepts a string if there is some sequence of possible moves that will put the machine in a final state at the end of the string. It rejects string only if there is no possible sequence of moves by which a final state can be reached. | A DFA accepts a string if the automaton is in one of its final states when the end of the string is reached. Otherwise it rejects the string. |

### SOLVED PROBLEM 1

**Design an NFA that accepts set of all strings over {0, 1} that have at least two consecutive 0's or 1's.**

**SOLUTION**

NFA, M, should be designed to accept the language.

i.e., $L(M) = \{00, 11, 0110, 1001, 10110101, \ldots\}$.

Where,

M is the NFA.

Consider,

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

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

$q_0$ = Initial state

oufastupdates.com

**Transition Diagram**

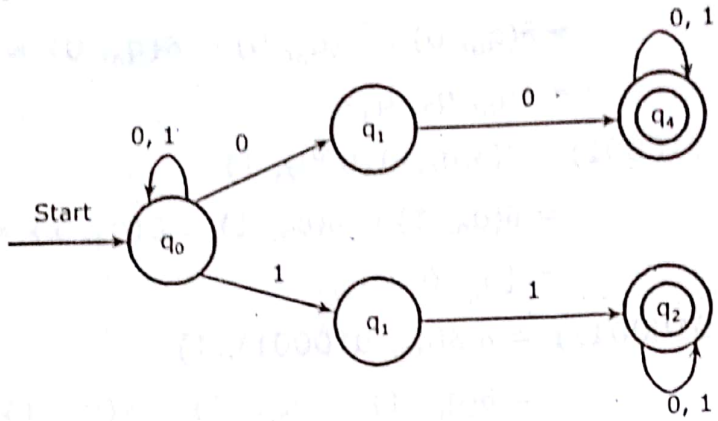



**Fig. 2.4.6** State Diagram to Represent L

**Transition Table**

**Table 2.4.4** State Transition Table

| State \ Input | Input Symbol 0 | Input Symbol 1 |
|---|---|---|
| → $q_0$ | $\{q_0, q_3\}$ | $\{q_0, q_1\}$ |
| $q_1$ | $\phi$ | $\{q_2\}$ |
| $*q_2$ | $\{q_2\}$ | $\{q_2\}$ |
| $q_3$ | $\{q_4\}$ | $\{\phi\}$ |
| $*q_4$ | $\{q_4\}$ | $\{q_4\}$ |

**NFA Action for the Input String**

(i) To show that the string 0100011 is accepted by the NFA.

Using extended transition function,

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

$$= \{q_0, q_3\} \text{ and } \delta(q_0, q_1) = \{q_0, q_1\}$$

$$\delta(q_0, 01) = \delta(\delta(q_0, 0), 1)$$

$$= \delta(q_0, 1) \cup \delta(q_3, 1) = \{q_0, q_1\} \cup \{\phi\}$$

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

$$\delta(q_0, 010) = \delta(\delta(q_0, 01), 0)$$

$$= \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_3\} \cup \{\phi\}$$

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

$$\delta(q_0, 0100) = \delta(\delta(q_0, 010), 0)$$

$$= \delta(q_0, 0) \cup \delta(q_3, 0) = \{q_0, q_3\} \cup \{q_4\}$$

$$= \{q_0, q_3, q_4\}.$$

oufastupdates.com

$$\delta(q_0, 01000) = \delta(\delta(q_0, 0100), 0)$$
$$= \delta(q_0, 0) \cup \delta(q_3, 0) \cup \delta\{q_4, 0\} = \{q_0, q_3\} \cup \{q_4\} \cup \{q_4\}$$
$$= \{q_0, q_3, q_4\}.$$

$$\delta(q_0, 010001) = \delta(\delta(q_0, 01000), 1)$$
$$= \delta(q_0, 1) \cup \delta(q_3, 1) \cup \delta\{q_4, 1\} = \{q_0, q_1\} \cup \{\phi\} \cup \{q_4\}$$
$$= \{q_0, q_1, q_4\}.$$

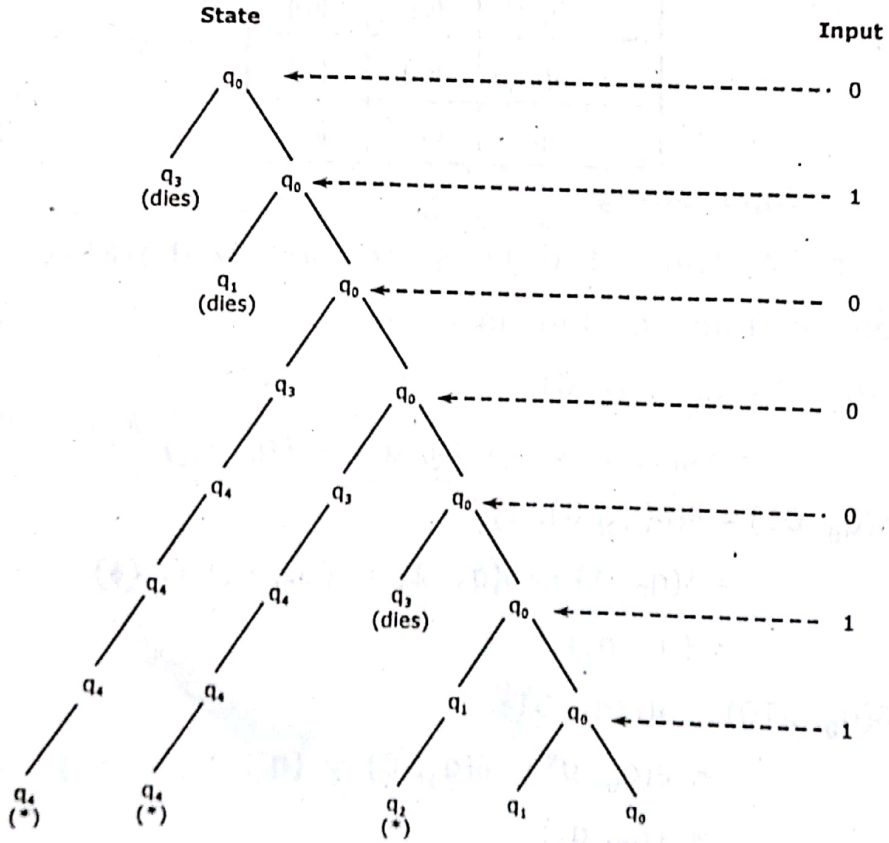
$$\delta(q_0, 0100011) = \delta(\delta(q_0, 010001), 1)$$
$$= \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta\{q_4, 1\}$$
$$= \{q_0, q_1\} \cup \{q_2\} \cup \{q_4\}$$

Now, let $P = \{q_0, q_1, q_2, q_4\}$.

Since P contains $q_4$ and $q_2$ (which are final states), clearly 0100011 is accepted by L(M).

i.e., $$\delta(q_0, 0100011) = \{q_0, q_1, q_2, q_4\} \cap F$$
$$= \{q_0, q_1, q_2, q_4\} \cap \{q_2, q_4\}$$
$$= \{q_2, q_4\}$$
$$= \{\phi\} \Rightarrow 0100011 \text{ is accepted.}$$

Using tree state diagram for input = 0100011.




(*) Indicates Accepting paths in NFA Tree.

**Fig. 2.4.7** Tree State Diagram for the String 0100011

Since there exists a final path, 0100011 is accepted.

oufastupdates.com

(ii) To show that the string 0101 is rejected by NFA.

Using extended function,

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

$$= \{q_0, q_3\}, \delta(q_0, 1) = \{q_0, q_1\}.$$

$$\delta(q_0, 01) = \delta(\delta(q_0, 0), 1)$$

$$= \delta(q_0, 1) \cup \delta(q_3, 1) = \{q_0, q_1\} \cup \{\phi\}$$

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

$$\delta(q_0, 010) = \delta(\delta(q_0, 01), 0)$$

$$= \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_3\} \cup \{\phi\}$$

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

$$\delta(q_0, 0101) = \delta(\delta(q_0, 010), 1)$$

$$= \delta(q_0, 1) \cup \delta(q_3, 1) = \{q_0, q_1\} \cup \{\phi\}$$

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

Let $P = \{q_0, q_1\}$,

Since P contains no final states, hence 0101 is rejected by the NFA.

i.e., $\delta(q_0, 0101) = \{q_0, q_1) \cap F$

$$= \{q_0, q_1) \cap \{q_2, q_4\}$$

$$= \phi \Rightarrow 0101 \text{ is rejected by NFA.}$$

Using tree state diagram for the input 0101,

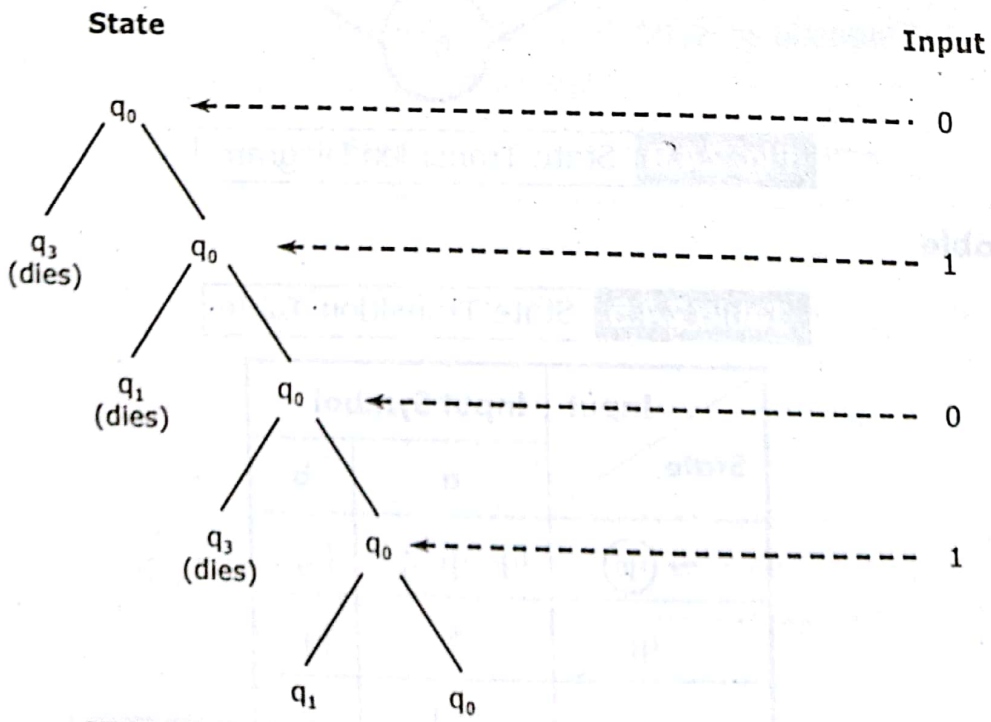



**Fig. 2.4.8** Tree State Diagram for the String 0101

Since no final paths exist, 0101 is rejected by NFA.

oufastupdates.com

**Design an NFA to accept set of all strings starting with 'a' followed by 'a' or 'b' and ending with a or any number of b's.**

**SOLUTION**

Here, it is required to design an NFA for the regular expression.

r = a* (ab + a + ba)(bb)*. In other words, NFA, M should be designed to accept the language of r.

i.e., L(M) = {a, ab, aaa, abbbb, ....}.

Consider,

$\Sigma = \{0, 1\}$, $Q = \{q_0, q_1, q_2, q_3,\}$, $q_0$ = initial state, $\{q_3\}$ = final state and $\delta$ is given by,

**Transition Diagram**

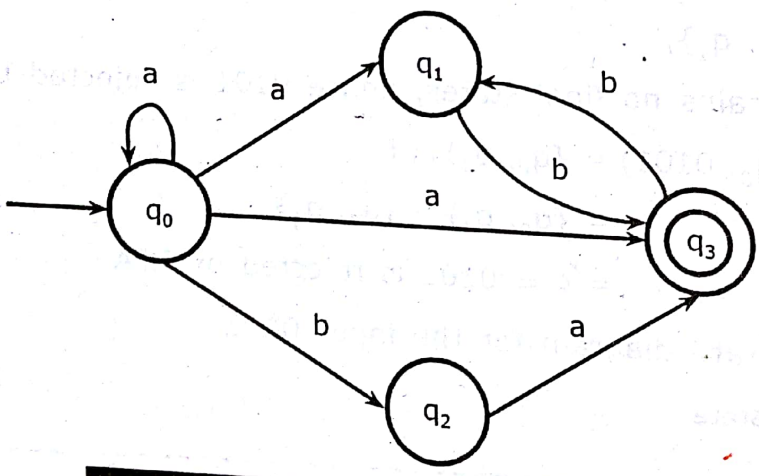



**Fig. 2.4.9** State Transition Diagram

**Transition Table**

**Table 2.4.5** State Transition Table

| Input / State | Input Symbol | |
|---|---|---|
| | a | b |
| → $q_0$ | $\{q_0, q_1, q_3\}$ | $\{q_2\}$ |
| $q_1$ | $\phi$ | $\{q_3\}$ |
| $q_2$ | $\{q_3\}$ | $\phi$ |
| *$q_3$ | $\phi$ | $\{q_1\}$ |

oufastupdates.com

**NFA Action for the Input String**

To show that the string abbbb is accepted by NFA.

(i) Using extended transition function,

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

$$\delta(q_0, ab) = \delta(\delta(q_0, a), b)$$
$$= \delta(q_0, b) \cup \delta(q_1, b) \cup \delta\{q_3, b\}$$
$$= \{q_2\} \cup \{q_3\} \cup \{q_1\}$$
$$= \{q_1, q_2, q_3\}.$$

$$\delta(q_0, abb) = \delta(\delta(q_0, ab), b)$$
$$= \delta(q_1, b) \cup \delta(q_2, b) \cup \delta\{q_3, b\}$$
$$= \{q_3\} \cup \{\phi\} \cup \{q_1\}$$
$$= \{q_3, q_1\}.$$

$$\delta(q_0, abbb) = \delta(\delta(q_0, abb), b)$$
$$= \delta(q_3, b) \cup \delta(q_1, b) = \{q_1, q_3\}$$

$$\delta(q_0, abbbb) = \delta(\delta(q_0, abbb), b)$$
$$= \delta(q_3, b) \cup \delta(q_1, b) = \{q_1, q_3\}$$

Let $P = \{q_1, q_3\}$. Since P contains the final state $q_3$.

Hence abbbb is accepted by the NFA.

i.e., $d(q_0, abbbb) = \{q_1, q_3\} \cap F$

$$= \{q_1, q_3) \cap \{q_3\} = \{q_3\} \Rightarrow \text{accepted.}$$

(ii) Using tree state diagram for input abbbb.

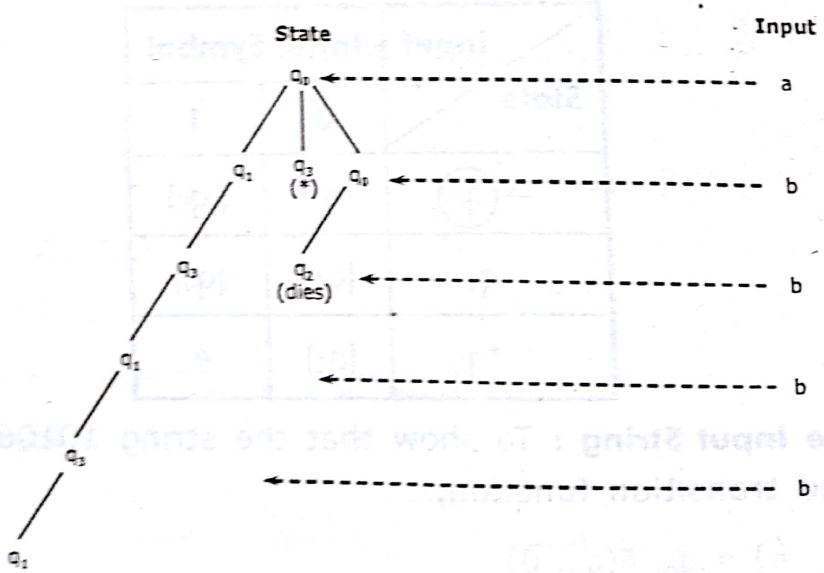



**Fig. 2.4.10** Tree State Diagram for the String abbbb

Since there exists a final path in the tree, hence abbbb is accepted.

oufastupdates.com

**Design an NFA that accepts a set of all strings ending in 00.**

**SOLUTION**

Here it is required to design an NFA for the regular expression,

$$r = (1 + 0)^* 00$$

i.e., L(M) = {00, 1100, 100, 000, 111000, ...}.

Consider,

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

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

$q_0$ = initial state.

F = $q_2$ is the final state and $\delta$ is given by,

**Transition Diagram**

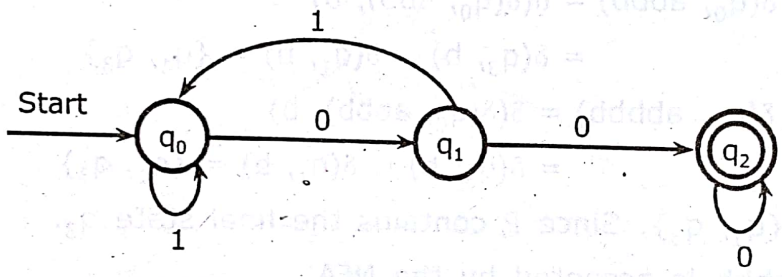



**Fig. 2.4.11** State Transition Diagram

**Transition Table**

**Table 2.4.6** State Transition Table

| Input / State | Input Symbol 0 | Input Symbol 1 |
|---|---|---|
| → $q_0$ | $\{q_1\}$ | $\{q_0\}$ |
| $q_1$ | $\{q_2\}$ | $\{q_0\}$ |
| *$q_2$ | $\{q_2\}$ | $\phi$ |

**NFA Action for the Input String :** To show that the string 10100 is accepted by NFA by using extended transition function,

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

$$= q_1 \text{ and } \delta\{q_0, 1\}$$

$$= q_0$$

oufastupdates.com

$$\delta(q_0, 10) = \delta(\delta(q_0, 1), 0)$$
$$= \delta(q_0, 0)$$
$$= \{q_1\}.$$
$$\delta(q_0, 101) = \delta(\delta(q_0, 10), 1)$$
$$= \delta(q_1, 1)$$
$$= \{q_0\}.$$
$$\delta(q_0, 1010) = \delta(\delta(q_0, 101), 0)$$
$$= \delta(q_0, 0)$$
$$= \{q_1\}.$$
$$\delta(q_0, 10100) = \delta(\delta(q_0, 1010), 0)$$
$$= \delta(q_1, 0)$$
$$= \{q_2\}.$$

Since $q_2$ is the final state, 10100 is accepted by the NFA.

i.e., $\delta(q_0, 10100) = \{q_2\} \cap F$

$$= \{q_2\} \cap \{q_2\}$$
$$= \{q_2\} \neq \phi \text{ hence accepted.}$$

## SOLVED PROBLEM 4

**Design an NFA to accept all the strings over the alphabet {a, b} ending in aba.**

**SOLUTION**

We are required to design an NFA for the regular expression, r = (a + b)* aba. In other words, NFA, M should be designed to accept the language of r, i.e., L(M) = {aba, aaaba, ababa, abababa, ....}.

Consider,

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

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

$q_0$ = Initial state.

$q_3$ = The final state and $\delta$ is given by,

**Transition Diagram**

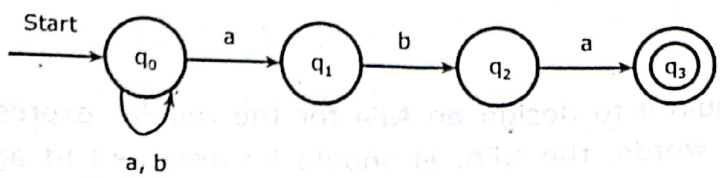



**Fig. 2.4.12** State Transition Diagram

oufastupdates.com

**Transition Table**

**Table 2.4.7** State Transition Table

| State \ Input | Input Symbol a | Input Symbol b |
|---|---|---|
| → $q_0$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| $q_1$ | $\phi$ | $\{q_2\}$ |
| $q_2$ | $\{q_3\}$ | $\phi$ |
| $*q_3$ | $\phi$ | $\phi$ |

**NFA Action for the Input String :** To show that ababa is accepted by NFA using tree state diagram.

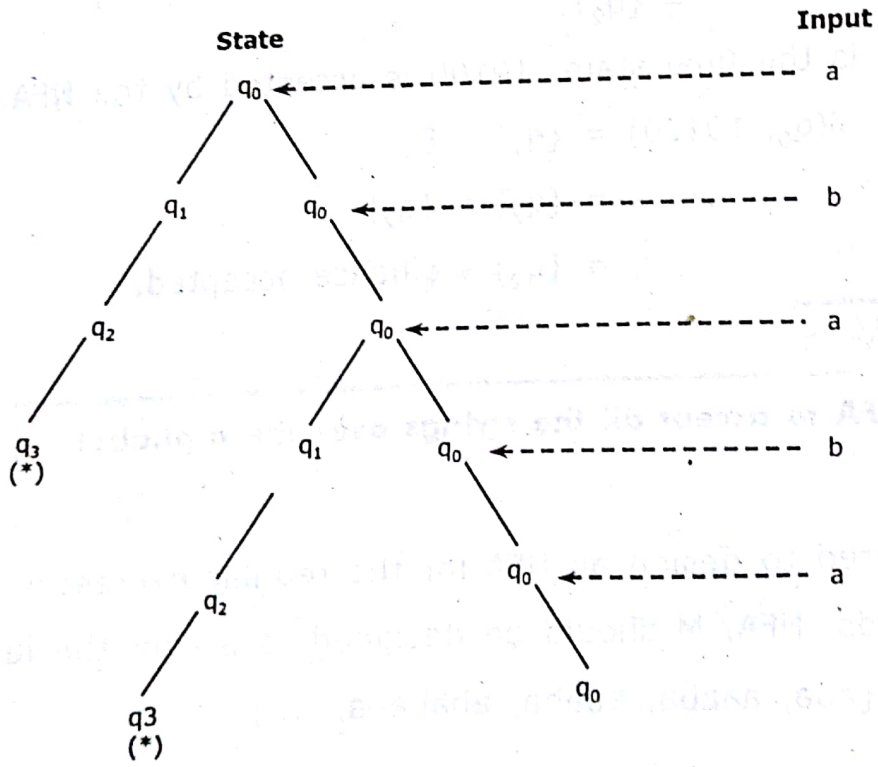



**Fig. 2.4.13** Tree State Diagram for the String ababa

Since there exists a path in the tree state diagram, ababa is accepted.

oufastupdates.com

**Construct an NFA whose language consists of all the strings over {0, ..., 9}, ending in either 1102, 0102, or 1110.**

**SOLUTION**

The NFA to accept all the strings ending in either 1102, 0102 or 1110 is given in Fig. 2.4.16.

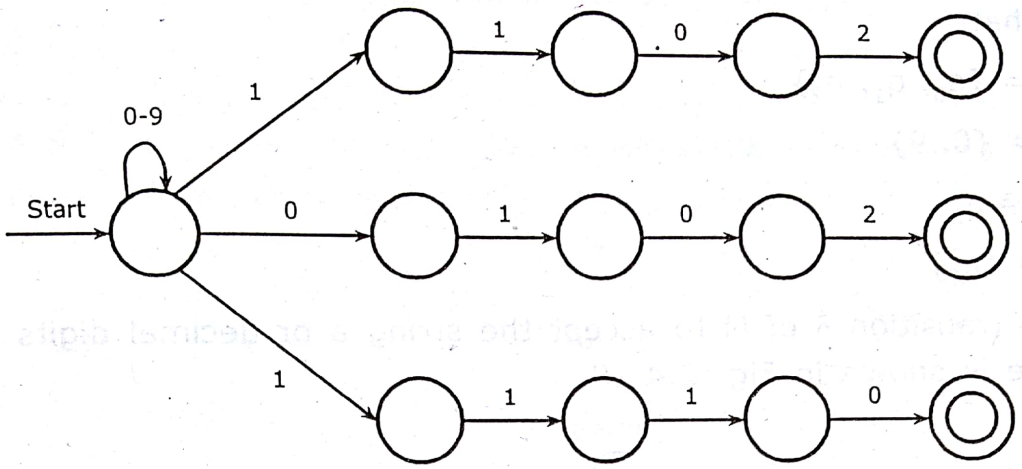



**Fig. 2.4.16** State Transition Diagram of NFA for L(M) = (0 -9)* (1102 + 0102 + 1110)

## SOLVED PROBLEM 8

**Construct an NFA to accept the string a or ab.**

**SOLUTION**

Let,

$M = (Q, \Sigma, q_0, \delta, F)$ be the NFA.

Such that,

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

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

$q_0 = q_0$

$F = \{q_2\}$

The transition $\delta$ of M to accept the string a or ab is shown in Fig. 2.4.17.

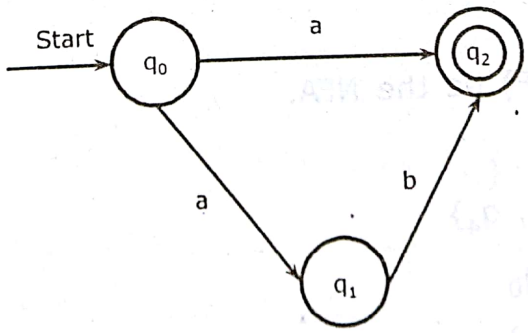



**Fig. 2.4.17** Transition Diagram, of NFA for r = a + ab

oufastupdates.com

The transition δ of NFA to accept the all strings of a's and b's, which are not multiples of 5, is shown in Fig. 2.4.20.

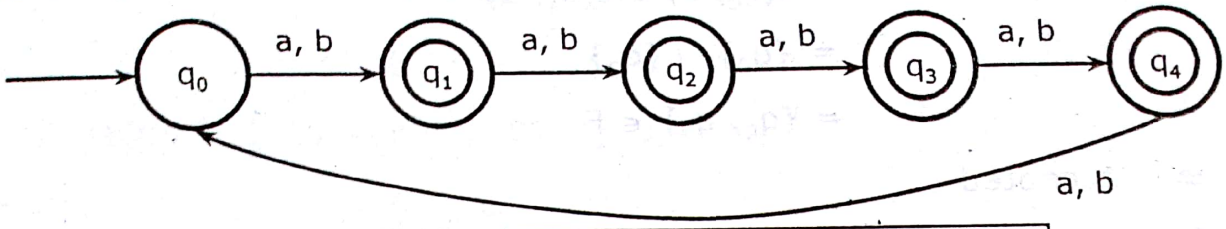



**Fig. 2.4.19** NFA to Accept Words of |w| mod 5 ≠ 0

## LVED PROBLEM 11

**Consider an NFA given below in Fig. 2.4.20. Check whether the strings 001, 011101, 01110, 010 are accepted by the machine or not,**

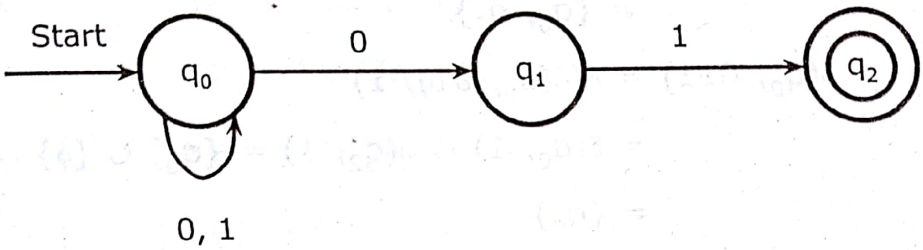



**Fig. 2.4.20** State Transition Diagram

**SOLUTION**

The transition function δ is defined by,

**Table 2.4.9** State Transition Table

| State \ Input | Input Symbol 0 | Input Symbol 1 |
|---|---|---|
| → $q_0$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| $q_1$ | $\phi$ | $q_2$ |
| * $q_2$ | $\phi$ | $\phi$ |

Given $q_0$ = initial state and F = $\{q_2\}$,

(i) Input = 001,

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

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

$$= \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta\{q_3, 0\}$$

$$= \{q_2\} \cup \{q_3\} \cup \{q_1\}$$

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

oufastupdates.com

$$\delta(q_0, 001) = \delta(\delta(q_0, 00), 1)$$
$$= \delta(q_0, 1) \cup \delta(q_1, 1)$$
$$= \{q_0\} \cup \{q_2\}$$
$$= \{q_0, q_2\} \in F.$$

$\Rightarrow$ accepted

(ii) Input = 011101

$$\delta(q_0, 01) = \delta(\delta(q_0, 0), 1)$$
$$= \delta(q_0, 1) \cup \delta(q_1, 1)$$
$$= \{q_0\} \cup \{q_2\}$$
$$= \{q_0, q_2\}$$
$$\delta(q_0, 011) = \delta(\delta(q_0, 01), 1)$$
$$= \delta(q_0, 1) \cup \delta(q_2, 1) = \{q_0\} \cup \{\phi\}$$
$$= \{q_0\}$$
$$\delta(q_0, 0111) = \delta(\delta(q_0, 011), 1)$$
$$= \delta(q_0, 1)$$
$$= \{q_0\}$$
$$\delta(q_0, 01110) = \delta(\delta(q_0, 0111), 0)$$
$$= \delta(q_0, 0)$$
$$= \{q_0, q_1\}$$
$$\delta(q_0, 01101) = \delta(\delta(q_0, 01110), 1)$$
$$= \delta(q_0, 1) \cup \delta(q_1, 1)$$
$$= \{q_0\} \cup \{q_2\}$$
$$= \{q_0, q_2\} \in F$$

$\Rightarrow$ Accepted

(iii) Input = 01110,

$$\delta(q_0, 01) = \delta(\delta(q_0, 0), 1)$$
$$= \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\}$$
$$= \{q_0, q_2\}$$
$$\delta(q_0, 011) = \delta(\delta(q_0, 01), 1)$$
$$= \delta(q_0, 1) \cup \delta(q_2, 1) = \{q_0\} \cup \{\phi\}$$
$$= \{q_0\}$$

oufastupdates.com

$$\delta(q_0, 0111) = \delta(\delta(q_0, 011), 1)$$
$$= \delta(q_0, 1)$$
$$= \{q_0\}$$
$$\delta(q_0, 01110) = \delta(\delta(q_0, 0111), 0)$$
$$= \delta(q_0, 0)$$
$$= \{q_0, q_1\} \notin F$$

$\Rightarrow$ Not accepted

(iv) Input = 010,

$$\delta(q_0, 01) = \delta(\delta(q_0, 0), 1)$$
$$= \delta(q_0, 1) \cup \delta(q_1, 1)$$
$$= \{q_0\} \cup \{q_2\}$$
$$= \{q_0, q_2\}$$
$$\delta(q_0, 010) = \delta(\delta(q_0, 01), 0)$$
$$= \delta(q_0, 0) \cup \delta(q_2, 0)$$
$$= \{q_0, q_1\} \cup \{\phi\}$$
$$= \{q_0, q_1\} \notin F$$

$\Rightarrow$ Not accepted

oufastupdates.com

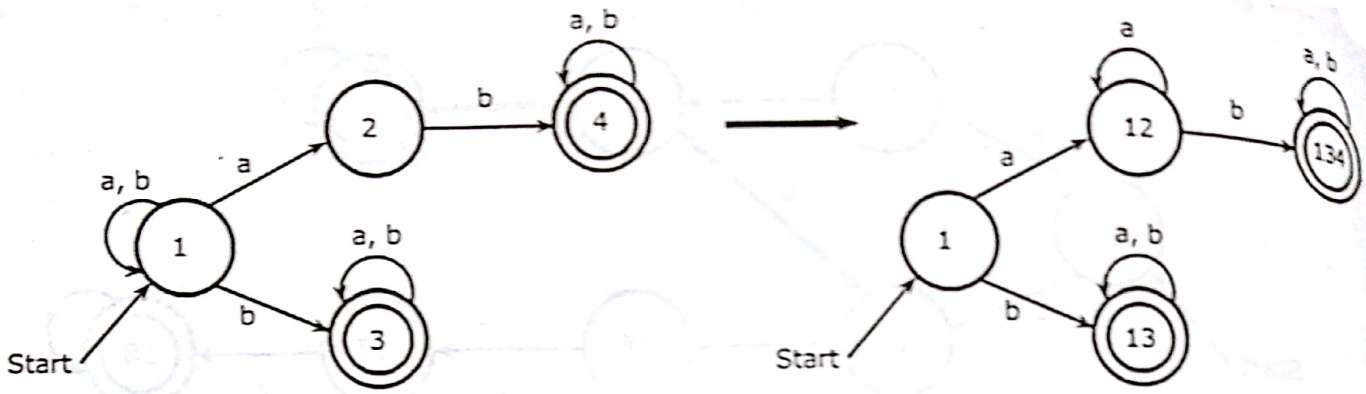
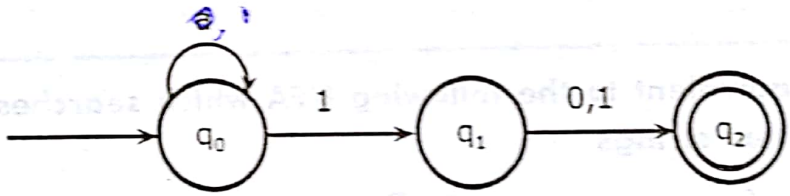



Fig. 2.5.5

## SOLVED PROBLEM 2

**Find DFA for the following NFA,**




**SOLUTION**

We start from $A = \{q_0\}$ (only one start state)

If we get 0, we can only go to the state $q_0$

If we get 1, we can go to $q_0$ or to $q_1$. We represent this by going to the state,

$$B = \{q_0, q_1\} = \delta_D(A, 1)$$

From B, if we get 0, we can go to $q_0$ or to $q_2$; we go to the state,

$$C = \{q_0, q_2\} = \delta_D(B, 0)$$

From B, if we get 1, we can go to $q_0$ or $q_1$ or $q_2$; we go to the state,

$$D = \{q_0, q_1, q_2\} = \delta_D(B, 1)$$

We get the following automaton,

$$A = \{q_0\}$$

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

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

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

oufastupdates.com

**Table 2.5.1**

| | 0 | 1 |
|---|---|---|
| → A | A | B |
| B | D | C |
| *C | A | B |
| *D | C | D |

Same automaton, as a transition system,

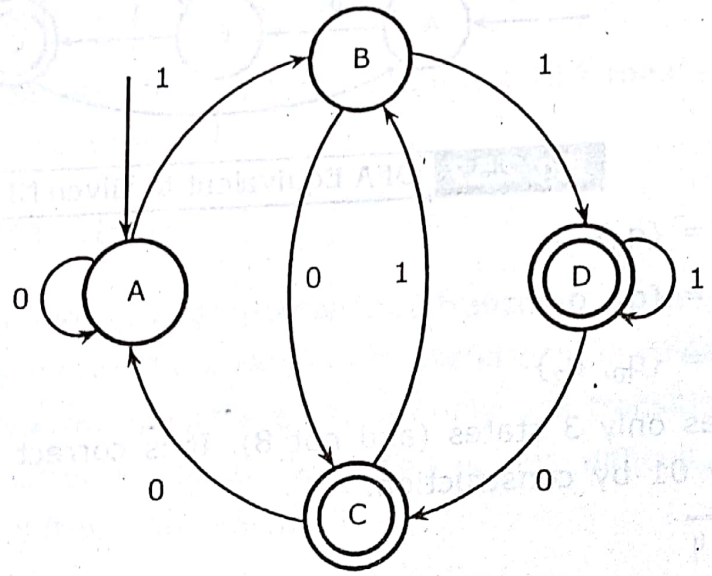
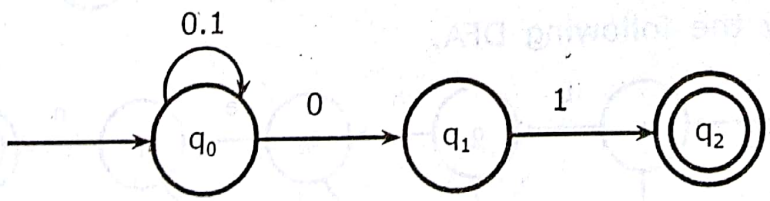



**Fig. 2.5.6** DFA for the Given NFA

The DFA "remembers" the last two bits seen and accepts if the next-to-last bit is 1,

## SOLVED PROBLEM 3

**Find the DFA equivalent to the following NFA where words ending by 01**




**SOLUTION**

The new states are,

$A = \{q_0\}$

$B = \{q_0, q_1\}$

$C = \{q_0, q_2\}$

oufastupdates.com

**Table 2.5.2**

| | 0 | 1 |
|---|---|---|
| → A | B | A |
| B | B | C |
| *C | B | A |

The DFA is,

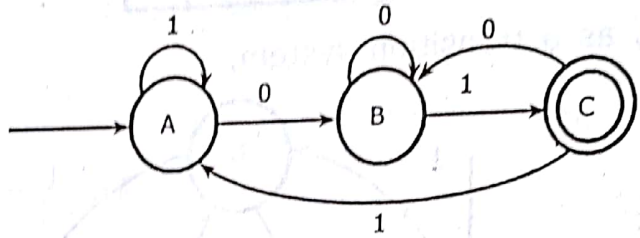



**Fig. 2.5.7** DFA Equivalent to Given NFA

$A = \{q_0\}$

$B = \{q_0, q_1\}$

$C = \{q_0, q_2\}$

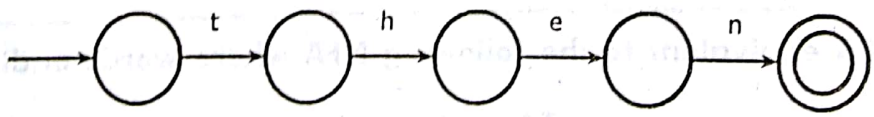
This DFA has only 3 states (and not 8). It is correct i.e., accepts only the word ending by 01 by construction.

## SOLVED PROBLEM 4

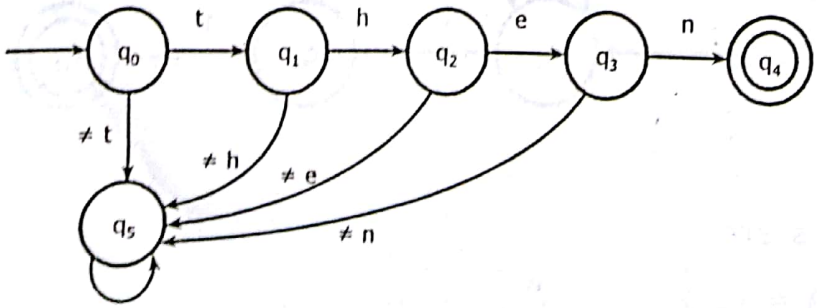
**Draw the NFA and DFA for the string.**

**SOLUTION**

By applying the subset construction to the NFA,




We get exactly the following DFA,




For this NFA, $\delta$ is a partial function,

With a "stop" or "dead" stage $q_5 = \theta$.

oufastupdates.com

## FINITE AUTOMATA WITH EPSILON TRANSITIONS

One of the objectives of this chapter is to show that (∈) there is one to one correspondence between regular languages and the finite automata. We are going to do that by showing that a finite automaton can be constructed from a given regular expression by combining simpler FAs using union, concatenation and Kleene star operation.

These operations of FAs can be described conveniently if ε transitions are used. Basically an NFA with ∈ transition is an NFA but can respond to an empty string ε and move to the next state. Here we are going to formally defined NFA with ε transitions (abbreviated as NFA ∈) and see some examples.

As we are going to see later, for any NFA ε there is a NFA (hence DFA) which accepts the same language and vice versa.

### Uses of ε-Transitions

An ^ transition (or) ε-moves is a spontaneous transition as the automaton changes its state without using any input. The ε-moves are useful only in NFAs. Because, in a DFA the transitions must be deterministic, there can be only one transition out of a state on any input symbol. The construction of the few DFA's is very difficult. One problem could be that the final state is difficult to reach.

For example we cannot construct a DFA or an NFA that accepts any number of a's followed by any number of b's without using ∈ moves.

In this case, we can first construct an NFA with ε-moves that reaches the final state without reading any input symbol. Then, convert the NFA with ε-moves to NFA without ∈ moves which can then be converted into a DFA.

The NFA with ε-moves accepting the language, $L = \{a^n b^n \mid n \geq 0\}$ is shown in Fig. 2.6.1.

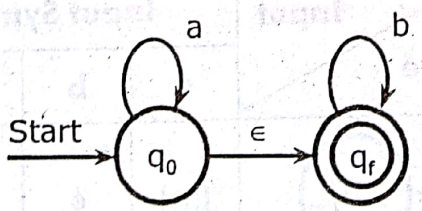



**Fig. 2.6.1** NFA With ∈ Moves for $L = \{a^n b^n \mid n \geq 0\}$

The NFA with ε-moves can be converted into a DFA. Thus, NFA with ε-moves are useful in designing an automaton that was not possible without ε-moves.

oufastupdates.com

## The Formal Notation for an ε-NFA

A non deterministic finite automata, NFA with ε-transitions, M is given by 5 tuples,

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

Where,

$Q$ = A finite set of states i.e., $q_0, q_1, q_2, \ldots\ldots$

$\Sigma$ = A finite set of input symbols.

$\delta$ = A transition function that defines the rules for change of the state. It maps from $Q \times (\Sigma \cup \epsilon)$ to $2^Q$.

$q_0$ = In initial state, $q_0 \in Q$.

$F$ = The set of final state, $F \subseteq Q$.

Consider,

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

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

$q_0$ = Initial state

$F = \{q_2\}$

The transition $\delta$ of NFA $\epsilon$ is shown in Fig. 2.6.2.

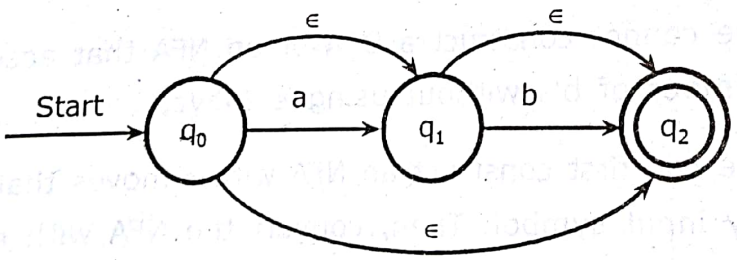



**Fig. 2.6.2** State Transition Diagram of NFA with ϵ

**Table 2.61** State Transition Table NFA.with ϵ

| State \ Input | Input Symbol | | |
|---|---|---|---|
| | a | b | ϵ |
| → $q_0$ | $\{q_1\}$ | $\phi$ | $\{q_1, q_2\}$ |
| $q_1$ | $\phi$ | $\{q_2\}$ | $\{q_2\}$ |
| * $q_2$ | $\phi$ | $\phi$ | $\phi$ |

oufastupdates.com

## Epsilon Closures

We should proceed to provide formal definition of an extendended transition function for ε-NFA, which leads to the definition of acceptance of strings and languages by this automata.

**Definition :** ε-Closure of a state q by following all transitions out of q that are labelled ε, finding every state can be reached from q along any path whose arcs are labelled ε. Formally, we define ε-Closure (q) recursively as,

**(1) Basis :** ε-Closure (q) = q for every q ε Q.

**(2) Induction :** If state p is in ε-Closure (q) i.e., ε Closure (q) = p and there is a transition from state p to state r labelled ε i.e., δ (p, ε) = r then r is in ε-Closure (q) i.e., ε Closure (q) = r. Therefore ε-Closure (q) = {p, r} for the Fig. 2.6.3 ε-NFA shows the ε-Closure (q) for each q ε Q is,

$$\varepsilon\text{-Closure } (q_0) = \{q_0, q_1, q_2\}$$

$$\varepsilon\text{-Closure } (q_1) = \{q_1, q_2\}$$

$$\varepsilon\text{-Closure } (q_2) = \{q_2\}$$

**Example**

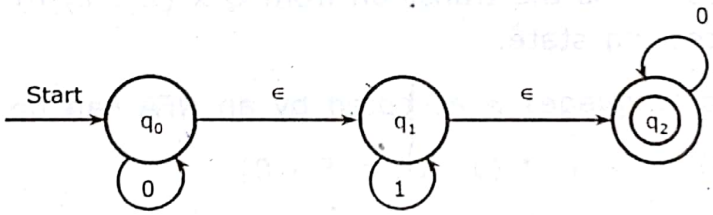



**Fig. 2.6.3** State Transition Diagram of NFA with ∈

**(1) To Compute ε-Closure ({$q_0$})**

$\{q_0\} \subseteq \varepsilon\text{-closure } (\{q_0\})$; $[\because \delta(q_0, \in) = q_0 \Rightarrow \delta (q_0, \in) \subseteq \varepsilon\text{-closure } (\{q_0\})]$

$\{q_0, q_1\} \subseteq \varepsilon\text{-closure } (\{q_0\})$; $[\because \delta(q_0, \in) = q_1]$

$\{q_0, q_1, q_2\} \subseteq \varepsilon\text{-closure } (\{q_0\})$; $[\because \delta(q_1, \in) = q_2]$

$\{q_0, q_1, q_2\} \subseteq \varepsilon\text{-closure } (\{q_0\})$; $[\because \delta(q_2, \in) = \phi]$

Hence, the process of generating ∈-closure terminates and ε-closure ({$q_0$}) = {$q_0$, $q_1$, $q_2$}.

**(2) To Compute ∈-closure ({$q_1$})**

$\{q_0\} \subseteq \varepsilon\text{-closure } (\{q_1\})$; $[\because \delta(q_1, \varepsilon) = q_1 \Rightarrow \delta (q_1, \varepsilon) \subseteq \in\text{-closure } (\{q_1\})]$

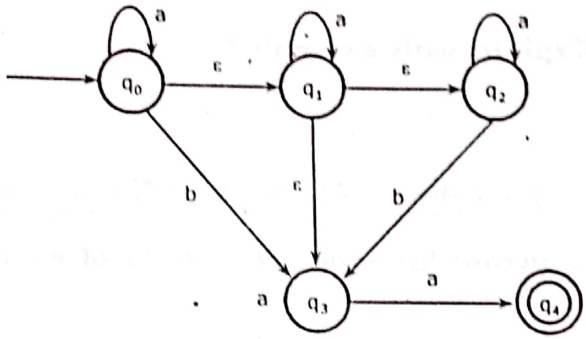
$\{q_1, q_2\} \subseteq \varepsilon\text{-closure } (\{q_1\})$; $[\because \delta(q_1, \varepsilon) = q_2]$

$\{q_1, q_2\} \subseteq \varepsilon\text{-closure } (\{q_1\})$; $[\because \delta(q_2, \varepsilon) = \phi]$

Hence, the process of generating ε-closure ({$q_1$}).

oufastupdates.com

**Obtain the ε closure on the following NFA**




**SOLUTION**

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

ε-closure $(q_1) = \{q_1, q_2, q_3\}$

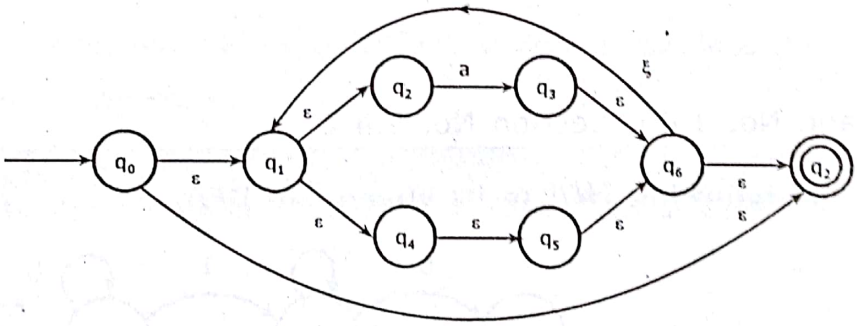
ε-closure $(q_2) = \{q_2\}$

ε-closure $(q_3) = \{q_3\}$

ε-closure $(q_4) = \{q_4\}$

## SOLVED PROBLEM 7

**Obtain the ε closure on the following NFA**




**SOLUTION**

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

ε-closure $(q_1) = \{q_1, q_2, q_4\}$

ε-closure $(q_2) = \{q_2\}$

ε-closure $(q_3) = \{q_3, q_2, q_1, q_6, q_4, q_7\}$

ε-closure $(q_4) = \{q_4\}$

ε-closure $(q_5) = \{q_1, q_2, q_4, q_5, q_6, q_7\}$

ε-closure $(q_6) = \{q_1, q_2, q_4, q_6, q_7\}$

ε-closure $(q_7) = \{q_7\}$

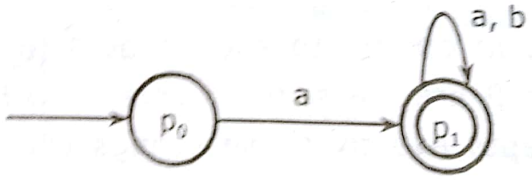
oufastupdates.com

**Construct a DFA over S = {a, b} that accept the set of strings starts with a and ends with b.**

**SOLUTION**

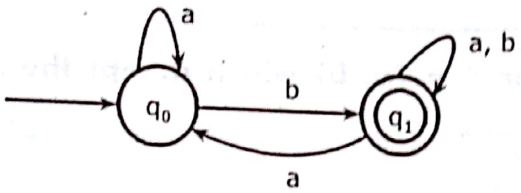
It is clear that the language is intersection of $L_1$ = {w | w starts with a} and $L_2$ = {w | w ends with b}.

The DFA $M_1$ which accept $L_1$ is,




oufastupdates.com

And DFA $M_2$ which accept $L_2$ is,




The DFA M which accept $L_1 \cap L_2$ is constructed as follows,

Since,

$$Q = Q_1 \times Q_2$$
$$= \{p_0, p_1\} \times \{q_0, q_1\}$$
$$= \{(p_0, q_0)\ (p_0, q_1), (p_1, q_0), (p_1, q_1)\}$$

The $(p_0, q_0)$ is initial state whereas $(p_1, q_1)$ is final state.

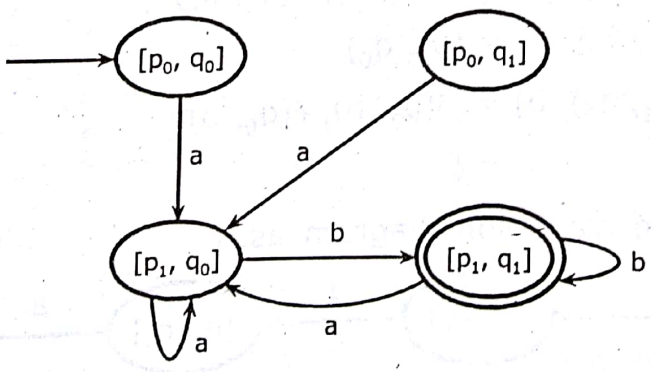
The $\delta'$ is defined as,

$$\delta'((p_0, q_0), a) = (\delta(p_0, a), \delta(q_0\ a))$$
$$= (p_1, q_0)$$
$$\delta'((p_0, q_0), b) = (\delta(p_0, b), \delta(q_0\ b))$$
$$= (\phi, q_1) = \phi$$

Similarly,

$$\delta((p_0, q_1), a) = (p_1, q_0)$$
$$\delta((p_0, q_1), b) = \phi$$
$$\delta((p_1, q_0), a) = (p_1, q_0)$$
$$\delta((p_1, q_0), b) = (p_1, q_1)$$
$$\delta((p_1, q_1), a) = (p_1, q_0)$$
$$\delta((p_1, q_1), b) = (p_1, q_1)$$

Transition graph of M is as follows,




Clearly, $(p_0, q_1)$ is unreachable. So, we can eliminate it.

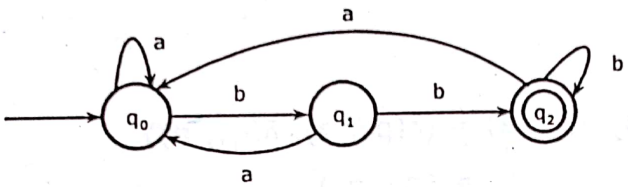
Instead of finding moves for all possible elements in Cartesian product. We can apply the method by starting with start state and attempt to generate the reachable states which is explained in the next problem.

oufastupdates.com

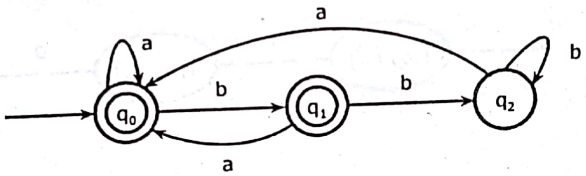
**Construct a DFA which accept the set of strings starts with aa and does not ends with bb.**

**SOLUTION**

The DFA for the language which ends with bb is as follows,

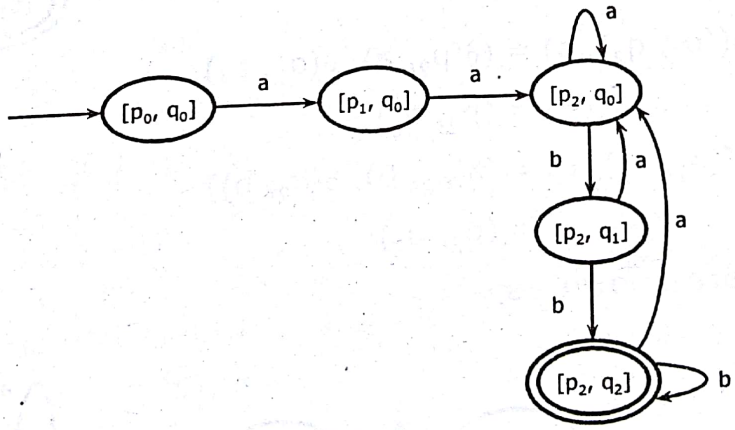



In order to find the DFA which accept the set of strings that does not ends with bb, we simply find the DFA equivalent to complement of the set of strings ends with bb. For which we make all final state to non final and non final to final. The corresponding DFA is,




To DFA corresponding to starts with aa is shown in previous example.

To find the DFA equivalent to intersection of these two languages we proceed in similar way as in previous example. The only difference is that the final states are $(p_2, q_0)$ and $(p_2, q_1)$ as final state is cartesian product of final states. The DFA is as follows,




oufastupdates.com

# DECISION PROPERTIES OF REGULAR LANGUAGES

It is important to have algorithms to answer various questions concerning regular languages. First, we should consider what it means to ask questions about a language.

The typical questions about language like,

(1) Whether a given language is empty, finite or infinite.

(2) Is a particular string w in the described language.

(3) Do two descriptions of a language actually describe the same language? This question is often called "equivalence" or not of language.

Than we say such algorithms are decision algorithms for regular languages.

The typical language is infinite, so you cannot represent the strings of the language to someone and ask a question that requires them to inspect suspect the infinite set of strings. Rather, a langauge presented by giving one of the finite representations for it that already studied i.e., developed DFA, and NFA an ε NFA or a regular expression.

## Converting Among Representations

We have studied that we can convert any of the representations for regular languages to any of the other 3 representations, sometimes we are interested not only in the possibility of making a conversion, but in the amount of time it takes. Importantly to differentiate among algorithms that take exponential time (function of size of their input) and therefore can be performed only for relatively small instances, from those that take time that is linear, quadratic or some small degree polynomial of their input size. Here we are considering the time complexity of each of the conversions in following sections.

## Converting NFA's to DFA's

We can start with either an NFA or and ε-NFA and convert it to a DFA, the time can be exponential in the number of states requires $O(n^3)$ time. We need to search each state of n including its all arcs labelled ε. If n states present, there can be not more than $n^2$ arcs. Data structures will make sure that we can explore each state in $O(n^2)$ time. The transitive closure algorithm such as warshall's algorithm can be used to compute entire ε-closure one time.

We have computed ε-closure, we can also compute equivalent DFA using subset construction. The dominant cost for the number of states of DFA, which can be $2^n$. For each state, we can compute the transition in $O(n^3)$. Time for NFA of each input symbols conclusion of the running time of NFA to DFA conversion, including the NFA has ε-transitions, is $O(n^3 2^n)$. In practice it is common that number of states generated is much less than $2^n$, often only n states. Therefore, bounding on the running times as $O(n^3 s)$, where s is the number of states of the actual DFA.

oufastupdates.com

## DFA to NFA Conversion

This algorithmic conversion is simple, and takes O(n) time on an n states DFA. Here we need to modify the transition table for DFA by applying set brackets around states and if output for ε-NFA, add one more column for ε. Since we consider the number of input symbols as a constant, copying and processing the table takes O(n) time.

## Automaton to Regular Expression Conversion

Here, we observe that at each of n rounds (where n is number of states of DFA) have quardruple the size of the regular expression built, since each built from 4 expressions of previous round simply, we write as $n^3$ expressions can take time $O(n^3 4^n)$.

The same construction works in the same running time if input from either NFA or ε-NFA. It is important to use those constructions for NFA's, however first convert an NFA to a DFA and convert the DFA to regular expression, it takes time $O(n^3 4^{n^3} 2^n)$ which results an exponential double.

## Regular Expression to Automaton Conversion

Conversion from regular expression to an ε-NFA requires linear time. We need to parse the expression efficiently, using the technique requires only O(n) time on a regular expression of length $n^3$. The output is expression tree with one node for each symbol of the regular expression.

After construction of expression tree for regular expression, we can construct the tree, building the ε-NFA for each node. It is not possible to add more than two states and 4 arcs for any node of the expression tree.

Thus, the number of states and arcs of the resulting ε-NFA are both O(n).

Conclusion of the construction of an ε-NFA from a regular expression takes time that is linear in the size of the expression we can eliminate ε-transitions from an n state ε-NFA to make an simple NFA, in $O(n^3)$ time, without increasing the number of states.

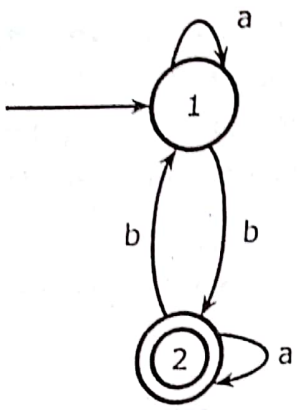
## Testing Empitness of Regular Languages

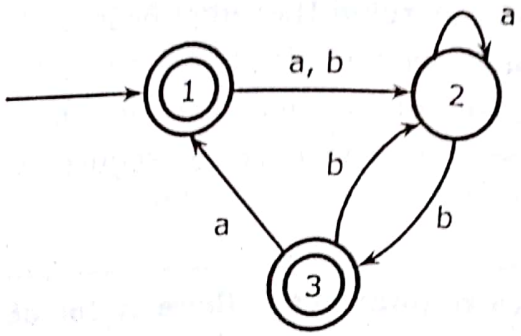
The start state is surely reachable from the start state, if there is an arc from q to p with any label (an input symbol, or ∈ if the automation is an ∈-NFA) then p is reachable. In that manner we can compute the set of reachable states. If any accepting state is among them, we answer "no" (the language of the automation is not empty) and otherwise we answer "yes". Note that the reachability calculation takes no more time that $O(n^2)$ if the automation has n states, and in fact it is no worse than proportional to the number of arcs in the automation is transition diagram, which could be less than $n^2$ and cannot be more than $O(n^2)$.

If we are given a regular expression representing the language L, rather than an automation, we could convert the expression to an ∈-NFA and proceed as above. Since the automation that results from a regular expression of length n has at most O(n) states and transitions, the algorithm takes O(n) time.

oufastupdates.com

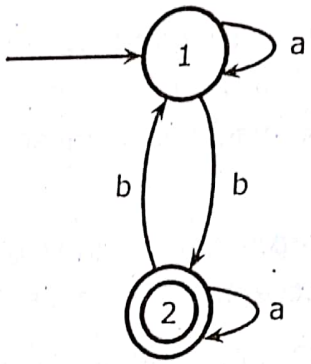
**Convert the following finite automata to regular expressions.**




**SOLUTION**

Let us first consider the following FA shown below,




A regular expression for the above finite automaton is as follows,

$$1 = 1a + 2b + \epsilon \quad \text{... (1)}$$

$$2 = 2a + 1b \quad \text{... (2)}$$

Substituting Eq. (2) in Eq. (1), we get,

$$1 = 1a + (2a + 1b)\, b + \epsilon$$

$$1 = 1a + 2ab + 1bb + \epsilon$$

$$1 = 1(a + bb) + 2ab + \epsilon \quad [\because \epsilon + R = \epsilon.R = R]$$

$$1 = \epsilon.1\,(a + bb) + 2ab$$

$$1 = 1\,(a + bb) + 2ab$$

Using Arden's theorem, we have,

$$R = 1,\ P = (a + bb),\ Q = 2ab$$

$$1 = 2ab(a + bb)^* \quad \text{... (3)}$$

oufastupdates.com

Substituting Eq. (3) in Eq. (2), we get,

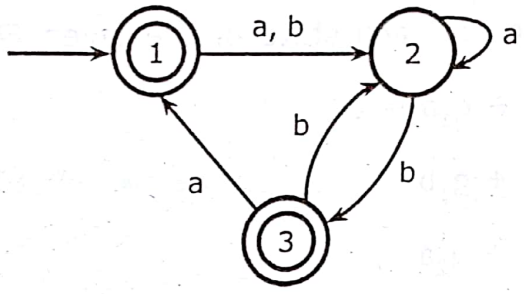
$$2 = 2a + 2ab(a + bb)^*b$$

$$2 = 2[a + ab(a + bb)^* b]$$

$$2 = (a + ab(a + bb)^*b)^*$$

Since, '2' is the final state, the regular expression for above finite automaton is $(a + ab\ (a + bb)^*b)^*$.

Now consider the FA as shown below,




A regular expression for the above finite automaton is obtained as follows,

$$1 = 3a + \epsilon \quad \text{... (4)}$$

$$2 = 1a + 1b + 2a + 3b \quad \text{... (5)}$$

$$3 = 2b \quad \text{... (6)}$$

Substituting Eq. (6) in Eq. (5) and solving Eq. (5), we get,

$$2 = 1(a + b) + 2a + 2bb$$

$$2 = 1(a + b) + 2(a + bb)$$

Using Arden's theorem, we have,

$$R = 2,\ P = (a + bb),\ Q = 1(a + b)$$

$$2 = 1(a + b)\ (a + bb)^* \quad \text{... (7)}$$

Substituting Eq. (7), in Eq. (6), we get,

$$3 = 1(a + b)\ (a + bb)^*b \quad \text{... (8)}$$

Substituting Eq. (8) in Eq. (4), we get,

$$1 = 1\ (a + b)\ (a + bb)^*ba + \epsilon$$

Using Arden's theorem, we have,

$$R = 1,\ P = (a + b)\ (a + bb)^*\ ba,\ Q = \epsilon$$

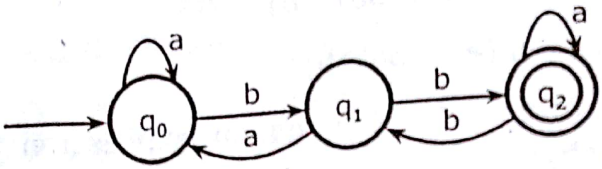
$$1 = \epsilon.((a + b)\ (a + bb)^*ba)^* \quad [\because \epsilon.R = R]$$

$$1 = ((a + b)\ (a + bb)^*\ ba)^*$$

Since, '1' is one of the final states, the regular expression for the above finite automation is $((a + b)\ (a + bb)^*\ ba)^*$.

oufastupdates.com

**Find the regular expression accepted by following deterministic finite automation.**




**SOLUTION**

The regular expression for each state of the given FA are,

$$q_0 = q_0a + q_1a + \epsilon$$

$$q_1 = q_0b + q_1b$$

$$q_2 = q_1b + q_2a$$

Since $q_2$ is a final state, solving it ultimately gives the required regular expression So, we solve for $q_2$.

Consider,

$$q_0 = q_0a + q_1a + \epsilon$$

$$q_0 = (\epsilon + q_1a)\ a^* \qquad (\because R = Q + RP \Rightarrow QP^*)$$

Substitute $q_0$ in $q_1$, we get,

$$q_1 = (\epsilon + q_1a)\ a^*b + q_1b$$

$$q_1 = \epsilon.a^*b + q_1aa^*\ b + q_2b$$

$$q_1 = a^*b + q_1aa^*\ b + q_2b \qquad (\because \epsilon.R = R)$$

$$q_1 = (a^*b + q_2b)\ (aa^*\ b)^* \qquad (\because R = Q + RP \Rightarrow QP^*)$$

Substitute $q_1$ in $q_2$, we get,

$$q_2 = (a^*b + q_2b)\ (aa^*\ b)^*\ b + q_2a$$

$$q_2 = a^*b(aa^*\ b)^*\ b + q_2b\ (aa^*\ b)^*\ b + q_2a$$

$$q_2 = a^*b(aa^*\ b)^*\ b + q_2\ (a + b\ (aa^*\ b)^*\ b)$$

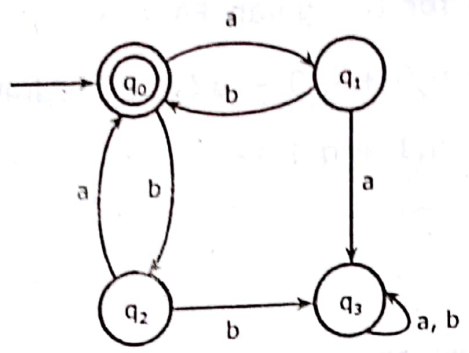
$$q_2 = a^*b\ (aa^*b)\ ^*b(a + b\ (aa^*\ b)^*\ b)^* \qquad (\because R = Q + RP \Rightarrow QP^*)$$

Therefore, the regular expression for the given DFA is,

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

oufastupdates.com

**Construct the regular expression accepted by following finite automaton.**




**SOLUTION**

The regular expressions for the given FA are,

$$q_0 = q_1b + q_2a + \epsilon \quad \text{... (1)}$$

$$q_1 = q_0a \quad \text{... (2)}$$

$$q_2 = q_0b \quad \text{... (3)}$$

$$q_3 = q_1a + q_2b + q_3a + q_3b \quad \text{... (4)}$$

Substituting Eq. (2) and (3) in Eq. (1), we get,

$$q_0 = q_0ab + q_0ba + \epsilon$$

$$q_0 = q_0\,(ab + ba) + \epsilon$$

$$q_0 = \epsilon.(ab + ba)^* \qquad (\because R = Q + RP \Rightarrow QP^*)$$

$$q_0 = (ab + ba)^* \qquad (\because \epsilon R = R)$$

Since $q_0$ is final state in the given FA. So, regular expression is given by $q_0$,

$$\therefore \quad r = (ab + ba)^*$$

## SOLVED PROBLEM 4

**Consider the FA and construct a regular expressions that is accepted by it,**

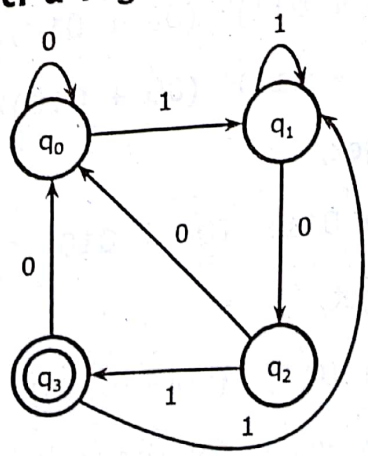



**Fig. 4.4.3**

oufastupdates.com

**SOLUTION**

Assume $q_0$ as initial state.

The regular expressions for the given FA are,

$$q_0 = q_0 0 + q_2 0 + q_3 0 + \epsilon \ (\because q_0 \text{ is initial state}) \quad \ldots (1)$$

$$q_1 = q_0 1 + q_1 1 + q_3 1 \quad \ldots (2)$$

$$q_2 = q_1 0 \quad \ldots (3)$$

$$q_3 = q_2 1 \quad \ldots (4)$$

Since $q_3$ is final state, we are interested in solving $q_3$ only.

Consider,

$$q_3 = q_2 1$$

$$q_3 = q_1 01 \quad \ldots (5)$$

Substitute Eq. (5) in Eq. (2), we get,

$$q_1 = q_0 1 + q_1 1 + q_1 011 + \epsilon$$

$$q_1 = q_0 1 + q_1 (1 + 011)$$

$$q_1 = q_0 1 (1 + 011)^* \quad (\because R = Q + RP \Rightarrow QP^*) \quad \ldots (6)$$

We have,

$$q_0 = q_0 0 + q_2 0 + q_3 0 + \epsilon$$

$$q_0 = q_0 0 + q_1 00 + q_1 010 + \epsilon$$

$$q_0 = q_0 0 + q_1 (00 + 010) + \epsilon$$

From Eq. (6), we get,

$$q_0 = q_0 0 + q_0 1 (1 + 011)^* (00 + 010) + \epsilon$$

$$q_0 = q_0 (0 + 1(1 + 011)^* (00 + 010)) + \epsilon$$

$$q_0 = \epsilon.(0 + 1 (1 + 011)^* (00 + 010))^* \quad (\because \epsilon.R = R)$$

Substitute $q_0$ in Eq. (6), we get,

$$q_1 = (0 + 1 (1 + 011)^* (00 + 010))^* (1 + (1 + 011)^*)01$$

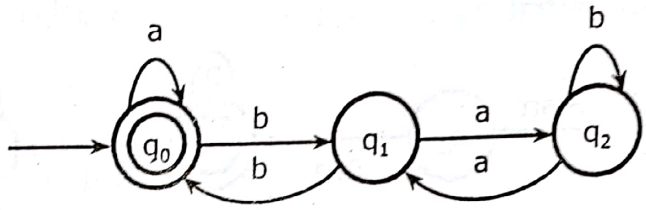
Substitute $q_1$ in Eq. (5), we get,

$$q_3 = (0 + 1 (1 + 011)^* (00 + 010))^* (1 + (1 + 011)^*) 01$$

Therefore, the regular expression for the given FA is $(0 + 1 (1 + 011)^* (00 + 010))^* (1 + (1 + 011)^*) 01$.

oufastupdates.com

**Find the regular expression accepted by following automaton.**




**SOLUTION**

The regular expressions for the given NFA are,

$$q_0 = q_0a + q_1b + \epsilon \quad \text{... (1)}$$

$$q_1 = q_0b + q_2a \quad \text{... (2)}$$

$$q_2 = q_1a + q_2b \quad \text{... (3)}$$

Since $q_0$ is a final state, solving it ultimately gives the required regular expression. So, we solve for $q_0$,

$$q_0 = q_0a + q_1b + \epsilon$$

$$q_0 = q_0a + (q_0b + q_2a)b + \epsilon$$

$$q_0 = q_0a + q_0bb + q_2ab + \epsilon$$

$$q_0 = q_0\ (a + bb) + q_2ab + \epsilon \quad \text{... (4)}$$

Substitute Eq. (2) in Eq. (3), we get,

$$q_2 = (q_0b + q_2a)\ a + q_2b$$

$$q_2 = q_0ba + q_2aa + q_2b$$

$$q_2 = q_0ba + q_2\ (aa + b)$$

$$q_2 = q_0ba\ (aa + b)^* \quad (\because R = Q = RP \Rightarrow QP^*) \quad \text{... (5)}$$

Substitute Eq. (5) in Eq. (4), we get,

$$q_0 = q_0\ (a + bb) + q_0\ ba\ (aa + b)^*\ ab + \epsilon$$

$$q_0 = q_0\ [(a + bb) + ba(aa + b)^*\ ab] + \epsilon$$

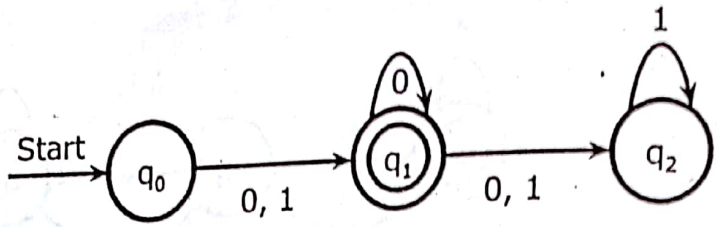
$$q_0 = \epsilon.[a + bb) + ba\ (aa + b)^*\ ab]^*$$

$$q_0 = [(a + bb) + ba\ (aa + b)^*\ ab]^* \quad (\because \epsilon.R = R)$$

Therefore, the regular epxression for the given FA is $[(a + bb) + ba(aa + b)^*\ ab]^*$.

oufastupdates.com

**Find regular expression for the following NFA.**




**SOLUTION**

The regular expressions for each state of the given NFA are,

$$q_0 = \epsilon \qquad (\because q_0 \text{ is an initial state})$$

$$q_1 = q_0 0 + q_0 1 + q_1 0$$

$$q_2 = q_1 0 + q_1 1 + q_2 1$$

Since $q_1$ is a final state, solving it ultimately gives the required regular expressio

$$q_1 = q_0 0 + q_0 1 + q_1 0$$

$$q_1 = q_0(0 + 1) + q_1 0$$

$$q_1 = q_0(0 + 1)0^* \qquad (\because R = Q + RP \Rightarrow QP^*)$$

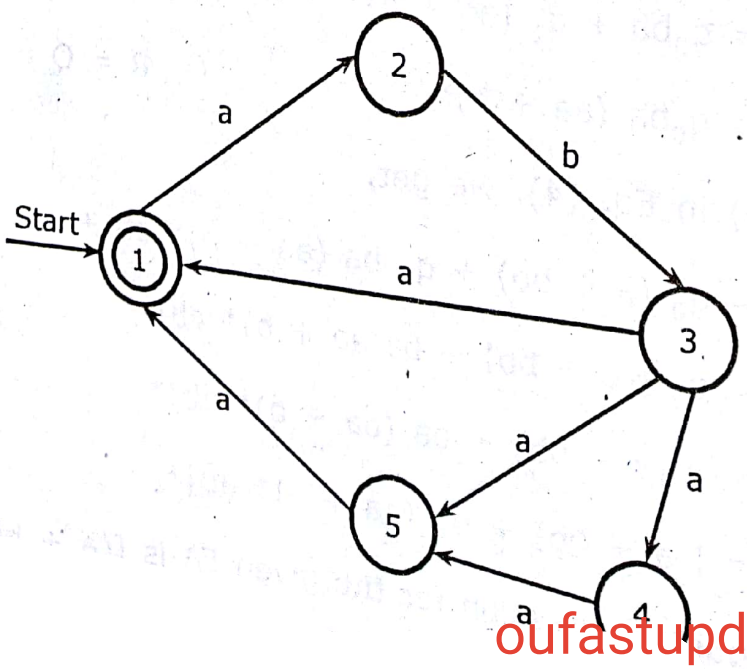
$$q_1 = \epsilon\,(0 + 1)^* \qquad (\because \epsilon.R = R)$$

$$q_1 = (0 + 1)0^*$$

Therefore, the regular expression for the given NFA is $r = (0 + 1)0^*$.

## SOLVED PROBLEM 7

**Find regular expression for the following NFA.**




oufastupdates.com

**SOLUTION**

The regular expression for each state of the given NFA are,

$$1 = 3a + 5a + \epsilon \qquad (\because 1 \text{ is an initial state}) \qquad \ldots (1)$$

$$2 = 1a \qquad \ldots (2)$$

$$3 = 2b \qquad \ldots (3)$$

$$4 = 3a \qquad \ldots (4)$$

$$5 = 3a + 4a \qquad \ldots (5)$$

Since 1 is the final state, solving it ultimately gives the required regular expression. So we solve for 1.

Consider,

$$3 = 2b$$

$$3 = 1ab \qquad \ldots (6)$$

$$4 = 3a$$

$$4 = 1aba \qquad \ldots (7)$$

$$5 = 3a + 4a$$

$$5 = 1aba + 1abaa \qquad \ldots (8)$$

Substituting Eq. (6) and (7) in Eq. (1), we get,

$$1 = 1aba + (1aba + 1abaa)a + \epsilon$$

$$1 = 1aba + 1abaa + 1abaaa + \epsilon$$

$$1 = 1(aba + abaa + 1abaaa + \epsilon$$

$$1 = \epsilon.(aba + abaa + abaaa)^* \qquad (\because R = RP + Q \in QP^*)$$

$$1 = (aba + abaa + abaaa)^* \qquad (\because \epsilon.R = R)$$

## SOLVED PROBLEM 8

**Construct DFA for the following regular expression $(ab \cup aba)^*a$.**

**SOLUTION**

Given regular expression (r.e) is,

$$r.e = (ab \cup aba)^*a$$

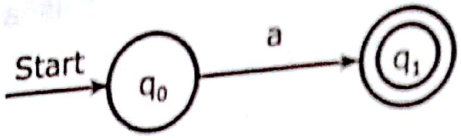
The construct a DFA, first construct an NFA and then convert NFA to DFA.

oufastupdates.com

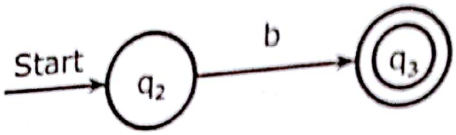
Constructiong NFA,

r.e = (ab ∪ aba)*a

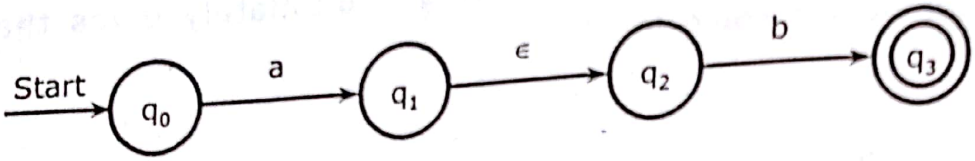
r.e = a



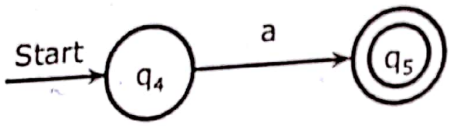

r.e = b



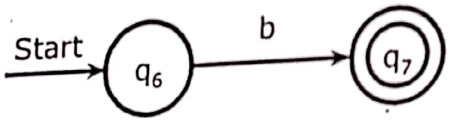

r.e = ab



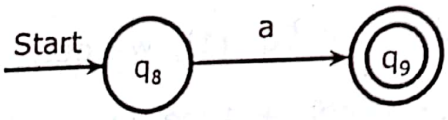

r.e = a



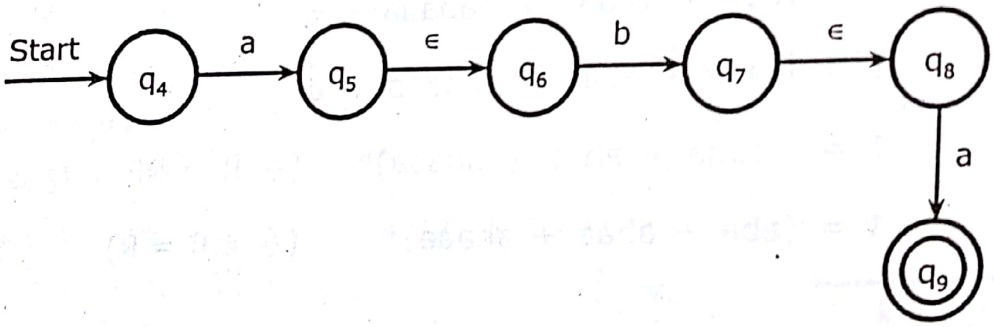

r.e = b



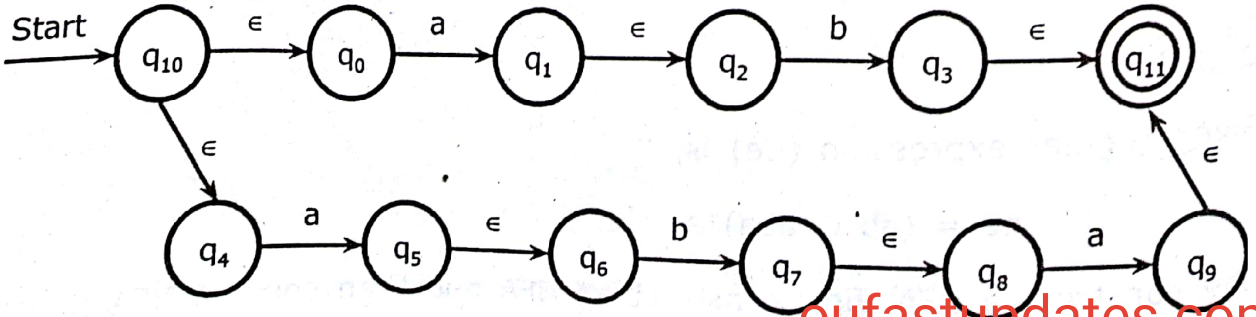

r.e = a




r.e = aba

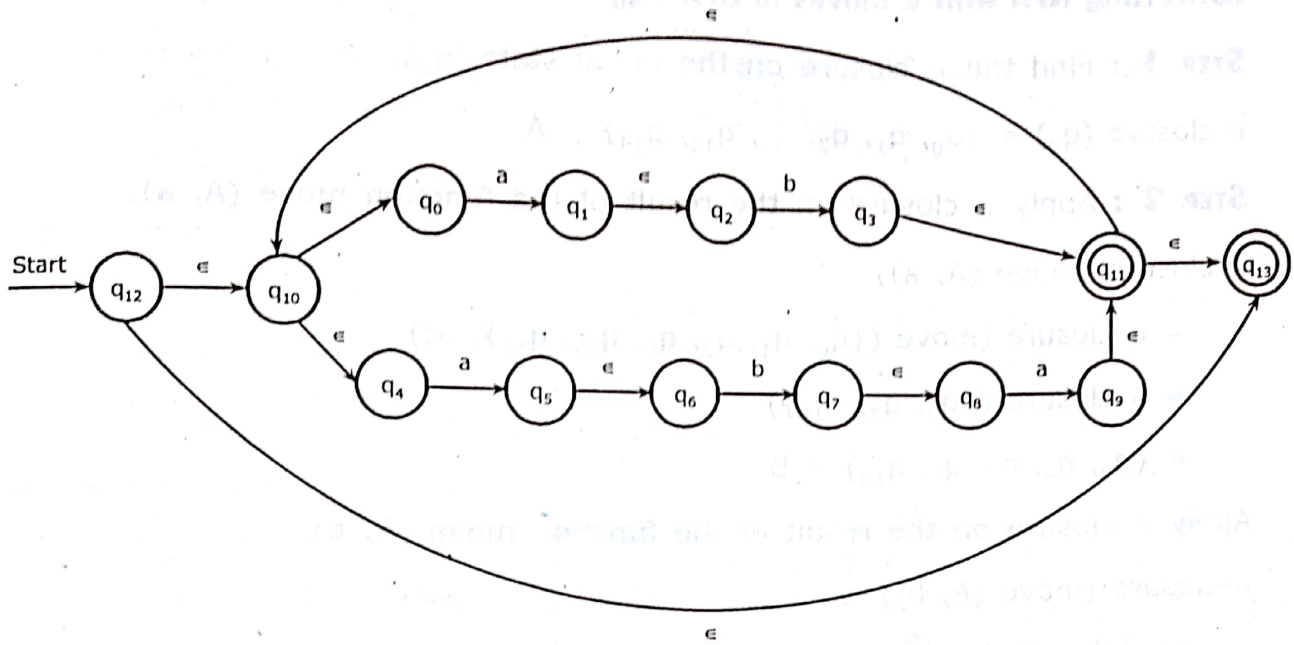



r.e (ab ∪ aba)

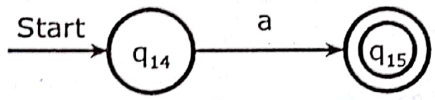



oufastupdates.com

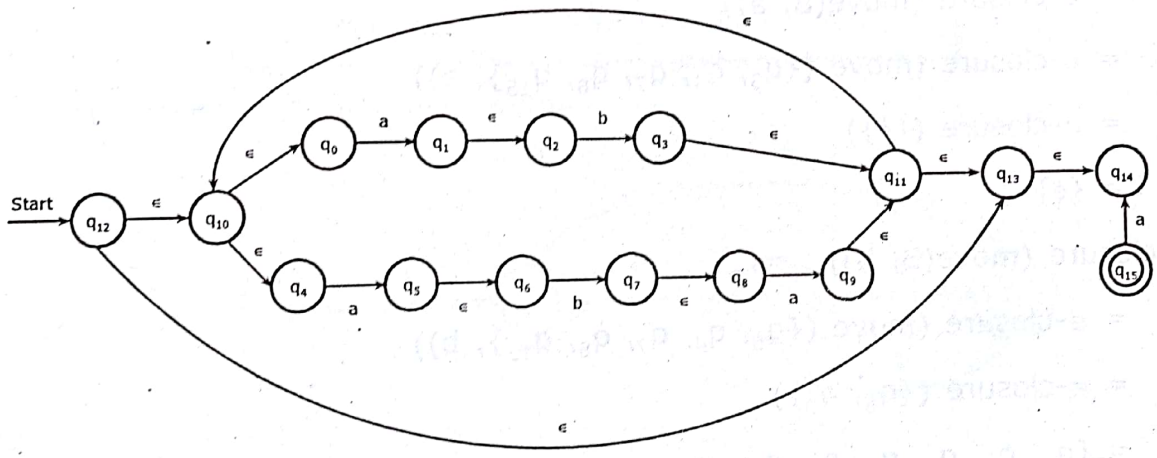
r.e (ab ∪ aba)*




r.e = a




r.e + (ab ∪ aba)*a




For simplicity rewriting states from $q_0$ to $q_{15}$ in sequence as shown in Fig. 4.4.4.

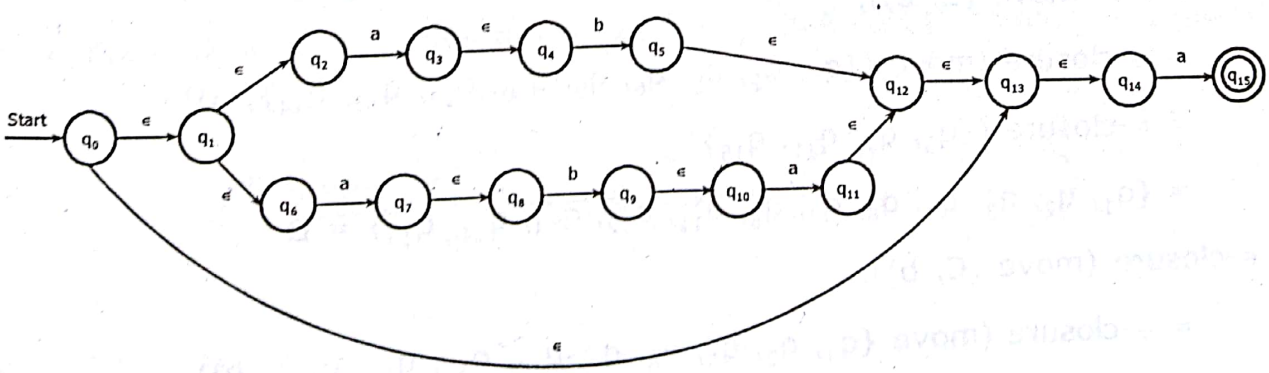



**Fig. 4.4.4** NFA for r.e (ab ∪ aba)* a

oufastupdates.com

# EQUIVALENCE AND MINIMIZATION OF AUTOMATA

In this section we are discussing how to test the two descriptors of regular language are equivalent or not and both defined as same language i.e., consider any DFA to find an equivalent DFA that has the minimum number of states. In fact, essentially a unique DFA given any two minimum states DFA's that are equivalent, we can determine how to rename the states so that the two DFA's becomes same. This can be achieved by determining *distinguishable* and *indistinguishable* states.

## Testing Equivalence of States

In single DFA, on our aim is to understand when two distinct states p and q can be replaced by a single state and acts like both p and q. We can say that p and q states are equivalent if and only if for all input string w, if $\delta(p, w)$ and $\delta(q, w)$ are accepting state or both $\delta(p, w)$ and $\delta(q, w)$ are non final states for all $w \in \Sigma^*$ i.e.,

If $\delta(p, w) \in F$ and $\delta(q, w) \in F$ So we can say p and q are *indistinguishable.*

If $\delta(p, w) \notin F$ and $\delta(q, w) \notin F$. Then also the p and q states are *indistinguishable.*

If there is a atleast one string w such that $\delta(p, w) \in F$ is the accepting states and $\delta(q, w) \notin F$ is non accepting states or vice versa then the states p and q are not equivalent and are called *distinguishable states.*

The distinguishable and indistinguishable states can be finding out from the table filling algorithm (mark algorithm) recursively as follows,

**Basis**

**STEP 1 :** For each pair (p, q) if $p \in F$ and $q \notin F$ or vice versa then the pair (p, q) is *distinguishable* and mark the pair (p, q) using x.

**Induction**

**STEP 2 :** Let p and q be pair of states such that for some input symbol a, $\delta(p, a) = r$ and $\delta(q, a) = s$ are pair (r, s) is already marked as *distinguishable* the pair (p, q) is also *distinguishable* and mark it using x.

**STEP 3 :** Repeat step 2 for each pair (p, q) till the pair (p, q) is not marked or atleast one pair (p, q) is not marked in the current step.

**NOTE :** *Before applying this procedure, ensure that the all the states of a DFA are reachable from the start and delete all those states that are not reachable from the start state.*

oufastupdates.com

**Consider the DFA and whose transition table close represented as,**

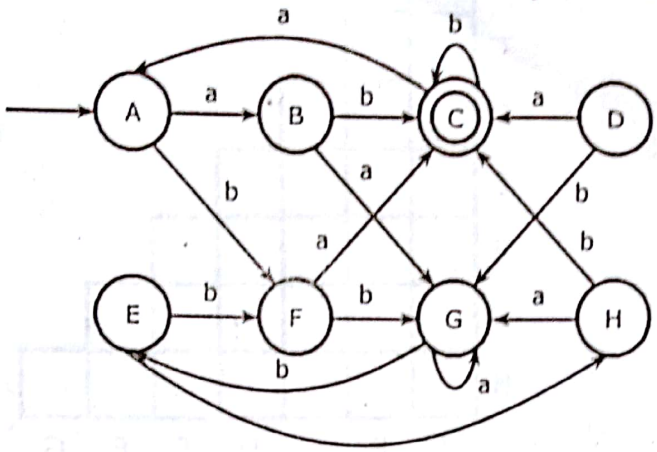



**Fig. 4.5.1** DFA

**Table 4.5.1** Transition Table

| | Σ | |
|---|---|---|
| δ | a | b |
| A | B | F |
| B | G | C |
| *C | A | C |
| D | C | G |
| E | H | F |
| F | C | G |
| G | G | E |
| H | G | C |

**SOLUTION**

First find out the table contains inequalities (distinguishable or indistinguishable) This can be done using different states given in DFA. The DFA states are,

A, B, C, D, E, F, G, H.

The pairs can be represented as below.

(A, B) (A, C) (A, D) (A, E) (A, F) (A, G) (A, H)

(B, C) (B, D) (B, E) (B, F) (B, G) (B, H)

(C, D) (C, E) (C, F) (C, G) (C, H)

(D, E) (D, F) (D, G) (D, H)

(E, F) (E, G) (E, H)

(F, G) (F, H)

(G, H)

oufastupdates.com

The pairs can be represented in tabular form and no pairs are marked,

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| B | | | | | | | |
| C | | | | | | | |
| D | | | | | | | |
| E | | | | | | | |
| F | | | | | | | |
| G | | | | | | | |
| H | | | | | | | |
| | A | B | C | D | E | F | G |

**STEP 1 :** For each pair (p, q), if p ∈ F and q ∉ F or vice versa then the pair (p, q) is *distinguishable* and make the pair (p, q) using X. This clears that one of the pair must be in accepting state and other must be in non accepting state. So the difference pair can fulfill the above criteria of the all the accepting state C and marked as X,

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| B | | | | | | | |
| C | X | X | | | | | |
| D | | | X | | | | |
| E | | | X | | | | |
| F | | | X | | | | |
| G | | | X | | | | |
| H | | | X | | | | |
| | A | B | C | D | E | F | G |

**STEP 2 :** Let p and q be pair of states such that for some input symbol a, find δ(p, q) = r and δ(p, q) = s. If the pair (r, s) is already marked as distinguishable (marked as X) Then pair (p, q) is also distinguishable and mark it as X.

Considering the pairs (p, q) which is not marked in step 1 and finding the corresponding pairs (r, s) on input symbol a and b as shown below.

**NOTE :** *In step 1 marked pairs of (r, s) its corresponding pair (p, q) must be marked. For example, for pair (A, B) on input symbol b is (E, C) and it is marked as x in step 1. So mark the pair (A, B). Similar way identify pair (r, s) which are marked in step1 and its corresponding pairs (p, q) also need to be marked as X. Updated marked table with X is shown in Table 4.5.2.*

oufastupdates.com

**Table 4.5.2** Updated Marked Table With X

| δ | a | mark | b | mark |
|---|---|---|---|---|
| (p, q) | (r, s) | x | (r, s) | x |
| (A, B) | (B, G) | - | (F, C) | x |
| (A, D) | (B, C) | x | (F, G) | - |
| (A, E) | (B, H) | - | (F, F) | - |
| (A, F) | (B, C) | x | (F, G) | - |
| (A, G) | (B, G) | - | (F, E) | - |
| (A, H) | (B, G) | - | (F, C) | x |
| (B, D) | (G, C) | x | (C, G) | x |
| (B, E) | (G, H) | - | (C, F) | x |
| (B, F) | (G, C) | x | (C, G) | x |
| (B, G) | (G, G) | - | (C, E) | x |
| (B, H) | (G, G) | - | (C, C) | - |
| (D, E) | (C, H) | x | (G, F) | - |
| (D, F) | (C, C) | - | (G, G) | - |
| (D, G) | (C, G) | x | (G, E) | - |
| (D, H) | (C, G) | x | (G, C) | x |
| (E, F) | (H, C) | x | (F, G) | - |
| (E, G) | (H, G) | - | (F, E) | - |
| (E, H) | (H, G) | - | (F, C) | x |
| (F, G) | (C, G) | x | (G, E) | - |
| (F, H) | (C, G) | x | (G, C) | x |
| (G, H) | (G, G) | - | (E, C) | x |

oufastupdates.com

(i) $\delta(A, a) = B$

$\delta(B, a) = G$

(ii) $\delta(A, b) = F$

$\delta(B, b) = C$

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| B | X | | | | | | |
| C | X | X | | | | | |
| D | X | X | X | | | | |
| E | - | X | X | X | | | |
| F | X | X | X | - | X | | |
| G | - | X | X | X | - | X | |
| H | X | - | X | X | X | X | X |

**Fig. 4.52** Marked Transitions for the Transitions Table

**STEP 3 :** Repeat the process by considering (p, q) which is not marked in the above table and look at the corresponding pairs (r, s) in a and b are marked as follows,

| δ | a | mark | b | mark |
|---|---|---|---|---|
| (p, q) | (r, s) | x | (r, s) | x |
| (A, E) | (B, H) | - | (F, F) | - |
| (A, G) | (B, G) | x | (F, E) | x |
| (B, H) | (G, G) | - | (C, C) | - |
| (D, F) | (C, C) | - | (G, G) | |
| (E, G) | (H, G) | x | (F, E) | x |

The marked pairs of (A, G) and (E, G) as shown below,

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| B | X | | | | | | |
| C | X | X | | | | | |
| D | X | X | X | | | | |
| E | - | X | X | X | | | |
| F | X | X | X | - | X | | |
| G | X | X | X | X | X | X | |
| H | X | - | X | X | X | X | X |

From the above, the pair (A, E), (B, H) and (D, F) are not marked and these states are to be *indistinguishable* and other states namely C and G states are *distinguishable.*

oufastupdates.com

## Testing Equivalence of Regular Languages

In this section, the method to use table filling algorithm to test whether two given DFAs represent the same language is discussed.

To test whether the given two FAs $M_1$ and $M_2$ are equivalent it is imagined as if the two FAs are combined. The imaginary FA will contain m + n states in it, where m is the number of states in $M_1$ and n is the number of states in $M_2$. The imaginary FA contains two initial states, initial state of $M_1$ and $M_2$ which is into allowed in real case.

Table filling algorithm is used for the imaginary DFA. A table of size (m + n – 1) x (m + n – 1) is used and the two initial states are tested for equivalence. If they are equivalent then it may be concluded that both the DFAs represent same language.

For example, the two DFAs in Fig. 4.5.3 and Fig. 4.5.4 are tested for equivalence. The two automata are combined imaginarily and there will be five states A, B, C, D and E in the combined imaginary DFA.

There are two initial states A and C in the automaton. As the first step of the process a 4 x 4 table is formed and the cells corresponding to a pair of states in which one is a final state and the other is not marked.

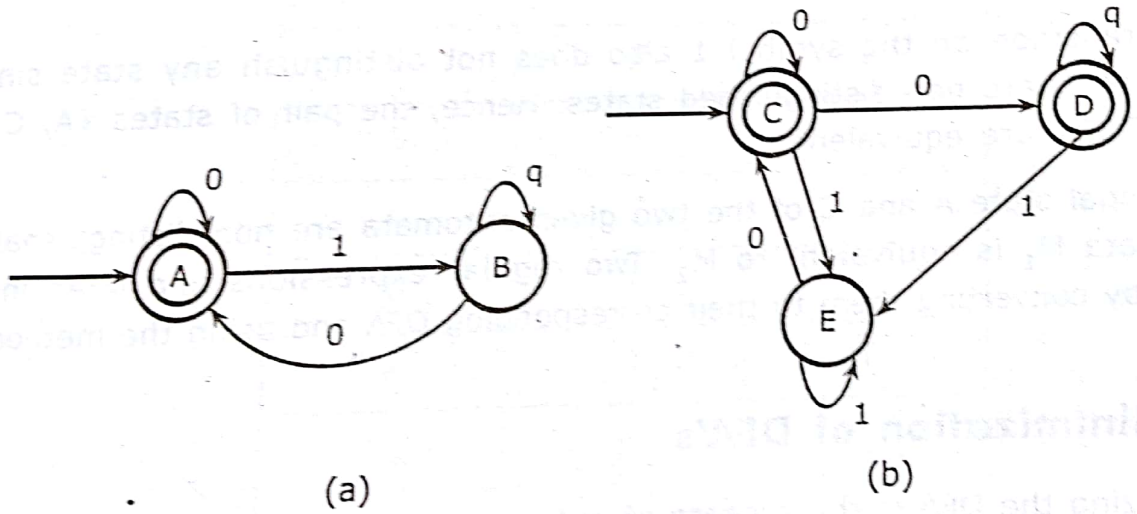



**Fig. 4.5.3** Two Equivalent DFA's

As per the base condition final states A, C and D are distinguishable from non final states B and E and the cells are marked in the table as in Fig. 4.5.4.

| | A | B | C | D |
|---|---|---|---|---|
| B | X | | | |
| C | | X | | |
| D | | X | | |
| E | X | | X | X |

oufastupdates.com

Then transition from all the states on the symbol 0 is determined as below,

$\delta(A, 0) = A$

$\delta(B, 0) = A$

$\delta(C, 0) = D$

$\delta(D, 0) = D$

$\delta(E, 0) = C$

No pair of states may be distinguished by looking at the above transition since the entire set of states moves to non distinguished states on reading the symbol 0.

The transition of the states on the symbol 1 is analyzed as given below,

$\delta(A, 1) = B$

$\delta(B, 1) = B$

$\delta(C, 1) = E$

$\delta(D, 1) = E$

$\delta(E, 1) = E.$

The transition on the symbol 1 also does not distinguish any state since all the resulting states are non distinguished states. Hence, the pair of states {A, C}, {A, D}, {B, E}, {C, D} are equivalent.

The initial state A and C of the two given automata are non distinguishable, hence, the automata $M_1$ is equivalent to $M_2$. Two regular expressions and NFAs may also be compared by converting them to their corresponding DFA and using the method described above.

## Minimization of DFA's

Minimizing the DFA is the process of reducing the number of states in a DFA without altering the language accepted by the DFA.

The basic idea used is to determine the set of equivalent states in the given DFA and a combine them. The procedure is just an extension of determining equivalent states. After determining equivalent states, the states in the DFA are partitioned to blocks such that.

(1) All states in a block are equivalent.

(2) No two states chosen from two different blocks are equivalent.

The states in the DFA is Fig. 4.5.4 can be put in blocks as follows,

{A, E}, {B, H}, {D, F}, {C}, {G}

oufastupdates.com

The minimized DFA looks as in Fig. 4.5.4.

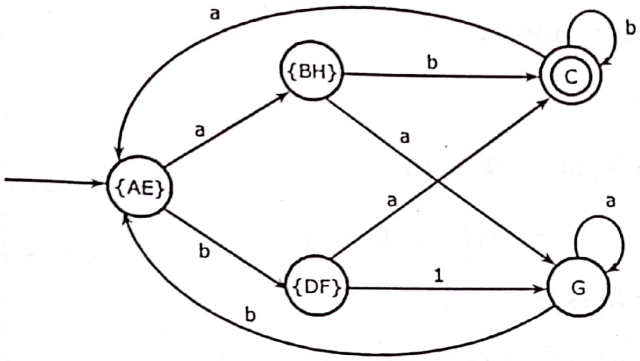



**Fig. 4.5.5** Minimized DFA Corresponding to the One in Fig 4.5.2

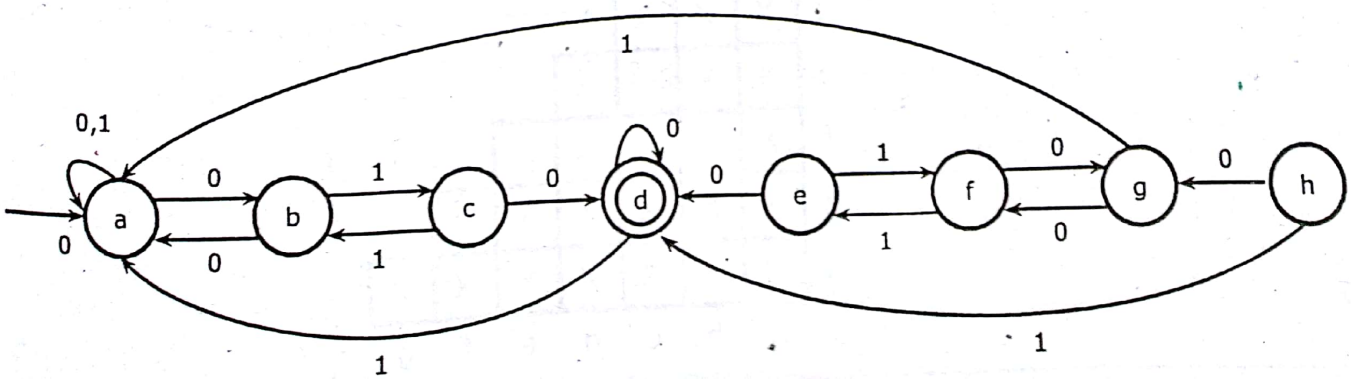
## EXAMPLE PROBLEM 4.6

**Construct a minimum state automaton whose transition table is given below diagram.**

**Transition Table**

| States | Inputs | |
|---|---|---|
| | 0 | 1 |
| → a | ab | a |
| b | a | c |
| c | d | b |
| *d | d | a |
| e | d | f |
| f | g | e |
| g | f | a |
| h | g | d |

**Transition Diagram**




oufastupdates.com

To find equivalent states,

a : (Initial state),

a ⇒ 0(b) ⇒ 1(1) ⇒ 0 ⇒ (d)

a ⇒ 1(a) ⇒ 0(b) ⇒ 1(1) ⇒ 0 ⇒ d

b : b ⇒ 0(1) ⇒ 0 ⇒ d

b ⇒ 0(a) ⇒ 0(b) ⇒ 1(1) ⇒ 0 ⇒ d

c : c ⇒ 0 ⇒ d

a ⇒ 1(b) ⇒ 0(a) ⇒ 0(b) ⇒ 1(1) ⇒ 0 ⇒ d

d : e ⇒ 0 ⇒ d

e ⇒ 1(f) ⇒ 0(g) ⇒ 0(f) ⇒ 1(e) ⇒ 0 ⇒ d

f : f ⇒ 0(g) ⇒ 0(f) ⇒ 1(e) ⇒ 0 ⇒ d

f ⇒ 1(e) ⇒ 0 ⇒ d

g : g ⇒ 0(f) ⇒ 1(e) ⇒ 0 ⇒ d

g ⇒ 1(a) ⇒ 1(b) ⇒ 1(c) ⇒ 0 ⇒ d

h : h ⇒ 0(g) ⇒ 0(f) ⇒ 1(e) ⇒ 0 ⇒ d

h ⇒ 1 ⇒ d

**Equivalent States :** [a, g], [b, f], [c, e].

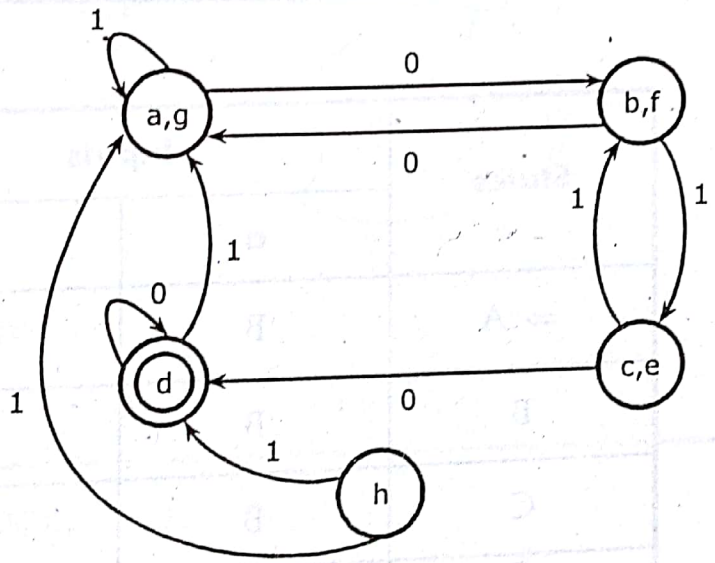
**Table Filling Algorithm**

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| a | X | | | | | | |
| c | X | X | | | | | |
| d | X | X | X | | | | |
| e | X | X | - | X | | | |
| f | X | - | X | X | X | | |
| g | - | X | X | X | X | X | |
| h | X | - | X | X | X | X | X |

oufastupdates.com

| States | Inputs | |
|---|---|---|
| | 0 | 1 |
| ⇒ [a, g] | [b, f] | [a, g] |
| [b, f] | [a, g] | [c, e] |
| [c, e] | d | [b, f] |
| *d | d | [a, g] |
| H | [a, g] | [d] |

**Minimal State DFA**




## EXAMPLE PROBLEM 4.7

**Construct the minimal state automata.**

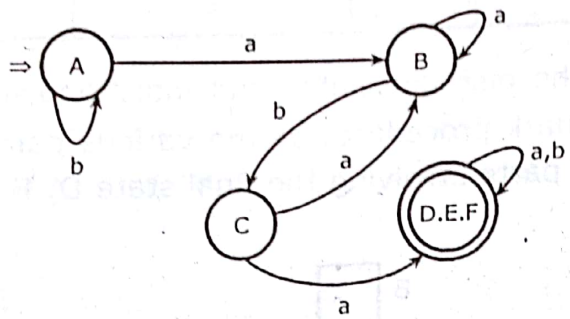
| States | Inputs | |
|---|---|---|
| | 0 | 1 |
| ⇒ A | B | A |
| B | B | C |
| C | B | D |
| *D | E | D |
| *E | E | F |
| *F | E | F |

oufastupdates.com

**Transition Table**

| States | Inputs | |
|---|---|---|
| | a | b |
| ⇒ A | B | A |
| B | B | C |
| C | B | [D, E, F] |
| *[D, E, F] | [D, E, F] | [D, E, F] |

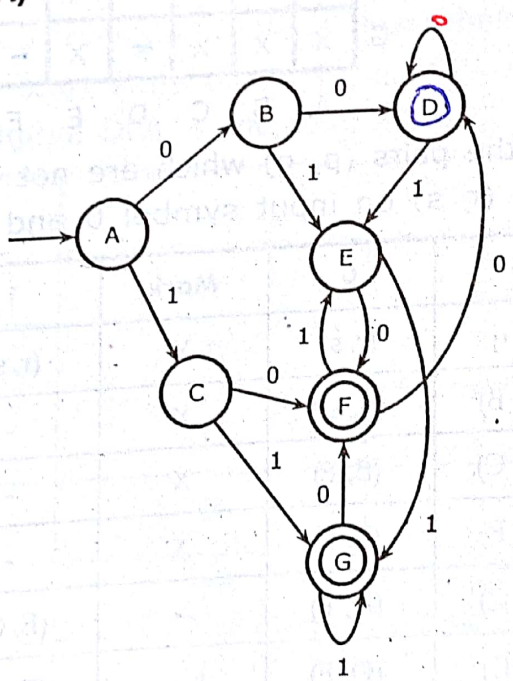
**Minimal State DFA**




## 4.5.4 Solved Problems

### SOLVED PROBLEM 1

**Consider the following DFA,**




**(i) Write the transition table and distinguishable states for this automata.**

**(ii) Construct the minimum state equivalent DFA.**

oufastupdates.com

**SOLUTION**

(i) The transition table for the above DFA

| δ | 0 | 1 |
|---|---|---|
| → A | B | C |
| B | D | E |
| C | F | G |
| *D | D | E |
| E | F | G |
| *F | D | E |
| *G | F | G |

**STEP 1 :** Obtain the distinguishable and indistinguishable states using the table filling algorithm (mark procedure) so the various pairs which satisfies table filling pair criteria for all parts involving the final state D, F and G are marked as shown below,

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| B | | | | | | |
| C | | | | | | |
| D | X | X | X | | | |
| E | | | | X | | |
| F | X | X | X | | X | |
| G | X | X | X | | X | |

Now, consider the pairs (p, q) which are not marked in step 1 and obtain corresponding pairs (r, s) on input symbol 0 and 1 as shown below,

| δ | 0 | Mark | 1 | Mark |
|---|---|---|---|---|
| p, q | (r, s) | X | (r, s) | X |
| (A, B) | (B, D) | X | - | - |
| (A, C) | (B, F) | X | - | - |
| (A, E) | (B, F) | X | - | - |
| (B, C) | (D, F) | - | (E, G) | X |
| (B, E) | (D, F) | - | (E, G) | X |
| (C, E) | (F, F) | - | (G, G) | - |

oufastupdates.com

**NOTE :** *If the pairs (r, s) on 0 and 1 are marked in above table, the corresponding pair (p, q) should be marked for examples for the pair (A, B) on input O is (B, D) and it is marked X in the step 1. So mark the pair (A, B). Similar way identify pairs (r, s) which are marked in step 1 and take the corresponding pair (p, q) and update the marking table as shown below.*

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| B | X | | | | | |
| C | X | X | | | | |
| D | X | X | X | | | |
| E | X | X | | X | | |
| F | X | X | X | - | X | |
| G | X | X | X | - | X | - |

From the above table filling the pair (C, E) is not marked and this state is to be *indistinguishable* and other states namely A, B, D, F and G states are *distinguishable*.

**(ii) Minimization of DFA**

**STEP 2 :** The pair (C, E) is not marked is called indistinguishable and the states A, B, D, F and G are distinguishable. So the different groups can be represented as the states of minimized DFA are,

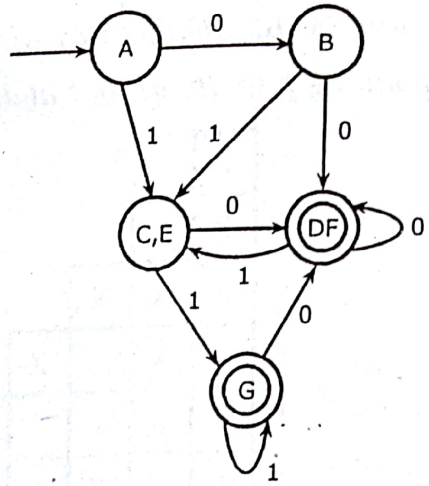
A, B, (C, E), D, F, G

**STEP 3 :** The minimized transition table is shown below. Note that the start state of original DFA is the start state of minimized DFA and the group which consist of a final state of original DFA is the final state of minimized DFA.

| States | Σ | |
|---|---|---|
| | 0 | 1 |
| → A | B | (C, E) |
| B | D | (C, E) |
| (C, E) | F | G |
| * D | D | (C, E) |
| * F | D | (C, E) |
| * G | F | G |

oufastupdates.com

Here D and F on input symbols transition results same therefore, we consider it as pair (D, F) and the final transition diagram for the minimized DFA is shown below,




## SOLVED PROBLEM 2

**Consider the DFA given by transition table.**

| δ | 0 | 1 |
|---|---|---|
| → $q_1$ | $q_2$ | $q_3$ |
| $q_2$ | $q_3$ | $q_5$ |
| * $q_3$ | $q_4$ | $q_3$ |
| $q_4$ | $q_3$ | $q_5$ |
| * $q_5$ | $q_2$ | $q_5$ |

**(i) Draw the table of distinguishable for their automaton.**

**(ii) Construct the minimum state equivalent DFA.**

**SOLUTION**

**(i) STEP 1 :** Obtain the distinguishable and indistinguishable states using table filling algorithm (mark procedure). So, the various pairs which satisfies table filling pair criteria for all pairs providing the final state $q_3$ and $q_5$ are marked as shown below.

| | $q_1$ | $q_2$ | $q_3$ | $q_4$ |
|---|---|---|---|---|
| $q_2$ | | | | |
| $q_3$ | X | X | | |
| $q_4$ | | | X | |
| $q_5$ | X | X | - | X |

oufastupdates.com

Now consider the pair (p, q) which are not marked in step 1 and obtain corresponding pair (r, s) on input symbol 0 and 1 as shown below,

| δ | 0 | Mark | 1 | Mark |
|---|---|---|---|---|
| p, q | (r, s) | | (r, s) | |
| $(q_1, q_2)$ | $(q_2, q_3)$ | X | - | - |
| $(q_1, q_4)$ | $(q_2, q_3)$ | X | - | - |
| $(q_2, q_4)$ | $(q_3, q_5)$ | - | $(q_5, q_5)$ | - |

**NOTE :** *If the pairs (r, s) on 0 and 1 are marked in the above table, the corresponding pair (p, q) should be marked. For example, for the pair $(q_1, q_2)$ on input O is $(q^2, q^3)$ and it is marked X in the step 1. So mark in pair $(q_2, q_3)$. Similar way identity pairs (r, s) which are marked in step 1 and take the corresponding pair (p, q) and update the marking table as shown below.*

| | | | | |
|---|---|---|---|---|
| $q_2$ | X | | | |
| $q_3$ | X | X | | |
| $q_4$ | X | - | X | |
| $q_5$ | X | X | - | X |
| | $q_1$ | $q_2$ | $q_3$ | $q_4$ |

Form the above table filling the pair $(q_2, q_4)$ is not marked and this state is indistinguishable and other states namely $q_1$, $q_3$ and $q_5$ states are distinguishable.

**(ii) Minimization of DFA**

**STEP 2 :** The pair $(q_2, q_4)$ is not marked is called indistinguishable and the states $q_1$, $q_3$ and $q_5$ are distinguishable. So, the different groups can be represented as the states of minimized DFA are,

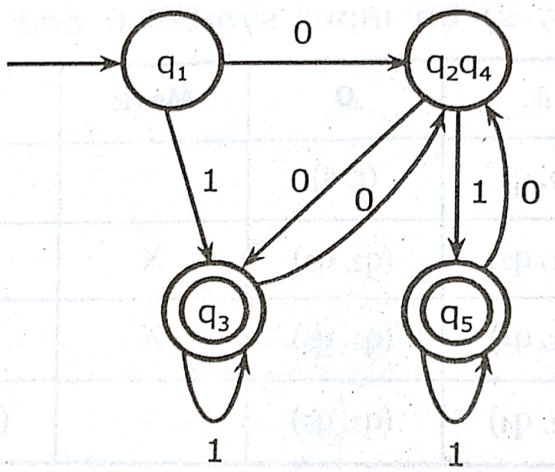
$$q_1, (q_2, q_4), q_3, q_5$$

**STEP 3 :** The minimized transition table is shown below. Note that the start state of original DFA is the start state of minimized DFA and the group which consist of a final state of original DFA is the final state of minimized DFA.

| States | Σ | |
|---|---|---|
| | 0 | 1 |
| → $q_1$ | $(q_2, q_4)$ | $q_3$ |
| $(q_2, q_4)$ | $q_3$ | $q_5$ |
| * $q_3$ | $(q_2, q_4)$ | $q_3$ |
| * $q_5$ | $(q_2, q_4)$ | $q_5$ |

oufastupdates.com

Therefore, the transition diagram for the minimized DFA is shown below,




**Thanks for downloading**

oufastupdates.com

# *Long questions ( UNIT 2)*

-> PDA

-> PDA to Grammar

-> Parse trees ,LFD, RMD

-> Grammar is ambiguous or not !

oufastupdates.com

# CONTEXT FREE GRAMMARS

We have studied regular languages in previous chapter, now our concentration is on large class of languages called as *"context free languages"*. This language contains natural and recursive notations are called *"context free grammars"*. We introduce context free grammar notations and define languages from grammar how it emerged and study of "parse tree" structure model that grammar defining on strings of languages.

A context free grammar basically consists of a finite set of grammar rules. In order to define grammar rules, we assume that we have two kinds of symbols, the *terminals*, which were the symbol of the alphabet underlying the language under considerations, and a *non terminals* which behave like variables ranging over string of terminals. A rule is of the form $A \rightarrow \alpha$ where A is a single nonterminal and the right hand side $\alpha$ is a string of terminal and/or nonterminal symbols. As usual, first we need to define what the object is (a context free grammar) and their need and it use. Unlike automate, grammar is used to generate strings rather than recognize strings.

## An Informal Example

Let us now discuss some examples of recursive grammars,

**Example 1 :** Consider a language $L1 = \{a^k b^k \mid k \geq 0\}$ over (a, b). The grammar rules for the language $L_1$ can be defined as follows,

**RULE 1 :** $\epsilon \in L_1$.

**RULE 2 :** For every string $W \in L_1$, $awb \in L_1$.

**RULE 3 :** Nothing else belongs to $L_1$.

Since k can be 0, $\epsilon$ is a valid string in the language. The definition in the form of awb, in Rule 2, embeds equal numbers of a's and b's in every valid string in $L_1$. Rule 3 prohibits any other mechanism to generate a string for the language $L_1$. An informal production system for the language $L_1$ can be written as follows,

$$S \rightarrow \epsilon \mid aSB$$

Where, S is a variable describing the initial form of strings belonging to a language. Let derive a string $a^4b^4 \in L_1$,

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow aaaSbbb \rightarrow aaaaSbbbb \rightarrow aaaabbbb.$$

In the last case, S is substituted by its terminal definition $\epsilon$, whereas in all the remaining cases it is substituted by its recursive definition aSb.

**Example 2 :** Let us consider a language $L_2 = \{a^k b^k \mid k > 0\}$ over {a, b}. The grammar rules for the language L can be defined as follows,

oufastupdates.com

**Rule 1 :** $ab \in L_2$.

**Rule 2 :** For every string $w \in L_2$, $awb \in L_2$.

**Rule 3 :** Nothing else belongs to $L_2$.

The difference between the languages $L_1$ and $L_2$ is that in language $L_2$, $\epsilon$ is not a valid string as $k = 0$ is not allowed. Therefore, the initial case in $L_2$ is different, but the subsequent string generation process is same.

An informal production system for the language $L_2$ can be written as follows,

$$S \rightarrow ab \mid aSb$$

Let us derive a string $a^4b^4 \in L_2$,

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow aaaSbbb \rightarrow aaaabbbb.$$

In the last case, S is substituted by its terminal definition ab, whereas in all the remaining cases, it is substituted by its recursive definition aSb.

**Example 3 :** Consider a language $L_3$ over {a, b, c} such that $L_3 = \{w \mid w \text{ is a palindrome}\}$. The grammar rules for the language $L_3$ can be defined as follows,

**Rule 1 :** $\epsilon \in L_3$.

**Rule 2 :** Every alphabet a, b, c $\in L_3$.

**Rule 3 :** For every string $w \in L_3$, $awa \in L_3$, $bwb \in L_3$, and $cwc \in L_3$.

**Rule 4 :** Nothing else belongs to $L_3$.

An informal production system for the language $L_3$ can be written as follows,

$$S \rightarrow \epsilon \mid aSa \mid bSb \mid cSc \mid a \mid b \mid c.$$

Let us derive a string $aacbbcaa \in L_3$,

$$S \rightarrow aSa \rightarrow aaSaa \rightarrow aacSaa \rightarrow aacbSbcaa \rightarrow aacbbSbbcaa \rightarrow aacbbcaa.$$

In the last case S is substituted by its terminal definition $\epsilon$, whereas in all the remaining cases it is substituted by its terminal definitions, which may be one among aSa, bSb, or cSc.

**Example 4 :** Consider a language of mathematical expressions $L_4$ over {a, b, c, d, e, +, -, x, /, (,)} such that,

**Rule 1 :** Alphabets a, b, c, d, e $\in L_4$.

**Rule 2 :** For every string $w \in L_4$, $w + w \in L_4$, $w - w \in L_4$, $w \times w \in L_4$, $w/w \in L_4$, and $(w) \in L_4$.

**Rule 3 :** Nothing else belongs to $L_4$.

An informal production system for the language $L_4$ can be written as follows,

$$S \rightarrow a \mid b \mid c \mid d \mid e \mid S + S \mid S - S \mid S/S \mid S \times S \mid (S).$$

oufastupdates.com

The language $L_4$ describes the mechanism to define a mathematical expression us derive a mathematical expression $(a + b) \times c/(d \times e) \in L_4$,

$$S \rightarrow S/S \rightarrow S \times S/(S \times S) \rightarrow (S) \times S/(S \times S) \rightarrow (S + S) \times S/(S \times S) \rightarrow (a + b) \times c/(d \times e)$$

**Example 5 :** Consider a language of simple assignment statement in a programming language $L_5$ over {a, b, c, d, e, +, –, x, /, (,), =} such that,

**RULE 1 :** Alphabets a, b, c, d, e $\in$ I.

**RULE 2 :** Every string $w_1 = w_2 \in L_5$, if $w_1 \in I$ and $w_2 \in L_4$.

**RULE 3 :** Nothing else belongs to $L_5$.

An informal production system for the language $L_5$ can be written as follows,

$$S \rightarrow I = S$$

$$S \rightarrow a \mid b \mid c \mid d \mid e \mid S + S \mid S - S \mid S/S \mid (S) \mid S \times S$$

$$I \rightarrow a \mid b \mid c \mid d \mid e$$

The language $L_5$ describes the mechanism to define an assignment statement in a programming language. Let us derive an assignment statement,

$$a = (a + b) \times c/(d \times e) \in L_5.$$

$$S \rightarrow I = S \rightarrow I = S/S \rightarrow a = S \times S/(S \times S) \rightarrow a = (S) \times S/(S \times S) \rightarrow a = (S + S) \times S/(S \times S) \rightarrow a = (a + b) \times c/(d \times e).$$

Thus, we see that a frequently used programming construct (assignment statement) can be implemented using a recursively defined grammar. All the languages defined earlier are nearly CFLs without one restriction that a CFL has to follow (substitution of $\epsilon$ string is not allowed). However, we will see later in this chapter that all these languages can also be generated without the substitution of null string.

A production of the form $S \rightarrow \epsilon$ is allowed in a CFG, to allow $\epsilon$ as a valid string in the corresponding CFL, L. However, the use of the production $S \rightarrow \epsilon$ for substitution purposes is not allowed. In language $L_5$, no production of type $S \rightarrow \epsilon$ has been used, hence it is valid CFG for a native arithmetic assignment statement in a programming language. This use of CFG for the generation of assignment statement in a programming language shows how a CFG can be used for the design of a parser module in the compiler. Now, let us start with the formal description of CFGs and languages.

oufastupdates.com

## Definition of Context Free Grammars

This is the way of describing language by recursive rules called production. It consists of set of variables non-terminal, set of terminal symbols and a starting variable as well as the production,

$$G = (V, T, P, S)$$

Where,

V = Variables.

T = Terminals.

P = Productions.

S = Starting variable.

(1) **Terminals :** Terminals are basic symbols from which strings are formed. The word token is a synonym for terminal.

**Example :** If, then, else, +, −, and soon.

(2) **Non-Terminals (Variables) :** These are syntactic variables that denote set of strings. The non-terminals define sets of strings that help define the language generated by the grammar.

(3) **Start Symbol :** One non-terminal is distinguished as the start symbol and the set of strings it denotes is the language defined by the grammar.

(4) **Production :** The production of a grammar specify the manner in which the terminals and non-terminals can be combined to form strings.

Each production consists of,

(i) A variable can be defined (partially) by the production. This variable oftenly called production head.

(ii) A production symbol →

(iii) A string of zero or more symbols/terminals and variables. These strings oftenly called production body represents one way to form strings in the language of variables of the head.

### EXAMPLE PROBLEM 5.1

**Find a context free grammar to generate the string of palindromes over T = {a, b}.**

**SOLUTION**

A palindrome is a string that input reads the same forward and backward i.e., Madam, Malyalam etc. Another way consider string w is a palindrome if and only if $w = W^R$ can be expressed in simple way with an alphabet {a, b} the, language contains string like abba, bbabb and ε.

oufastupdates.com

A context free grammar is a formal notation for expressing as recursive definitions of languages. A grammar consists of one or more variables that represent classes of strings i.e., languages.

The palindrome can be recessively defined as follows,

(i) ε is the palindrome.

(ii) a, b are palindrome.

(iii) If S is a palindrome, that aSa, bSb, are palindrome.

The productions corresponding to these definitions are,

(i) $S \rightarrow \varepsilon$

(ii) $S \rightarrow a$

(iii) $S \rightarrow b$

(iv) $S \rightarrow aSa$

(v) $S \rightarrow bSb$

## EXAMPLE PROBLEM 5.2

**Construct a CFG for arithmetic expression that contain identifies defined by regular expression = (a + b) (a + b + 0 + 1)*. The arithmetic expression = (A + B) (A + B + 0 + 1)*. The arithmetic expression to have operators + and * and should have facility for parentheses.**

**SOLUTION**

The regular expression (a + b) (a + b + 0 + 1) is of the form $S_1S_2$.

Where,

$S_1 \rightarrow (A + B)$.

$S_2$ (A + B + x + y).

The regular expression $S_1(S_2)^*$ means that $S_1$ is followed by any number of $S_2$. So the $S_1$ can be generated using the productions,

$S_1 \rightarrow (A + B)$

$A \rightarrow a$

$B \rightarrow b$

oufastupdates.com

Any number of $S_2$ can be generated using the productions,

$$S_2 \rightarrow (A + B + x + y)$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$x \rightarrow 0$$

$$y \rightarrow 1$$

Now, the language generated from the regular expression (a + b) (a + b + 0 + 1)* can be obtained by concatenating $S_1$ and $S_2$ using the production,

$$S \rightarrow S_1 S_2$$

So, the final grammar is,

$$G = (V, T, P, S)$$

Where,

$V = \{S_1, S_2, A, B, x, y\}$

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

$P = \{S \rightarrow S_1 S_2\}$

$S_1 \rightarrow (A + B)$

$S_2 \rightarrow (A + B + x + y) S_2 \mid S$

$A \rightarrow a$

$B \rightarrow b$

$x \rightarrow 0$

$y \rightarrow 1$

S is the S that symbol.

## Derivations Using a Grammar

The process of obtaining string of terminals or non terminal from the start symbol by applying some number of productions is called the *derivation*.

Let,

$A \rightarrow \alpha B \gamma$

$B \rightarrow \beta$ are productions in grammar G.

Where,

$V = \{A, B\}$.

$T = \{\alpha, \beta, \gamma\}$ i.e., $\alpha\beta\gamma \in (VUT)^*$.

oufastupdates.com

Form the non terminal A, the string $\alpha, \beta, \gamma$ can be obtained as follows,

$$A \Rightarrow \alpha\ B\ \gamma \Rightarrow \alpha\ \beta\ \gamma \text{ (by applying } B = \beta)$$

If a string is obtained by applying one or more productions, we use a symbol $\overset{+}{\Rightarrow}$.

If the string is obtained by applying only one productions then, we use the symbol $\Rightarrow$.

If the string is obtained by applying zero or more productions, we use a symbol $\overset{*}{\Rightarrow}$.

**Example**

(1) $A \Rightarrow \alpha B \beta$. This indicates that the string $\alpha B \beta$ can be obtained from A by applying one production.

(2) $A \overset{+}{\Rightarrow} \alpha B \beta$. This indicates that the string B is obtained from A by applying one or more productions.

(3) $A \overset{*}{\Rightarrow} \alpha B \beta$. This indicates that the string $\alpha B \beta$ can is obtained from A by applying zero or more productions.

## EXAMPLE PROBLEM 5.3

**Consider the grammar shown below,**

| | | |
|---|---|---|
| **E → E + E** | – | **$P_1$** |
| **E → E * E** | – | **$P_2$** |
| **E → (E)** | – | **$P_3$** |
| **E → id** | – | **$P_4$** |

**SOLUTION**

The string id + (id * id) can be derived from the above grammar as than below,

| | |
|---|---|
| $E \overset{+}{\Rightarrow} E + E$ | By applying $P_1$ |
| $\overset{+}{\Rightarrow}$ id + E | By applying $P_4$ |
| $\overset{+}{\Rightarrow}$ id + (E) | By applying $P_3$ |
| $\overset{+}{\Rightarrow}$ id + (E * E) | By applying $P_2$ |
| $\overset{+}{\Rightarrow}$ id + (id * E) | By applying $P_4$ |
| $\overset{+}{\Rightarrow}$ id + id * id | By applying $P_4$ |

oufastupdates.com

## Leftmost and Rightmost Derivations

### LEFT MOST DERIVATION

If at each step in derivation, a production is applied to the left most variable (or) left most, non-terminal then the derivation method is called *left most derivation*. It is indicated by $\underset{lm}{\Rightarrow}$ and $\overset{*}{\underset{lm}{\Rightarrow}}$ for one or many steeps.

**Example :** Consider the grammar,

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow (E)$$

$$E \rightarrow id$$

The string id + id * id can be obtained by applying leftmost derivation as shown below,

$$E \underset{lm}{\Rightarrow} E + E$$

$$\underset{lm}{\Rightarrow} id + E$$

$$\underset{lm}{\Rightarrow} id + E * E$$

$$\underset{lm}{\Rightarrow} id + id * E$$

$$\underset{lm}{\Rightarrow} id + id * id$$

### RIGHT MOST DERIVATION

A derivation in which the right most variable is replaced at each step then, the derivation method is called *right most derivation*. It is indicated by $\underset{rm}{\Rightarrow}$ and $\overset{*}{\underset{rm}{\Rightarrow}}$ for one or many steps.

**Example :** Consider the grammar,

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow (E)$$

$$E \rightarrow id$$

oufastupdates.com

The string id + id * id can be obtained by applying right most derivation as shown below,

$$E \Rightarrow E + E$$
$$\underset{rm}{\Rightarrow} E + E * E$$
$$\underset{rm}{\Rightarrow} E + E * id$$
$$\underset{rm}{\Rightarrow} E + id * id$$
$$\underset{rm}{\Rightarrow} id + id * id$$

**Example :** Consider the grammar,

$$G = \{V, T, P, S\}$$

Where,

$V = \{S, A, B\}$,

$T = \{a, b, c\}$ and P is given by,

$S \rightarrow aABb$

$A \rightarrow bAb$

$A \rightarrow C$

$B \rightarrow aB$

$B \rightarrow b$

(1) The string abcbabb could be derived,

$$S \Rightarrow aABb$$
$$\Rightarrow abAbBb$$
$$\Rightarrow abcbBb$$
$$\Rightarrow abcbaBb$$
$$\Rightarrow abcbabb$$

Following a rule that always expands the leftmost nonterminal first such a derivation is called a left most derivation.

(2) The string abcbabb could be also derived,

$$S \Rightarrow aABb$$
$$\Rightarrow aAaBb$$
$$\Rightarrow aAabb$$
$$\Rightarrow abAbabb$$
$$\Rightarrow abcbabb$$

Following a rule that always expands the rightmost nonterminal first such a derivation called a rightmost derivation.

oufastupdates.com

## The Language of Grammar

Let,

$G = (V, T, P, S)$ be a context free grammar.

The language of G, denoted by L(G), is the set of terminal string that have generated by grammar G from start symbol, that is,

$$L(G) = \left\{ w \in T \mid S \overset{+}{\underset{G}{\Rightarrow}} w \right\}$$

If a language L is the language of some context free grammar then L is said to be a context free language of CFL.

## Sentential Forms

If $A \overset{*}{\Rightarrow} \beta$, $\beta$ is said to be in sentential form if $\beta$ contains terminals or non-terminals. Hence, the symbol $\overset{*}{\Rightarrow}$ means a number of derivation steps used.

**Left-Sentential Form :** If $A \overset{*}{\underset{rm}{\Rightarrow}} \beta$ using leftmost derivation then $\beta$ is a left sentential form of the grammar.

**Right-Sentential Form :** If $A \overset{*}{\underset{rm}{\Rightarrow}} \beta$ using rightmost derivation (canonical derivation) and $\beta$ may contain non-terminals, then $\beta$ is a right-sentential form of the grammar.

### EXAMPLE PROBLEM 5.4

**Consider the grammar,**

$$E \rightarrow E + E \mid E * E \mid (E) \mid - E \mid id.$$

**SOLUTION**

The string w = id + id can be derived as follows,

$$E \Rightarrow E + E$$

$$\overset{lm}{\Rightarrow} id + E$$

$$\overset{lm}{\Rightarrow} id + id$$

Here, the strings E + E, id + E, and id + id are all sentential forms, specifically, they are left sentential forms.

The right sentential forms can be obtained as,

$$E \Rightarrow E + E$$

$$\overset{rm}{\Rightarrow} E + id$$

$$\overset{rm}{\Rightarrow} id + id$$

Here, E + E, E + id and id + id are also sentential forms, specifically they are right sentential forms.

oufastupdates.com

**Write a CFG, which generates string of balanced parenthesis.**

**SOLUTION**

This grammar will accept the balanced right and left parenthesis.

For example ( ) ( ) is acceptable, ((( ))) is also accepted.

Let us design the CFG for this,

Let CFG be,

$G = (V_n, T, P, S)$

Where,

$V_n$ = Set of non terminals = {S}.

T = Set of terminals = {(,)}

And set of production P is given by,

$S \rightarrow SS$

$S \rightarrow (S)$

$S \rightarrow \epsilon$

Now we see what this grammar generates,

$S \Rightarrow SS$

$\Rightarrow (S)\ S$

$\Rightarrow (S)\ SS$

$\Rightarrow (S)\ (S)\ S$

$\Rightarrow (S)\ (S)\ (S)$

$\Rightarrow (\ )\ (S)\ (S)$

$\Rightarrow (\ )\ (\ )\ (S)$

$\Rightarrow (\ )\ (\ )\ (\ )$

Thus the above grammar always generates balanced pairs of parenthesis.

oufastupdates.com

**Write a CFG for the regular expression,**

$$r = 0^* \, 1(0 + 1)^*$$

**SOLUTION**

Let us analyse regular expression,

$$r = 0^* \, 1 \, (0 + 1)^*$$

Clearly regular expression is the set of strings which starts with any number of 0's, followed by a one and end with any combination of 0's and 1's.

Let CFG be,

$$G = (V_n, T, P, S)$$

Where,

$$V_n = \{S, A, B\}$$

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

And productions P are defined as,

$$S \rightarrow A1B$$

$$A \rightarrow 0A/\epsilon$$

$$B \rightarrow 0B/1B/\epsilon$$

Let us see the derivation of the string 00101,

$$S \Rightarrow A1B$$

$$\Rightarrow 0A10B$$

$$\Rightarrow 00A101B$$

$$\Rightarrow 00101$$

So clearly G is CFG for regular expression r.

## SOLVED PROBLEM 3

**Write a CFG which generates strings having equal number of a's an b's.**

**SOLUTION**

Let CFG be,

$$G = (V_n, T, P, S)$$

$$V_n = \{S\}$$

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

Where, P is defined as,

$$S \rightarrow aSbS/bSaS/\epsilon$$

oufastupdates.com

Let us derive a string,

$$w = bbabaabbaa$$

$$S \Rightarrow b\underline{S}aS$$

$$\Rightarrow bb\underline{S}aSas$$

$$\Rightarrow bbaSb\underline{S}aSaS$$

$$\Rightarrow bbaSba\underline{S}bSaSaS$$

$$\Rightarrow bba\underline{S}baaSbSbSaSaS$$

$$\Rightarrow bbabaa\underline{S}bSbSaSaS$$

$$\Rightarrow bbabaaba\underline{S}bSaSaS$$

$$\Rightarrow bbabaabb\underline{S}aSaS$$

$$\Rightarrow bbabaabba\underline{S}aS$$

$$\Rightarrow bbabaabbaa\underline{S}$$

$$\Rightarrow bbabaabbaa$$

## SOLVED PROBLEM 4

**Write a CFG for the language $L(G) = \{ww^r : w \in \{0, 1\}^*\}$.**

**SOLUTION**

Let grammar be G,

$$G = (V_n, T, P, S)$$

$$V_n = \{S\}$$

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

Productions are defined as,

$$S \rightarrow 0S0 \quad \text{... (1)}$$

$$S \rightarrow 1S1 \quad \text{... (2)}$$

$$S \rightarrow \epsilon \quad \text{... (3)}$$

Let us derive a string $\underset{w}{\underline{001}}\ \underset{w^r}{\underline{100}}$ by above CFG.

$$S \Rightarrow 0S0 \quad \text{(By Eq. 1)}$$

$$\Rightarrow 00S00 \quad \text{(By Eq. 1)}$$

$$\Rightarrow 001S100 \quad \text{(By Eq. 2)}$$

$$\Rightarrow 001100 \quad \text{(By Eq. 3)}$$

oufastupdates.com

**Design a CFG for the language,**

$$L(G) = \{ab\ (bbaa)^n\ bba\ (ba)^n : n \geq 0\}.$$

**SOLUTION**

Let CFG be G,

$G = (V_n, T, P, S)$

$V_n = (S, X, Y)$

$T = \{a, b\}$

Productions are defined as,

$S \rightarrow abX$

$X \rightarrow bbYa$

$Y \rightarrow aaXb$

$Y \rightarrow \epsilon$

We can also define production as,

$S \rightarrow abX$

$X \rightarrow bbaaXba$

$X \rightarrow bba$

Let us derive a string $ab(bbaa)^2\ bba(ba)^2$,

$$\begin{aligned}
S &\Rightarrow ab\underline{X} \\
&\Rightarrow abbb\underline{Y}a \\
&\Rightarrow abbbaa\underline{X}ba \\
&\Rightarrow abbbaabb\underline{Y}aba \\
&\Rightarrow abbbaabbaa\underline{X}baba \\
&\Rightarrow ab(bbaa)^2bbYa(ba)^2 \\
&\Rightarrow ab\ (bbaa)^2bb\epsilon a(ba)^2 \\
&\Rightarrow ab(bbaa)^2bba(ba)^2
\end{aligned}$$

$$\begin{aligned}
S &\Rightarrow abX \\
&\Rightarrow abbbaaXba \\
&\Rightarrow abbbaabbaaXbaba \\
&\Rightarrow ab(bbaa)^2\ bba(ba)^2.
\end{aligned}$$

oufastupdates.com

**Design a CFG for the language,**

$$L = \{a^n b^m : n \neq m\}.$$

**SOLUTION**

If $n \neq m$ then there are only two cases are possible.

**CASE 1 :** Let us say language L on condition $n > m$ is,

$$L_1 \text{ and } L_1 = \{a^n b^m : n > m\}.$$

Let us say $G_1$ be the CFG for the language $L_1$,

$$G_1 = (V_n^1, V_t^1, P^1, S^1)$$

$$V_n^1 = \{S_1, A, S^1\}$$

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

Then productions $P^1$ are defined as,

$$S^1 \rightarrow AS_1$$

$$S_1 \rightarrow aS_1b/\epsilon$$

$$A \rightarrow aA/a$$

**CASE 2 :** Let us say L on condition $n < m$ is $L_2$.

And, $$L_2 = \{0^n 1^m : n < m\}$$

Let CFG for $G_2$ be $G_2$,

$$G_2 = (V_n^2, V_t^2, P^2, S^2)$$

$$V_n^2 = \{S^2, S_2, B\}$$

$$V_t^2 = \{a, b\}$$

$P^2$ are defined as follows,

$$S^2 \rightarrow S_2B$$

$$S_2 \rightarrow aS_2b/\epsilon$$

$$B \rightarrow bB/b$$

By the help (by the combining $G_1$ and $G_2$) of $G_1$ and $G_2$ we can write the CFG for L as,

$$S \rightarrow S^1/S^2$$

Where, S is start symbol of CFG for L.

oufastupdates.com

**Write the CFG for the language,**

$$L = \{a^n b^n c^m d^m / n >= 1, m >= 1\}.$$

**SOLUTION**

Let CFG for the language L be G,

$$G = (V_n, T, P, S)$$

$$V_n = \{S, X, Y\}$$

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

Productions P is defined as follows.

$$S \rightarrow XY \quad \ldots (1)$$

$$X \rightarrow aXb/ab \quad \ldots (2)$$

$$Y \rightarrow cYd/cd \quad \ldots (3)$$

When we use Eq. (1) (i.e., $S \rightarrow XY$) then X can generate equal number of a's and b's with no a's follows when any b is encountered. Similarly Y can generate strings of equal number of c's and d's with no c's follows when any d is encountered.

Let us derive a string aaabbbcd that n = 3, m = 1,

$$S \Rightarrow XY$$

$$\Rightarrow aXbY$$

$$\Rightarrow aaXbbY$$

$$\Rightarrow aaabbbY$$

$$\Rightarrow aaabbbcd$$

Which required string.

## SOLVED PROBLEM 8

**Design a CFG for the language $L = \{a^n b^{2n} / n >= 0\}$.**

**SOLUTION**

Let CFG be G for the language L,

$$G = (V_n, T, P, S)$$

$$V_n = \{S\}$$

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

And P is defined as,

$$S \rightarrow aSbb/\epsilon$$

oufastupdates.com

Let us derive a string aaabbbbbb,

$$S \Rightarrow aSbb$$

$$\Rightarrow aaSbbbb$$

$$\Rightarrow aaaSbbbbbb$$

$$\Rightarrow aaa\epsilon bbbbbb$$

$$\Rightarrow aaabbbbbb$$

Which is required string.

Now let us derive another string abbb,

$$S \Rightarrow aSbb$$

From this derivation only it is clear that third b is not possible against a single 'a'.

oufastupdates.com

## PARSE TREES

There is a tree representation for derivation that has proved extremely useful. It is the second way of showing derivations, independent of the order in which productions are used, is also called *"derivation tree"*. "A *parse tree* is an ordered tree in which nodes are labeled with the left sides of productions and in which the children of a node represent its corresponding right sides".

### Constructing Parse Trees

Let,

$G = (V_n, T, P, S)$ be a context free grammar.

An ordered tree for this CFG, G is a derivation tree if and only if it has the following properties,

(1) The root is labeled by the starting non terminal of the CFG that is S.

(2) Every leaf of the ordered tree has a label from $v_t \cup \{\epsilon\}$.

(3) Every interior node of ordered tree has a label from $V_n$.

(4) Let us assume that a vertex has label $X \in V_n$, and its children are labeled (from left to right) $y_1, y_2, y_3, \ldots, y_n$, then production must contain a production of the form,

$$X \rightarrow y_1, y_2, \ldots, y_n.$$

(5) A leaf labeled $\epsilon$ has no siblings, that is, vertex with a child labeled $\epsilon$ can have no other children. Clearly if the leaf is labeled $\epsilon$, then it must be the only child of its parent.

### The Yield of a Parse Tree

If we look at the leaves of any parse tree and can concate them from the left, we get a string, called the yield of the tree, which is always a string that is derived from the root will be proved shortly of special importance are those parse trees such that,

(1) The yield is a terminal string. That is all leaves are labeled either with a terminal or with $\epsilon$.

(2) The root is labeled by the start symbol.

oufastupdates.com

**Consider the CFG,**

**S → XX**

**X → XXX/bX/Xb/a**

**Find the parse tree for the string bbaaaab.**

**SOLUTION**

Given CFG is,

S → XX

S → XXX/bX/Xb/a

And string is w = bbaaaab

We begin with S and apply the production S → XX.

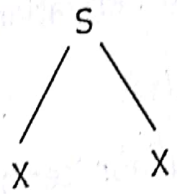



**Fig. 52.1** A Parse Tree Showing the Derivation of XX from S

To the left hand X, let us apply the production X → bX. To the right hand X, let us apply X → XXX.

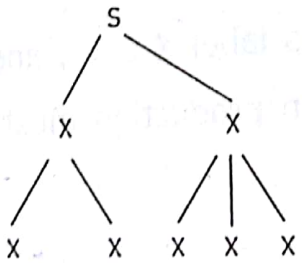



**Fig. 52.2** A Parse Tree Showing the Derivation of XXX from S

The b that we have on the bottom line is a terminal, so it does not descend further. In the terminology of trees, it is called a "terminal node". Let the four A's left to right, undergo the production A → bA, A → a, A → a, and A → Ab, respectively. We now have,

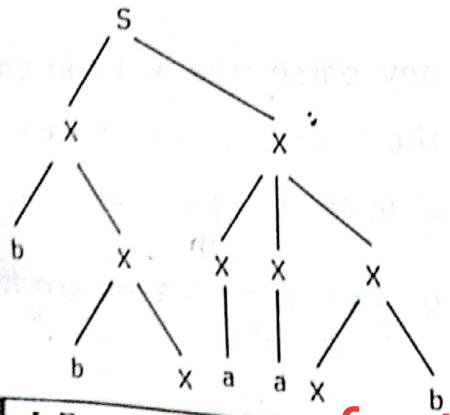



**Fig. 52.3** A Parse Tree for the Given Production

oufastupdates.com

Let us finish off the generation of a word with the production X → a and X → a.

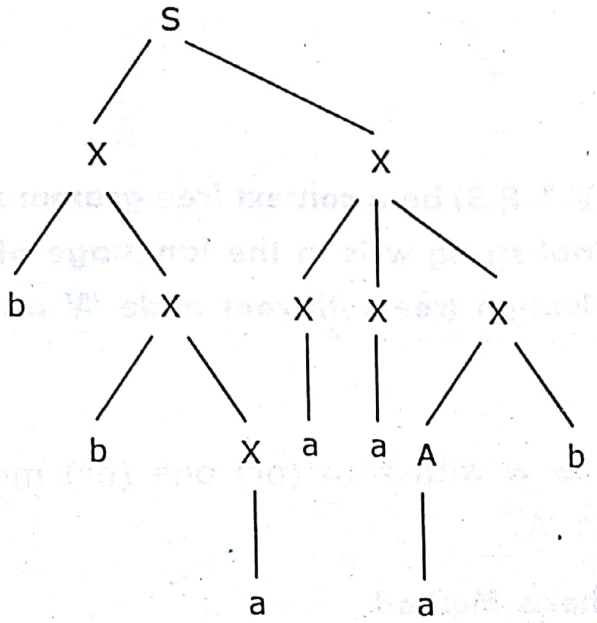



**Fig. 5.2.4** A Parse Tree of the string bbaaaab

Reading from left to right, we see the word we have produced is bbaaaab.

These tree diagrams are called "syntax trees", "parse trees", "generation trees", "production trees" or derivation trees.

## Inference, Derivations and Parse Trees

Given a grammar G = (V, T, P, S), we should show that the following are equivalent.

(1) There is a *recursive inference* procedure to derive a string w.

(2) $A \overset{*}{\Rightarrow} w$

(3) $A \overset{lm*}{\Rightarrow} w$ (There is a leftmost derivation for the strings in the language).

(4) $A \overset{rm*}{\Rightarrow} w$ (There is a rightmost derivation for the strings in the language).

(5) There is a parse tree for all the strings in L with the start symbol, A as root.

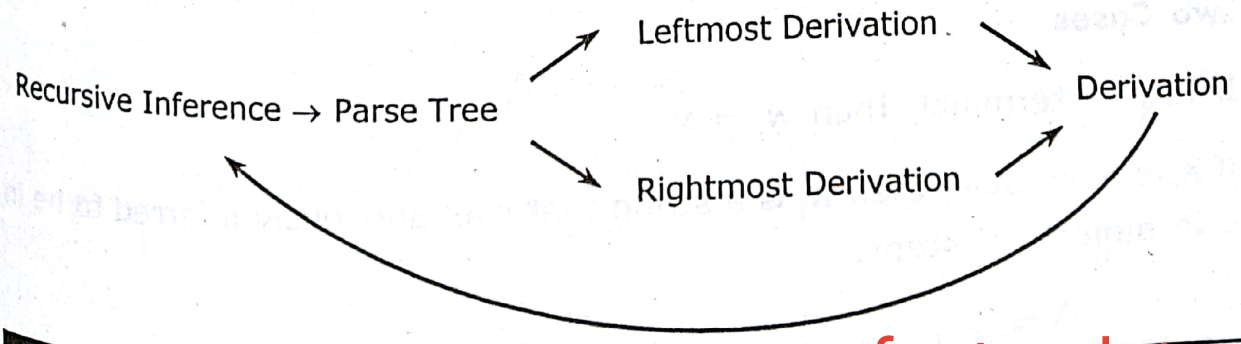



**Fig. 5.2.5** Proving the Equivalence of Certain Statements about Grammars

oufastupdates.com

## From Inferences to Trees

### THEOREM

**STATEMENT**

**Let grammar G = (V, T, P, S) be a context free grammar if recursive inference procedure tells that the terminal string w is in the language of the variable 'A', then there is a parse tree (or) derivation tree with root node 'A' and yield w.**

**PROOF**

To prove that if $A \Rightarrow w$ with zero (or) one (or) more steps then there is a parse tree with root node 'A'.

**By Induction Hypothesis Method**

**(i) Basis Step :** Let the number of steps used to infer the string is 1, then the tree has a single leaf node it may be either $A \Rightarrow a_1, a_2, a_3, \ldots \ldots \ldots \ldots a_n$ (or) $A \Rightarrow \xi$.

**Parse Tree**

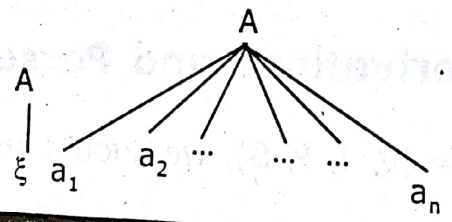



**Fig. 5.2.6** Parse Tree for $A \Rightarrow \varepsilon$

**(ii) Induction Step :** Let the number of steps used to derive the step is n+1. Assume that there is a parse tree for all the variables 'B' and string 'x' is inferred for the non terminal 'B' using n+1 number of steps. Consider, that the last step of inference that assumes w is the language of the variable (or) else non terminal 'A' then $A \Rightarrow X_1, X_2, \ldots\ldots X_k \ldots$ Then each $X_i$ is either a variable (or) terminal string. Then, we have to break up the string w having k components such that w having k components such that $w = w_1, w_2, \ldots\ldots w_k$.

**Two Cases**

(1) *If $X_i$ is a terminal, then $w_i = x_i$*

(2) *If $X_i$ is a variable, then $w_i$ is a string that was previously inferred to be in language L in almost 'n' steps.*

$$A \Rightarrow w_1, w_2, \ldots\ldots w_k.$$

$$A \overset{*}{\Rightarrow} w$$

oufastupdates.com

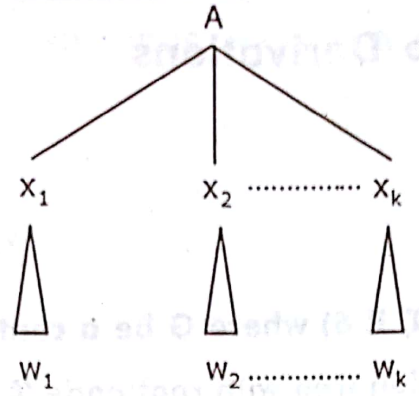



**Fig. 5.2.7** Tree Used in the Inductive Part of the Proof of Theorem

**Parse Tree :** $X_i \Rightarrow w_i$

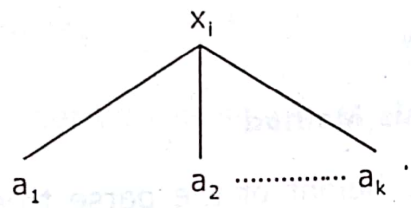



**Fig. 5.2.8** Parse Tree $X_i \Rightarrow W_i$

Then, the yield is obtained by concatenating the sub trees from left to right.

**Example :** $E \Rightarrow E + E$

$E \Rightarrow E * E$

$E \Rightarrow id$

**Parse Tree :** $w = id + id * id$

$= E + id * id$

$= E + E * id$

$= E + E * E$

$= E$

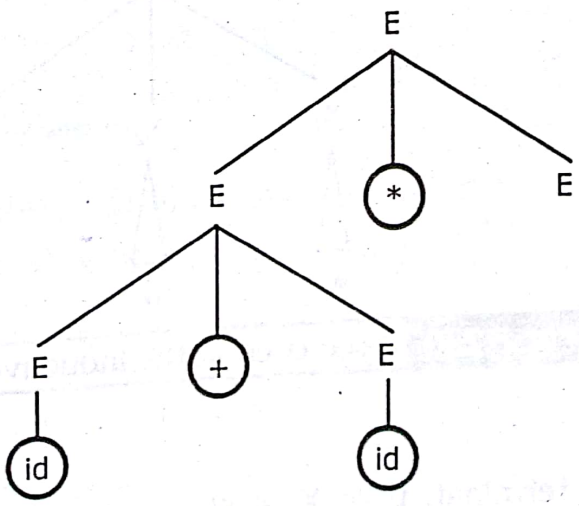



**Fig. 5.2.9** Parse Tree for String (E + E)*E

oufastupdates.com

## From Trees to Derivations

### THEOREM

**STATEMENT**

**Let grammar G = (V, T, P, S) where G be a context free grammar. Suppose, there is a parse tree (or) derivation tree with root node 'A' and yield 'w' (i.e.) $A \overset{*}{\Rightarrow} w$. Then, there is a left most derivation for a given grammar 'G'.**

**PROOF**

To prove that $A \underset{lmd}{\overset{*}{\Rightarrow}} w$

**By Induction Hypothesis Method**

**(i) Basis Step :** If the height of the parse tree is 1, then the parse tree has a root node 'A' and yield 'w',

$A \Rightarrow id$

$w = id$

**(ii) Induction Step :** If the height of the parse tree is n + 1. Assume that, there is left most derivative.

For every parse tree, left most derivation will be $A \underset{lmd}{\overset{*}{\Rightarrow}} x_1, x_2, \ldots x_n$

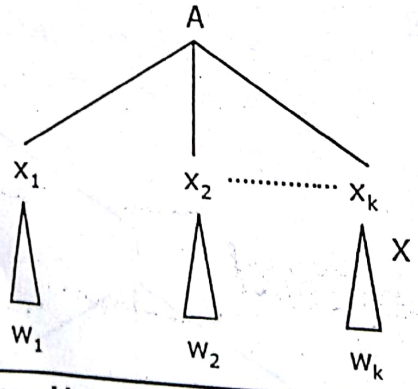



**Fig. 5.2.10** Tree Used in the Inductive Part of Theorem

**Two Cases**

(i) If $X_i$ is a terminal, then $X_i = w_i$

oufastupdates.com

(ii) If $X_i$ is a variable, then there must be root of some sub trees with yield w

By applying Induction hypothesis, there is a L.M.D.

$$X_i \overset{*}{\underset{LMD}{\Rightarrow}} w_i$$

$$A \Rightarrow X_1, X_2, \ldots\ldots X_k$$

For each i = 1, 2, ..... k.

$A \Rightarrow w_1, w_2, \ldots\ldots w_i$

(i) If $X_i$ is a terminal then there is no need to change any thing,

$$A \overset{*}{\underset{LMD}{\Rightarrow}} w_1, w_2, \ldots\ldots w_k$$

(ii) If $X_i$ is a variable then derive the string for each $X_i$ to $W_i$

$$X_i \underset{LMD}{\Rightarrow} a_1 \underset{LMD}{\Rightarrow} a_2 \ldots\ldots \underset{LMD}{\Rightarrow} a_n \underset{LMD}{\Rightarrow} w_i$$

$$\therefore \quad A \overset{*}{\underset{LMD}{\Rightarrow}} w_1, w_2, \ldots\ldots w_n$$

$$\therefore \quad A \overset{*}{\underset{LMD}{\Rightarrow}} w$$

## From Derivations to Recursive Inferences

THEOREM

**STATEMENT**

**Let grammar G = (V, T, P, S). Let G be a CFG if there is a derivation $A \overset{*}{\Rightarrow} w$ with zero (or) more than 1 step then there is a recursive inference procedure applied to a grammar. Determine that w is in the language of the variable (or) else non terminal 'A'.**

**PROOF**

**By Induction Hypothesis Method**

(i) **Basis Step :** If the derivation is the single step (i.e.) $A \overset{*}{\Rightarrow} w$ must be a production. $A \overset{*}{\Rightarrow} w$ where, w is in the language of variable 'A'.

(ii) **Induction Step :** Suppose the derivation takes 'n + 1' steps

$$A \Rightarrow X_1, X_2, \ldots\ldots X_k$$

Break w as $w_1, w_2, \ldots\ldots w_k$ where,

i) If $X_i$ is a terminal, then $X_i = w_i$

oufastupdates.com

(ii) If $X_i$ is a variable, then,

$$X_i \overset{*}{\Rightarrow} w_i$$

$$A \Rightarrow X_1, X_2, \ldots\ldots X_k$$

$$A \overset{*}{\Rightarrow} w_1, w_2, \ldots\ldots w_k$$

$$A \overset{*}{\Rightarrow} w$$

Hence, the theorem is proved.

**Example**

$$w = 01c10$$

$$E \Rightarrow 0E0/1E1/c$$

$$E \Rightarrow 0E0$$

$$\Rightarrow 01E10$$

$$\Rightarrow 01c10$$

$$E \Rightarrow 01c10$$

## Solved Problems

### SOLVED PROBLEM 1

**Draw derivation tree for the string abaaba for the CFG given by G where p is S → aSa | bSb | b | a | ε.**

**SOLUTION**

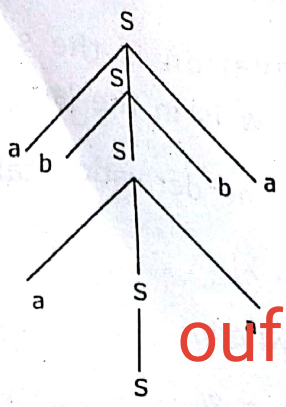
$S \rightarrow aSa$

$S \rightarrow abSba$

$S \rightarrow abaSaba$

$S \rightarrow aba\&aba$

$S \rightarrow abaaba$




oufastupdates.com

Consider the grammar G = ({A, S}, {a, b}, p, s) where p consist of,

$S \rightarrow aAS \mid a$

$A \rightarrow SbA \mid SS \mid ba$

Draw the derivation tree for string "aabbaa".

**SOLUTION**

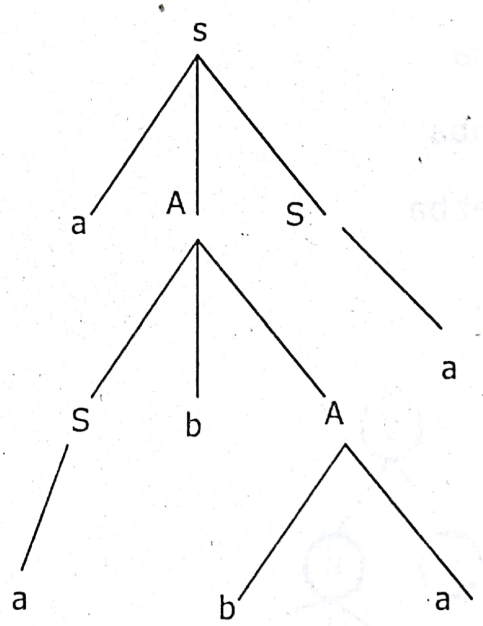
$S \rightarrow aAs$

$S \rightarrow aSbAs$

$S \rightarrow aabAs$

$S \rightarrow aabbaS$

$S \rightarrow aabbaa$




## SOLVED PROBLEM 3

Derive the string "aabbabba" for LMD, RMD and parse tree using CFG. By G where production is,

$S \rightarrow aB \mid ba$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

**SOLUTION**

Let G = ({S, A, B}, {a, b}, {S → aB, S → ba, A → a, A → aS, A → bAA, B → b, B → bS, B → aBB}, {S})

oufastupdates.com

**LMD**

$S \rightarrow aB$

$\rightarrow aaBB$

$\rightarrow aabB$

$\rightarrow aabbS$

$\rightarrow aabbaB$

$\rightarrow aabbabS$

$S \rightarrow aabbabba$

**RMD**

$S \rightarrow aB$

$\rightarrow aaBB$

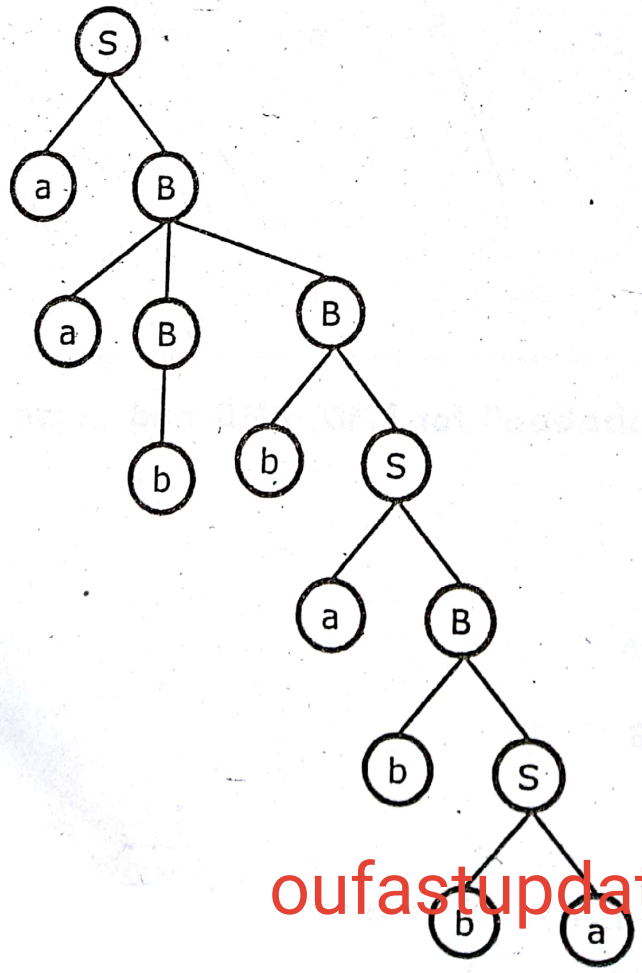
$\rightarrow aaBbS$

$\rightarrow aaBbba$

$\rightarrow aabSbba$

$S \rightarrow aabbabba$

**Parse Tree**




oufastupdates.com

**Consider the grammar G = ({S, A}, {a, b}, P, S) where P consists of s → aAS.|a, A → Sba | SS | ba. Derive "aabbaa" using LMD and RMD.**

**SOLUTION**

Let,

$$G = (\{S, A\}, \{a, b\}, \{S \to aAS, S \to a, A \to SbA, A \to SS, A \to ba\}, \{S\})$$

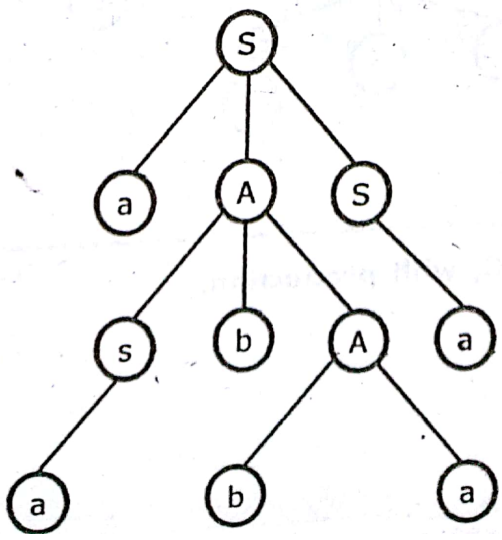
**LMD**

$$\begin{aligned} S &\to aAS \\ &\to aSbAS \\ &\to aabAS \\ &\to aabbaS \\ S &\to aabbaa \end{aligned}$$

**RMD**

$$\begin{aligned} S &\to aAS \\ &\to aAa \\ &\to aSbAa \\ &\to aSbbaa \\ S &\to aabbaa \end{aligned}$$

**Parse Tree**




oufastupdates.com

## SOLVED PROBLEM 1

**Design a CFG over {0, 1} such that each string in the corresponding language is of odd length and the middle character is 1. For example, 1, 010, 111, 00101, and 11100 are some valid strings in language.**

**SOLUTION**

1 is a valid string. Hence, we have the production S → 1. To ensure that each string is of odd length and S is in the middle, we use the productions S → ASA | BSB | ASB | BSA. Any transitive application of these productions will generate odd length of variables with S in the middle. To ensure the odd length of output string, each variable is associated with only one terminal.

So, the total CFG is as follows,

$$S \rightarrow ASA \mid BSB \mid ASB \mid BSA \mid 1$$

$$A \rightarrow 0$$

$$B \rightarrow 1$$

Let us generate the string 00101,

$$S \Rightarrow ASB \Rightarrow AASAB \Rightarrow 00101$$

## AMBIGUITY IN GRAMMARS AND LANGUAGES

Grammars can be used to put structure on programs and documents. The assumption was that a grammar uniquely determines a structure for each string in its language. However not every grammar does provide unique structure.

When a grammar fails to provide unique structures, it is sometimes possible to redesign the grammar to make the structure unique for each string in the language. Unfortunately sometimes we cannot do so. That is, there are some CFL's that are "inherently ambiguous", every grammar for the language puts more than one structure for some strings in the language.

### Ambiguous Grammars

A context free grammar G = {V, T, P, S} is an ambiguous consisting of at least one string $w \in T^*$ for which we can find two different parse tree i.e., by applying left most derivation and rightmost derivation, each with root labelled S and yield ω.

If each string has at most one parse tree in the grammar, then the grammar is *unambiguous*. A context free language L is inherently ambiguous if every CFG for L is *ambiguous*.

oufastupdates.com

**NOTE**

(1) Construct the left most derivations for the getting string ω and construct the right most for getting the string, when we construct the parse tree using both of derivations and if there are two different parse tree obtained then the of grammar is ambiguous.

(2) Apply leftmost derivations twice for getting string ω and construct parse tree. If two parse trees are different then the grammar is ambiguous.

(3) Apply rightmost derivations twice for getting string ω and construct parse tree. If two parse trees are different then the grammar is ambiguous.

Whether or not a grammar is ambiguous affects the complexity of parsing. Parsing algorithms for unambiguous grammar are more efficient then parsing algorithms for ambiguous grammars.

**Example :** Consider the grammar G of,

$$P = \{ \quad E \rightarrow E + E \mid E * E \mid id \quad \}$$

The derivations are,

**Right Most Derivation**

$$E \Rightarrow E + E$$
$$E \Rightarrow E + E * E$$
$$E \Rightarrow E + E * id$$
$$E \Rightarrow E + id * id$$
$$E \Rightarrow id + id * id$$

**left Most Derivation**

$$E \Rightarrow E * E$$
$$E \Rightarrow E + E * E$$
$$E \Rightarrow id + E * id$$
$$E \Rightarrow id + E * E$$

These two derivations can be represented in parse tree,

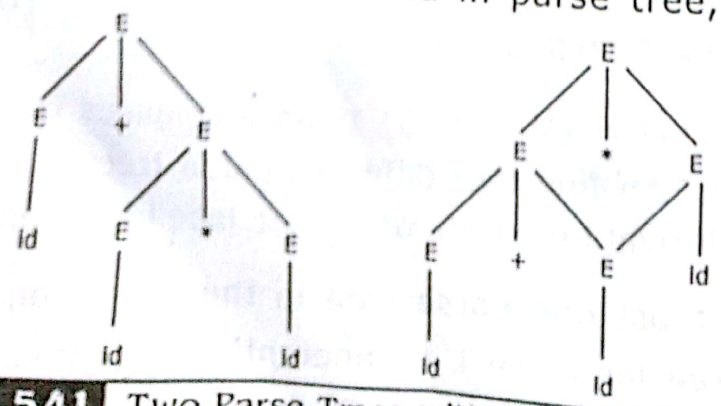



**Fig. 5.4.1** Two Parse Trees with the Same Yield.

oufastupdates.com

These two parse trees derivations has significant difference. The expression structure is concerned, in the right most derivation is the second and third are multiplied and final result is added to first expression where as in left most derivation add the first two expressions multiple the result by the third expression. Consider the example with different values of id using the parenthesis i.e., first (right most derivation) derivation is 2 + (3 + 4) = 14 and the second (left most derivation) derivation (2 + 3) * 4 = 20. These two expressions are not providing same value. So the parse tree structure also results in different way so the grammar is ambiguous.

In some cases, it is possible to modify a grammar to makes it unambiguous.

**Example :** The grammar $G^1$ can be modified as follows,

$$G^1 = \{\{S, T, F\}, \{+, *, (\ ), a\}, P, S\}$$

Where, P is the set of productions,

$S \rightarrow S + T$

$S \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (S)$

$F \rightarrow a$

## Removing Ambiguity From Grammars

In this section we should able to provide algorithms for removing ambiguity from CFG's but one more surprising thing is that, there is no algorithms for checking whether a CFG is ambiguous. An inherently ambiguous, grammar are context free language that have nothing but ambiguous, CFG's for these languages removal of ambiguous is impossible.

Consider the CF grammar for simple expressions,

(1) $E \Rightarrow E + E \Rightarrow E + E * E$

(2) $E \Rightarrow E * E \Rightarrow E + E * E$

In this grammar there are two causes of ambiguity,

(1) The precedence of operator is not considered while in the first grammar of property holds good the * before the + operator. Where as in the second grammar property holds good the + is ahead of *. Both grammar has a valid parse tree. But we need to get the structure of the first grammar equal to the second grammar for the result of valid unambiguous grammar.

oufastupdates.com

(2) Identical operator sequences can be combined either from left or right. In the above grammar * is replaced by + and we can see two different parse tree, for the string E + E + E. Since addition and multiplication are associative, it does not matter whether combining from left or right but to remove the ambiguity we should choose one. The good approach is to consume from the left results the correct grouping two + signs.

For this problem solution need to force the precedence for introducing more different variable, Every variable should express that share a level of "binding strength" specially,

(1) An expression *factor* cannot be brokened with adjacent operator, either a * or a +. The expression language factor is,

(i) Identifies, letting an Identifies is not possible to separate using operator attaching.

(ii) Given expression is parenthesized, does not matter what contains inside whether the parenthesis. But purpose of parenthesis is to prevent what is inside becomes operated using any operator outside the parenthesis.

(2) The expression *term* cannot be brokened using + operator. In this example * and + only operator, product of one or more factor is called term, for instance, a * b term can be split using left associalivity and place, $a_1$ * i.e., $a_1$ * a * b is combined as ($a_1$, *a) *b which splits a *b apart but placing term as $a_1$ + a * b is $a_1$ + (a * b) and a * b * $a_1$ is (a * b) + $a_1$.

(3) As whole expression finally refers any possible expression including these can be split by * or + adjacent.

**Example :** Consider the grammar,

E → X

E → E + E

E → E * E

E → (E)

X → a | b | *a | *b | XO | X1

Can be written as sum of one or more terms,

X → a | b | Xa | Xb | XO | X1

Y → X(E)

Z → Y | Z * Y

E → Z | E + Z

oufastupdates.com

**Show that the following grammar is ambiguous,**

$$S \rightarrow iCtS \mid iCtSeS \mid a$$

$$C \rightarrow b \mid ibtibtaea$$

**SOLUTION**

The string can ba obtain by applying left most derivation as shown below,

$$S \Rightarrow iCtSes$$

$$\Rightarrow ibtSes$$

$$\Rightarrow ibtictSes$$

$$\Rightarrow ibtibtses$$

$$\Rightarrow ibtibtaea$$

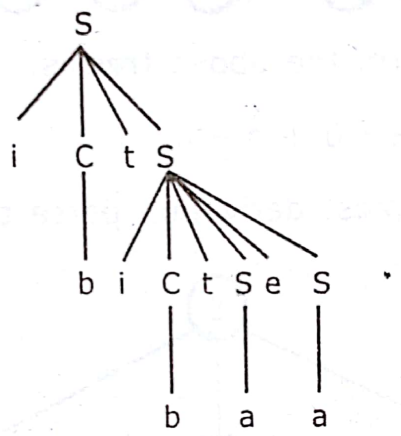
The parse tree for the above left most derivation is shown below,

oufastupdates.com

The string ibtibtaea can be obtained again by applying the left most derivation but using different set of productions as shown below,

$$S \Rightarrow icts$$
$$\Rightarrow ibts$$
$$\Rightarrow ibtictSes$$
$$\Rightarrow ibtibtSes$$
$$\Rightarrow ibtibtaes$$
$$\Rightarrow ibtibtaea$$

The parse tree for the above left most derivation shown below,




Therefore, there are two different parse tree for the string 'ibtibtaca' by applying leftmost derivation the above grammar is ambiguous.

## 5.4.4 Solved Problems

### SOLVED PROBLEM 1

**Show that the following grammar G is ambiguous. Also obtain the unambiguous grammar G for the following grammar.**

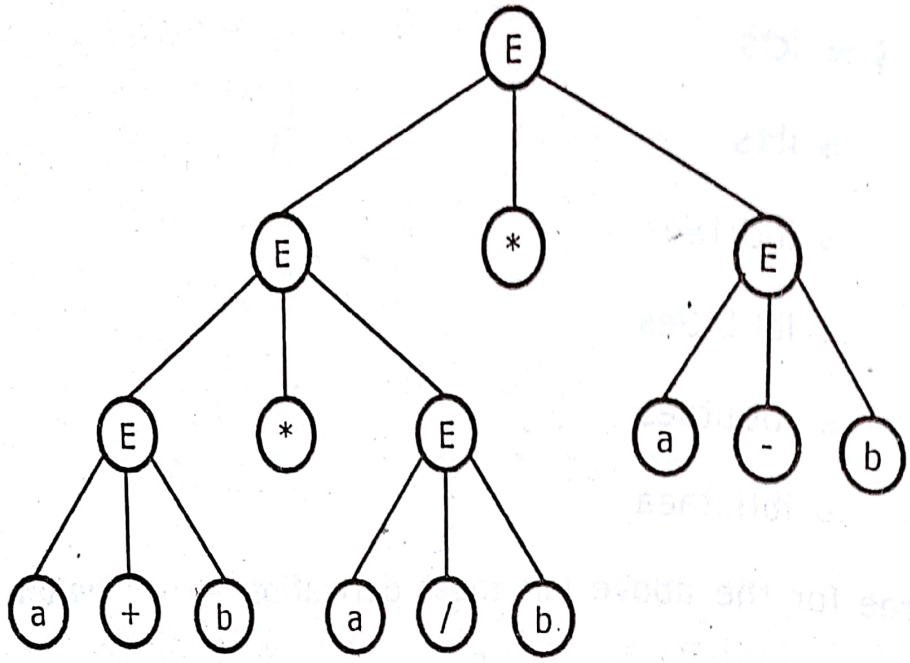
$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow E / E$$
$$E \rightarrow E - E$$
$$E \rightarrow (E) \mid a \mid b.$$

**Prove that L (G) > L (G¹)**

**Assume the operator + and – are left association and * ad/as right associative with**

oufastupdates.com

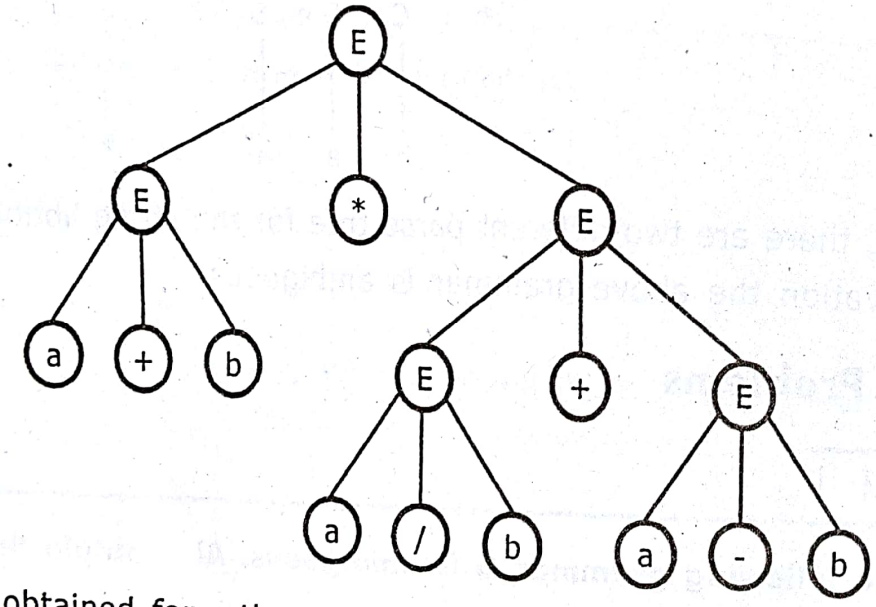
First the left most derivation parse tree as,




The result is obtained form the above tree is,

= a + b * a / b + a - b

The second one is right most derivation parse tree as,




The result is obtained form the above tree is,

a + b * a / b + a - b

So we have two parse tree for same languages so the above grammar is ambiguous.

So, the now unambiguous grammar will be,

S → T | S | T | S - T

T → F | F * T | F | T

F → I | (S)

I → a / b

oufastupdates.com

$G^1$ = {S, T, F, I}, {a, b, *, -, 1}, {P, S}.

**When the languages is said to be inherently ambiguous? Show that the following grammar is ambiguous,**

$$S \rightarrow AB \mid aaB$$

$$A \rightarrow a \mid Aa$$

$$B \rightarrow b$$

**Construct an ambiguous grammar equivalents to above grammar.**

**SOLUTION**

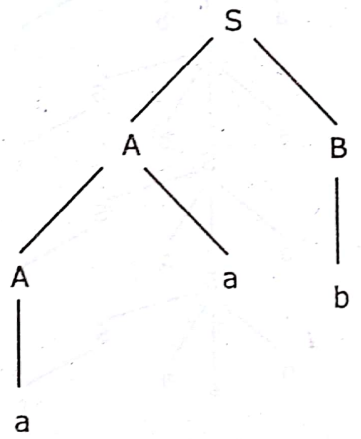
Consider the left most derivation for the string aab and the corresponding parse tree,

$$S \Rightarrow AB$$

$$\Rightarrow AaB$$

$$\Rightarrow aaB$$

$$\Rightarrow abb$$




Consider the left most derivation again for the string and but using different let of productions along with the parse tree,

$$S \Rightarrow aaB$$

$$\Rightarrow aaB$$

The unambiguous grammar can be written as,

$$S \rightarrow AB \mid SAB$$

$$A \rightarrow a \mid Aa$$

$$B \rightarrow b \mid Bb$$

oufastupdates.com

**Show that the following grammar is ambiguous,**

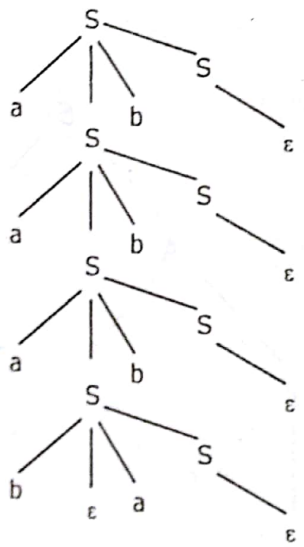
$$S \rightarrow aSbS$$

$$S \rightarrow bSaS \mid \varepsilon$$

**SOLUTION**

Consider the the left most derivation for the string aaababbb and corresponding parse tree,

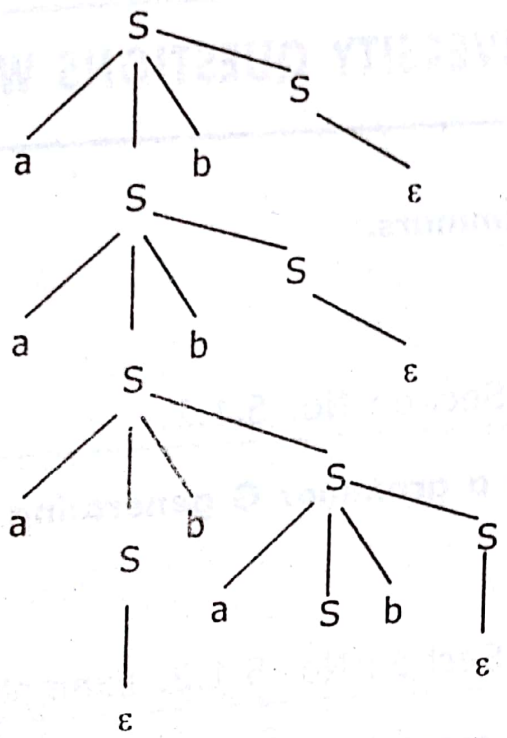
| | |
|---|---|
| $S \Rightarrow aSbS$ | By applying $S \rightarrow aSbS$ |
| $\Rightarrow aaSbSbS$ | By aaplying $S \rightarrow aSbS$ |
| $\Rightarrow aaaSbS\ b\ SbS$ | By aaplying $S \rightarrow aSbS$ |
| $\Rightarrow aaabSaS$ | By aaplying $S \rightarrow bSaS$ |
| $\Rightarrow aaabaSbSbSbS$ | By aaplying $S \rightarrow \varepsilon$ |
| $\Rightarrow aaabab\ SbSbS$ | By aaplying $S \rightarrow aSbS$ |
| $\Rightarrow aaababbb$ | By aaplying $S \rightarrow aSbS$ |




Consider the left most derivation again for the string aaababbb but using different of productions

| | |
|---|---|
| $S \Rightarrow aSbS$ | By aaplying $S \rightarrow aSbS$ |
| $\Rightarrow aaSbSbS$ | By aaplying $S \rightarrow aSbS$ |
| $\Rightarrow aaaSbSbSbSbS$ | |
| $\Rightarrow aaaSbaSbSbb$ | |
| $\Rightarrow aaabaSbSbSbb$ | |
| $\Rightarrow aaababsbb$ | |
| $\Rightarrow aaababbb$ | |

oufastupdates.com




Since, there are two parse tree S for the string aaababbb by applying left most derivation and right most derivation so the grammar is ambiguous.

oufastupdates.com

We have studied that the finite automaton accepts the regular languages but not every context free languages. This is because some context free languages are not regular. Intuitively, we understand that this is because finite automata have finite memories, whereas a context free language may require storing an unbounded amount of information for its recognition. Therefore, it is important to recognize a context free language for both practical and theoretical reasons. This suggests a new type of computation model with an extra component called stack. On introduction of stack, it provides additional memory beyond finite amount available in the control. This leads to a class of machines called pushdown automata.

In this chapter we discuss pushdown automata, acceptance of sets by pushdown automata, auxiliary and two stack pushdown automata. We also explore the relation between pushdown automata and context free languages.

## DEFINITION OF PUSHDOWN AUTOMATA (PDA)

A pushdown automata consist of a finite tape, a reading head which can read from the tape, a stack memory which is for read/write operation and operations are performed in Last-In-First-Out (LIFO) fashion. Like regular languages which are accepted by finite automata, context-free languages are also accepted by automata but not finite automata. They need a little more complex automata called pushdown automata.

Let us consider a context-free language $a^n b^n$. Any string of this language can be tested for membership for the language by a finite automaton if there is a memory element in it. Pushdown automata's makes use of stack (takes as memory element) that can store a's of a given input string. For example, as a's are read by the automaton, push them into the stack. When the symbol b appears stop storing a's and instead start popping up one by one every time when b is read. If another element, other than a or b is read after the first b, reject the string. When all the symbols of the input string are read, check the stack. If it is empty, accept the string. Otherwise reject it.

This automaton behaves like a finite automaton except the following two points,

(1) Its next state is determined not only by the input symbol being read, but also by, the symbol at the top of the stack.

(2) The contents of the stack can also be changed every time an input symbol is read. Thus its transition function specifies the new top of the stack contents as well as the next state. Model of pushdown automata is shown in Fig. 6.2.1.

oufastupdates.com

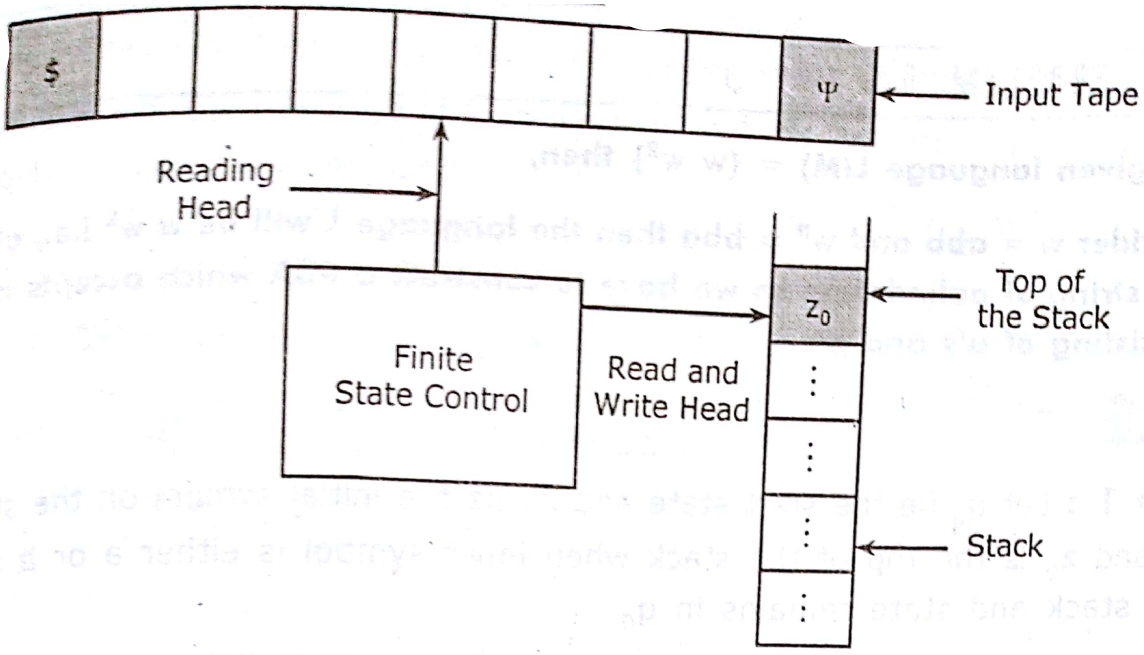



**Fig. 6.2.1** Model of Pushdown Automata

Here is input tape from which finite control (which works like finite control of finite automata) reads the input and at the same time it reads the symbol from stack top. After reading the inputs from the tape and stack pushdown automata moves in next state and write zero or more symbols on the stack. After writing on stack, the pointer to the top of the stack is automatically adjusted which then points to the top of the stack. This operation is carried out for entire memory and is considered as the final output produced by pushdown automata.

## The Formal Definition of Pushdown Automata

A Pushdown Automaton (PDA) involves seven tuples,

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

Where,

$Q$ = A finite set of states.

$\Sigma$ = A finite set of input symbols.

$\Gamma$ = A finite set of stack symbols.

$\delta$ = Transition function formally $\delta$ contains three arguments.

$Q \times (\Sigma \cup \varepsilon) \times \Gamma$ to sub set of $Q \times \Gamma^*$.

Each transition output is finite set of pairs from $Q \times \Gamma^*$.

$q_0$ = The start state i.e., $q_0 \in Q$.

$F$ = The set of accepting/final states, i.e., $F \subseteq Q$.

oufastupdates.com

**The given language L(M) = {w $w^R$} then,**

**Consider w = abb and $w^R$ = bba then the language L will be w $w^R$ i.e., abbbba which is a string of palindrome so we have to construct a PDA which accepts a palindrome consisting of a's and b's.**

**SOLUTION**

**STEP 1 :** Let $q_0$ be the start state and $Z_0$ be the initial symbol on the stack. If state $q_0$ and $z_0$ is the top of the stack when input symbol is either a or b pushed in to the stack and state remains in $q_0$.

The transition function is,

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

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

In $q_0$ state if we encounter ε then string empty need to be accepted and we should able to reach the accepting state, and so, we need one more transition as,

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

If first input symbol is pushed on to the stack then stack will contain either a or b. We have to keep pushing all the symbol into the stack till we encounter midpoint but irrespective of either a or b or what is present on the stack so the number of transitions can be,

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

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

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

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

**STEP 2 :** Once we encounter (assumption only) the midpoint if there is an input symbol and is same as the symbol in top of the stack, change the state to $q_1$ and remove an element from the stack using the transitions,

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

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

**STEP 3 :** In state $q_1$, if the input symbol is same as symbol on the top of the state remain in state $q_1$ and delete an element from the stack using the transitions,

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

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

oufastupdates.com

**STEP 4 :** Finally in state $q_1$, if input is empty and stack should be empty i.e., $z_0$ should contains stack then change the state to $q_f$ and do not after the contents of stack using the following transitions,

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

So, the PDA to accept the language M.

i.e., $L(M) = \{w\, w^R \mid w \in (a + b)^*\}$ is given by,

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

Where,

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

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

$\Gamma = (a, b, z_0)$

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

$$\delta(q_0, a, z_0) = (q_1, az_0)$$

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

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

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

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

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

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

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

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

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

$q_0 \in Q$ is the start state of

$z_0 \in G$ is initial stack symbols

$F = \{q2\}$ in final state.

**NOTE :** *The given PDA W $W^R$ is non determistic PDA since both the conditions are not satisfied.*

oufastupdates.com

## A Graphical Notation for PDA's

The graphical representation shows the number of transitions occurred during PDA but it is not so easy to understand. But, some time an generalized transition diagram will provide clear PDA behavior aspect.

So, we introduce a PDA graphical representation also called transition diagram of PDA in which,

(1) The number of nodes corresponds to number of states of PDA.

(2) Start state indicates an labelled arrow and doubly circled states are accepting/final states of automaton.

(3) An arc shows that transition of the PDA. An arc labelled a, x | ε from state q to state p means that δ(q, a, x) contains pair (p, ε) perhaps among different pairs. i.e., are label provides what input is used, also provides previous and current top of the stack.

The only thing that the transition diagram will not provide us in which is the start stack symbol conventionally, it is $z_0$, otherwise we indicate. The transition diagram for example 6.1 is shown below,

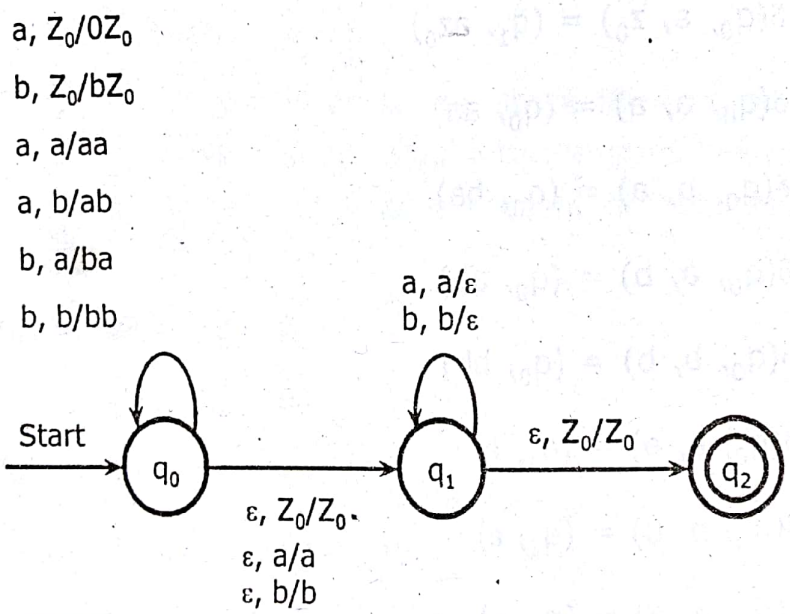



**Fig. 6.2.2** Representing a PDA as a Generalized Transition Diagram for $WW^R$

## Instantaneous Descriptions of a PDA

The step by step execution of the PDA on a input string can be given using the instantaneous descriptions. In instantaneous description, shortly referred as ID, there are three components and they are,

(1) Current state of the automata.

(2) Part of the input string yet to be read.

(3) Stack content from top to bottom.

oufastupdates.com

**Example :** The instantaneous description of the PDA of example 6.1 on the string bbbb, is as shown in Fig. 6.2.3.

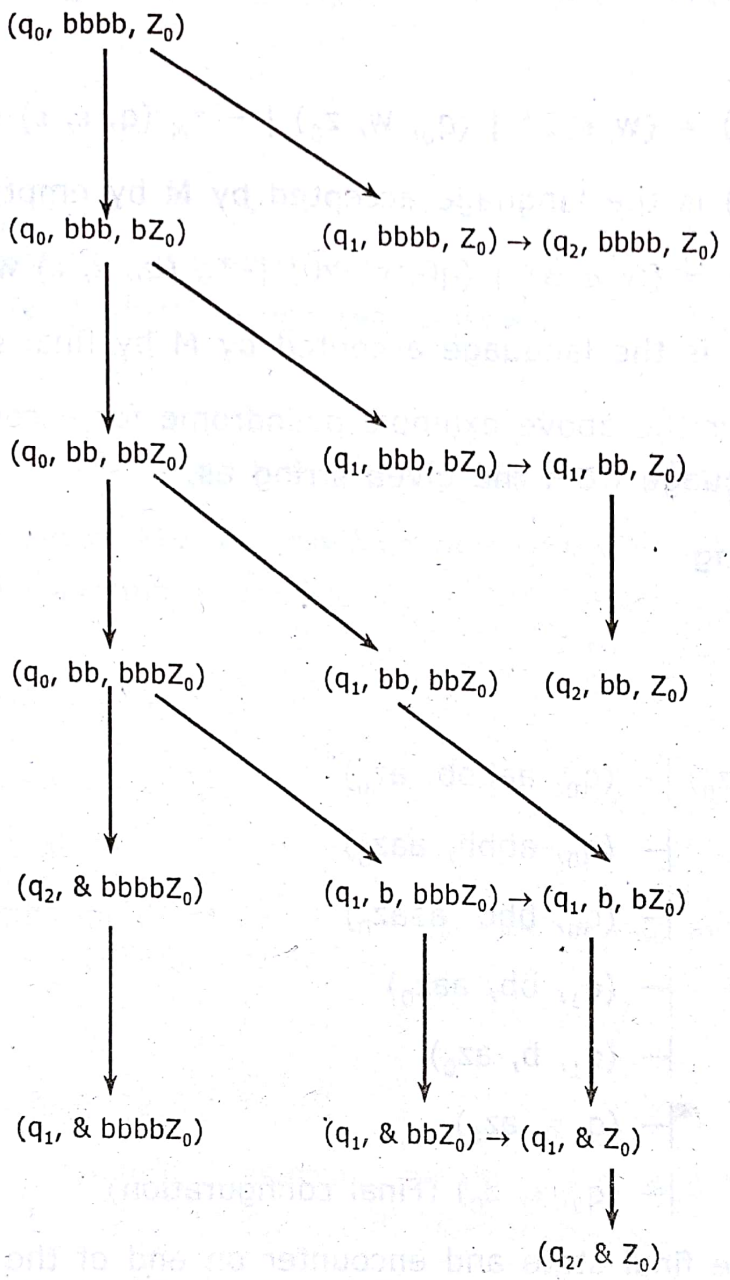



**Fig. 6.2.3** ID's of the PDA of Example 6.1 on Input bbbb

## THE LANGUAGES OF A PDA

There are two ways of defining acceptance by a PDA by accepting state or by empty stack,

Given a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ the following languages are defined.

### Acceptance Final State

The language accepted by a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ by final state T(M) is defined as,

$$T(M) = \{w \in \Sigma^* \mid (q_0, w, z_0) \vdash^*_M (q_f, \varepsilon, \alpha)$$

We say that T(M) is the language accepted by M final state.

oufastupdates.com

As we get $(s_0, w, z_0z'_0) \overset{*}{\underset{M_2}{\vdash}} (s, \wedge, z'_0)$ by applying rule 2 repeatedly $z'_0$ does not get affect at the bottom.

Therefore, we get,

$$(s_0, w, z_0) \overset{*}{\underset{M_2}{\vdash}} (s, \wedge, \wedge) \text{ for } s \in S.$$

Again by construction of rule 2, we have $(s_0, w, z_0) \overset{*}{\underset{M_1}{\vdash}} (s, \wedge, \wedge)$.

Therefore, we get $w \in N(M_1)$ and hence $L(M_2) \subseteq N(M_1)$. Thus we have $N(M1) \subseteq L(M_2)$ and $L(M_2) \subseteq N(M_1)$.

Hence, $N(M_1) = L(M_2) = L$.

## SOLVED PROBLEM 1

**Obtain PDA to recognize the language,**

$$L = \{a^i b^j \mid i < J \text{ and } i, j \geq 1\}$$

**To solve this problem, we need to keep track number of a's and look into number of b's are more.**

**SOLUTION**

Let $q_0$ be the start state and $z_0$ be the initial symbol on the stack,

**STEP 1 :** In state $q_0$ if the input symbol is a, push two a's on to the stack, the transitions are shown below,

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

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

**STEP 2 :** In state $q_0$ if the input symbol is b, change the state to $q_1$ and delete an element from the stack, using the transitions,

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

**STEP 3 :** In state $q_1$ if the input symbol is b, and top of the stack is a, remain in $q_1$, and delete an element from stack using the transition,

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

oufastupdates.com

**STEP 4 :** In state $q_1$ if i/p symbol is b and top of the stack is $z_0$, change the state of $q_2$ and do not alter contents of stack using the transition,

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

Finally,

$$\delta(q_2, b, z_0) = (q_2, z_0)$$

So the PDa is given by,

$$M = (Q, \Sigma, \Gamma, q_0, z_0, F)$$

Where,

$Q = \{q_0, q_1, q_f\}$.

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

$\Gamma = \{a, z_0\}$.

$q_0$ = the start state.

$F = \{q_f\}$.

$z_0$ is initial symbol on stack is shown below,

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

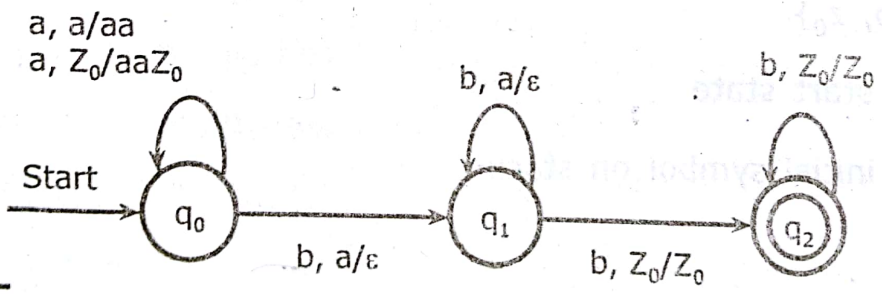
$$\delta(q_0, a, a) = (q_0, aaa)$$

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

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

$$\delta(q_2, b, z_0) = (q_2, z_0)$$

The transition diagram is shown below,




## SOLVED PROBLEM 2

**Obtain PDA to recognize the language,**

$$L = \{\omega \mid \text{no. of a's in } \omega = \text{no. of b's in } \omega\}.$$

**SOLUTION**

Let $q_0$ be the start state and $z_0$ be the initial symbol on the stack,

**STEP 1 :** In state $q_0$ if the stack is empty push the i/p symbol on to the stack using the transitions,

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

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

oufastupdates.com

**STEP 2 :** In state $q_0$ if the i/p symbol is same as the symbol on top of the stack, remain in $q_0$ and push the symbol into the stack using the transitions,

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

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

**STEP 3 :** If the i/p symbol is different from top of stack remain in $q_0$ and delete an element from the stack,

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

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

**STEP 4 :** In state $q_0$ if i/p is empty and top of the stack is $z_0$, change the state to $q_f$ and do not alter the contents of the stack,

$$\delta(q_0, \epsilon, z_0) = (q_f, z_0)$$

The PDA is given by,

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

Where,

$Q = \{q_0, q_f\}$

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

$\Gamma = \{a, b, z_0\}$

$q_0$ = the start state

$z_0$ = The initial symbol on stack.

$F = \{q_f\}$

And $\delta$ is shown below,

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

$$\delta(q_0, b, z_0) = (q_1, bz_0)$$

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

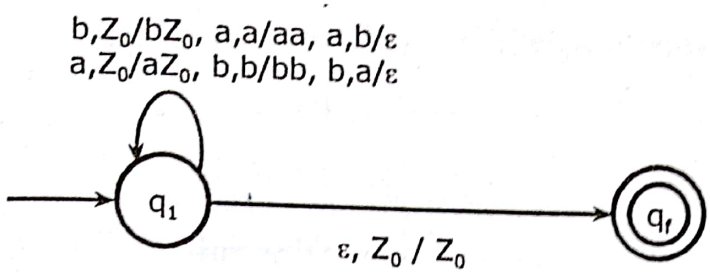
$$\delta(q_0, b, b) = (q_1, bb)$$

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

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

$$\delta(q_1, \epsilon, z_0) = (q_f, z_0)$$

oufastupdates.com




**NOTE :** *Since $\delta(q_0, \in, z_0)$ is not empty $\delta(q_0, a, z_0)$ or $\delta(q_0, b, z_0)$ should be empty for deterministic since $\delta(q_0, a, z_0)$ and $\delta(q_0, b, z_0)$ is not empty the given PDA is non deterministic PDA.*

## SOLVED PROBLEM 3

**Design the PDA to accept the language $L = \{w \mid w \in (a, b)^* \text{ and } n_a(w) > n_b(w)\}$.**

**SOLUTION**

The given language accepts the strings where number of a's are greater than the number of b's.

Let PDA be $M = (\{q_0, q_1\}, \{a, b\}, \{z_0, a, b\}, q_0, z_0, q_1)$

$\delta$ is given below,

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

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

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

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

$$\delta(q_1, a, z_0) = (q_1, az_0)$$

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

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

$$\delta(q_1, b, a) = (q_1, \in)$$

The transaction diagram for the PDA is shown in Fig. 6.3.2.

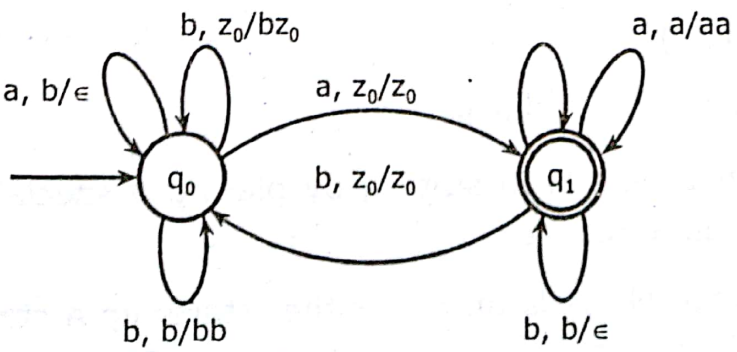



**Fig. 6.3.2** Transition Diagram

oufastupdates.com

For example consider the string abbabaa is,

$$(q_0, abbabaa, z_0) \vdash (q_1, bba, baa, z_0)$$
$$\vdash (q_0, ba\ baa, z_0)$$
$$\vdash (q_0, abaa, bz_0)$$
$$\vdash (q_0, baa, z_0)$$
$$\vdash (q_0, aa, bz_0)$$
$$\vdash (q_0, a, z_0)$$
$$\vdash (q_1, \epsilon, z_0) \qquad \text{(Accept)}$$

## SOLVED PROBLEM 4

**Find the PDA with only one state that accepts the language $\{a^m b^n : n > m\}$.**

**SOLUTION**

Let PDA be,

$$P = (Q, \Sigma, \tau, \delta, q_0, F)$$

Where,

$Q = \{s, q, f\}$

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

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

$F = \{f\}$

$\delta$ is given below,

(i) $\delta(S, \epsilon, \epsilon) = (q, c)$

(ii) $\delta(q, a, c) = (q, ac)$

(iii) $\delta(q, a, a) = (q, aa)$

(iv) $\delta(q, b, a) = (q, \epsilon)$.

(v) $\delta(q, b, c) = (f, \epsilon)$

(vi) $\delta(f, b, \epsilon) = (f, \epsilon)$

This automata works as follows,

- The transition 1 puts M in state q by placing a special symbol C on the bottom of the stack as a market.
- In state q, when M reads an a, it either starts up a stack of a's from the bottom while keeping the bottom marker (transition 2) or pushes an a onto a stack of a's (transition 3).

oufastupdates.com

In state q, when M reads b and topmost symbol on the stack is a then, it pops a (transition 4) implying that it has completed reading a sequence of a's. When the topmost symbol on the stack is c then the machine pops it and moves to a final state, f (transiton 5).

- In the final state f, if the input is left and stack is empty, then it reads and pops sequence of b's and remains in the final state.

The machine M acepts the string aabbb by using the following sequence of transitions.

| State | Input | Stack | Transition |
|---|---|---|---|
| s | aabbbb | ∈ | - |
| s | aabbbb | c | 1 |
| s | abbbb | ac | 2 |
| s | bbbb | aac | 3 |
| s | bbb | ac | 4 |
| s | bb | c | 4 |
| f | b | ∈ | 5 |
| f | ∈ | ∈ | 6 |

## SOLVED PROBLEM 5

**Construct the PDA that recognizes the languages $L = \{x = x^R : x \in \{a, b\}^+\}$.**

**SOLUTION**

Let PDA be,

$$P = (Q, \Sigma, \tau, \delta, q_0, F)$$

Where,

$Q = \{s, f\}$

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

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

$F = \{f\}$

oufastupdates.com

δ is given below,

(i) $\delta(s, a, \epsilon) = (s, a)$

(ii) $\delta(s, b, \epsilon) = (s, b)$

(iii) $\delta(s, =, \epsilon) = (f, \epsilon)$

(iv) $\delta(f, a, a) = (f, \epsilon)$

(v) $\delta(f, b, b) = (f, \epsilon)$

This automata works as follows,

- The machine M remains in its initial state s as long as it is completed reading half of its inputs. It uses transition 1 to 2 appropriately to push the input symbol from input string onto the stack.
- When M reads = from the input string, it chances its state from s to f without performing any operation of the stack.
- In state f, M pops the topmost symbol on the stack using transitions 4 and 5 if and only if it matches to the next input symbol. If the topmost symbol on the stack does not match the next input symbol then the machine halts.
- If machine reaches to the configuration $(f, \epsilon, \epsilon)$ - final state with end of input and empty stack then it is said that M accepts the input $x = x^R$.

The machine M accepts the input string bba = abb by using the following sequence of transitions.

| State | Input | Stack | Transition |
|---|---|---|---|
| S | bba = abb | ∈ | - |
| S | ba = abb | b | 2 |
| S | a = abb | bb | 2 |
| s | = abb | abb | 1 |
| f | abb | abb | 3 |
| f | bb | bb | 4 |
| f | b | c | 5 |
| f | ∈ | ∈ | 5 |

oufastupdates.com

## EQUIVALENCE OF PDA'S AND CFG'S

We are studying in the section, exercising the language defined by the PDA's are same as the context free languages. The equivalence plan is represented in the graphical representation as,

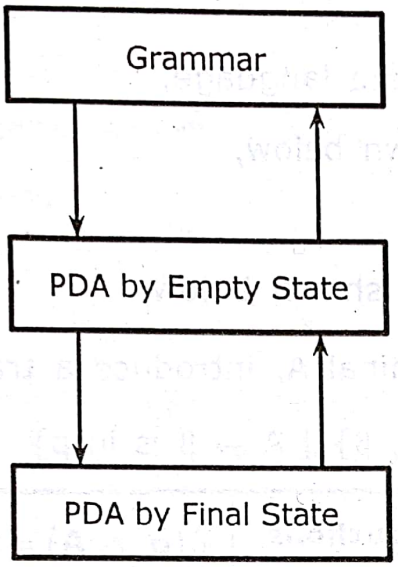



**Fig. 6.4.1** Organization of Constructions Showing Equivalence of Three Ways of Defining the CFL's

The aim of the section is to prove the following three different categories of languages,

(1) CFL, i.e., language defined by CFG's.

(2) Language accepted by final state by few PDA.

(3) Language accepted by empty stack by few PDA all are the same categories. We have already studied(2) and (3) are same.

### 6.4.1 From Grammars to Pushdown Automata Procedure

For every context free grammar G, there will be a pushdown automata M such that L(G) = L(M).

Let,

$$G = (v, \Sigma, p, S) \text{ be a CFG.}$$

We can obtain equivalent PDA,

$$M = (\{q\}, \Sigma, V \cup \Gamma, \delta, q, S, \phi)$$

Where $\delta$ can be defined as follows,

**STEP 1 :** For each non terminal A,

$$\delta(q, \varepsilon, A) = \{q, \beta\} \mid A \to \beta \text{ is in } p\}$$

**STEP 2 :** For each terminal a, introduce the transition

$$\delta(q, a, a) = \{q, \varepsilon\} \text{ for all } a \in \Sigma.$$

oufastupdates.com

**Obtain a PDA for the language,**

$$L = \{a^n b^n \mid n \geq 1\}.$$

**SOLUTION**

The grammar to generate the language,

$L = \{a^n b^n \mid n \geq 1\}$ is shown below,

$$S \rightarrow asb|ab$$

We can obtain, the PDA as shown below,

**STEP 1 :** For each non terminal A, introduce a transition as,

$$\delta(q, \varepsilon, A) = \{q, \beta\} \mid A \rightarrow \beta \text{ is in } p\}$$

| $A \rightarrow \beta$ **Productions** | $\delta(q, \varepsilon, A) = (q, \beta)$ **Transtions** |
|---|---|
| $S \rightarrow aSb$ | $\delta(q, \varepsilon, S) = (q, aSb)$ |
| $S \rightarrow ab$ | $\delta(q, \varepsilon, S) = (q, ab)$ |

**STEP 2 :** For each terminal a, introduce the transition as,

$\delta(q, a, a) = \{q, \varepsilon\}$ as shown below.

| **a Terminals** | $\delta(q, a, a) = (q, \varepsilon)$ **Transtions** |
|---|---|
| a | $\delta(q, a, a) = (q, \varepsilon)$ |
| b | $\delta(q, b, b) = (q, \varepsilon)$ |

So the final PDA of machine,

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$$

Where,

$Q = \{q\}$

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

$\Gamma = \{a, S, b\}$

$q_0 = \{q\}$

$F = \{q\}$

$z_0 = \phi$

oufastupdates.com

$\delta$ is shown below,

$\delta(q, \varepsilon, S) = (q, ab)$

$\delta(q, \varepsilon, S) = (q, aSb)$

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

$\delta(q, b, b) = (q, \varepsilon)$

Consider the string aabb generated by G using the following derivation,

$$S \Rightarrow aS\beta \Rightarrow aabb$$

Considering of M on aabb is,

(Initial ID)

$q(aabb, S) \vdash (q, a, abb, a\,Sb)$

$\vdash (q, abb, S\,b)$

$\vdash (q, abb, abb)$

$\vdash (q, bb, bb)$

$\vdash (q, b, b)$

$\vdash (q, \varepsilon, \varepsilon)$ [accepting configuration]

## EXAMPLE PROBLEM 6.4

**Given,**

**CFG L = ({S, A,}, {0, 1}, {p, S} where p consists of,**

**$S \rightarrow AS/\varepsilon$**

**$A \rightarrow 0A1/A1/01$.**

**SOLUTION**

**STEP 1 :** For each non terminal A, introduce a transition as,

$\delta(q, \varepsilon, A) = \{q, \beta\} \mid A \rightarrow \beta$ is in p}

| **$A \rightarrow \beta$ Productions** | **$\delta(q, \varepsilon, A) = (q, \beta)$ Transtions** |
|---|---|
| $S \rightarrow AS$ | $\delta(q, \varepsilon, S) = (q, AS)$ |
| $S \rightarrow \varepsilon$ | $\delta(q, \varepsilon, S) = (q, \varepsilon)$ |
| $A \rightarrow 0A1$ | $\delta(q, \varepsilon, A) = (q, 0A1)$ |
| $A \rightarrow A1$ | $\delta(q, \varepsilon, A) = (q, A1)$ |
| $A \rightarrow 01$ | $\delta(q, \varepsilon, A) = (q, 01)$ |

oufastupdates.com

**STEP 2 :** For each terminal 'a, introduce the transition as,

$\delta(q, a, a) = \{q, \varepsilon\}$ as shown below.

| a Terminal | $\delta(q, a, a) = (q, \varepsilon)$ Transtions |
|---|---|
| 0 | $\delta(q, 0, 0) = (q, \varepsilon)$ |
| 1 | $\delta(q, 1, 1) = (q, \varepsilon)$ |

So the final PDA of machine,

$$M = (\{q\}, \{0, 1\}, \{0, 1, A, S\}, \delta, q, S, \phi)$$

Where $\delta$ is defined by,

$$\delta(q, \varepsilon, S) = \{(q, AS)\}, (q, \varepsilon)\}$$

$$\delta(q, \varepsilon, A) = \{(q, 0A1)\}, (q, A1)\ (q, 01)\}$$

$$\delta(q, 0, 0) = \{(q, \varepsilon)\}$$

$$\delta(q, 1, 1) = \{(q, \varepsilon)\}$$

Consider the sentence 0011 generated by G using the following derivation,

$$S \Rightarrow AS \Rightarrow 0A1S \Rightarrow 0011S \Rightarrow 0011$$

Construction M on 0011 is,

(initial ID)

$q\ (0011, s) \vdash (q, 0011, As)$

$\vdash (q_1, 0011, 0A1s)$

$\vdash (q_1, 011, 011s)$

$\vdash (q_1, 0011, 0011)$

$\vdash (q_1, 11, 11)$

$\vdash (q_1, 1, 1)$

$\vdash (q_1, \varepsilon, \varepsilon)$ [Accepting configuration]

oufastupdates.com

## From PDA's to Grammars

### THEROEM

**Let $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ be PDA, there exists an equivalent grammar $G = \{V, T, P, S\}$ such that $L(M) = L(G)$.**

### PROOF

We shall construct $G = (V, T, P, S)$, where the set of variables consist of,

(i) The special symbol S, which is the start symbol.

(ii) All symbols of the form [pxq], where p and q are states in Q and x is a stack symbol in $\Gamma$.

The productions of G are as follows,

(i) For every transition of the form,

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

Introduce the productions of the form,

$$(q_i, zq_k) \rightarrow a\,(q_j, Aq_l)\,(q_lBq_k)$$

Where $q_k$ and $q_l$ take all possible values from Q.

(ii) For each transition of the form,

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

Introduce the production,

$$(q_i, zq_j) \rightarrow a$$

**NOTE :** *This procedure, may introduce useless symbols. So, finally unless symbols should be eliminated.*

### XAMPLE PROBLEM 6.5

**Obtain the CFG that generates the language for the PDA shown below, with the following transitions,**

$$\delta(q_0, a, z) = (q_0, Az)$$

$$\delta(q_0, b, A) = (q_0, AA)$$

$$\delta(q_0, a, A) = (q_1, \varepsilon).$$

### OLUTION

To obtain an CFG from the PDA, all the given transitions could be of the form,

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

(or)

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

oufastupdates.com

(i) For each trasitions of the form,

$\delta(q_i, a, z) = (q_j, AB)$ introduce the production

$\delta(q_i, zq_k) \rightarrow (q_j, A\ q_l)\ (q_l\ Bq_k)$

| Given Transitions $\delta\ (q_i, a, z) = (q_j, AB)$ | Resulting Productions $(q_i\ zq_k) \rightarrow a\ (q_j\ Aq_l)\ (q_l\ Bq_k)$ |
|---|---|
| - $\delta\ (q_0, a, z) = (q_0, Az)$ | $\rightarrow a\ (q_0\ Aq_0)\ (q_0zq_0)$<br>$\rightarrow a\ (q_0\ Aq_1)\ (q_1zq_0)$<br>$(q_0, z, q_1) \rightarrow a\ (q_0\ Aq_0)\ (q_0zq_1)$<br>$(q_0, z, q_1) \rightarrow a\ (q_0\ Aq_1)\ (q_1zq_1)$ |
| - $\delta\ (q_0, b, A) = (q_0, AA)$ | $(q_0, z, q_0) \rightarrow b\ (q_0\ Aq_0)\ (q_0Aq_0)$<br>$\rightarrow b\ (q_0\ Aq_1)\ (q_1Aq_0)$<br>$(q_0, zq_1) \rightarrow b\ (q_0\ Aq_0)\ (q_0Aq_1)$<br>$\rightarrow b\ (q_0\ Aq_1)\ (q_0Aq_1)$ |

(ii) For every transition of the form,

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

Introduce the production,

$(q_i, zq_j) \rightarrow a.$

| Given Transition $\delta\ (q_{i0}, a, z) = (q_j, \varepsilon)$ | Resulting Productions $(q_i\ zq_k) \rightarrow a$ |
|---|---|
| $q\ (q_0, a, A) = (q_1, \varepsilon)$ | $(q_0, Aq_1) \rightarrow a$ |

So, final grammar $G = \{v, \Sigma, p, s\}$

Where,

$V = (q_0zq_0), (q_0zq_1)\ (q_0Aq_0)\ (q_0Aq_1)$

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

P = RHS of resulting productions.

$S = \{q_0, zq_0\}$

oufastupdates.com

## SOLVED PROBLEM 1

**Convert the following context free grammar to push down automata,**

**$S \rightarrow aA \mid bB.$**

**$A \rightarrow aB \mid a.$**

**$B \rightarrow b.$**

**SOLUTION**

Given CFG is,

$$S \rightarrow aA \mid bB.$$

$$A \rightarrow aB \mid a.$$

$$B \rightarrow b.$$

**STEP 1 :** The given grammar is already in GNF. So, directly converting it into PDA as follows,

**STEP 2 :** Consider the production,

$$S \rightarrow aA$$

This production is of the form $A \rightarrow \gamma$.

$$A = S$$

$$a = a$$

$$\gamma = A$$

$$\therefore \quad \delta(q, a, s) = (q, A)$$

Consider the second production,

$$S \rightarrow bB$$

$$A = S$$

$$a = b$$

$$\gamma = B$$

$$\therefore \quad \delta(q, b, S) = (q, B)$$

Consider the third production,

$$A \rightarrow aB$$

$$A = A$$

$$a = a$$

$$\gamma = B$$

$$\therefore \quad \delta(q, a, A) = (q, B)$$

oufastupdates.com

Consider the fourth production,

$$A \rightarrow a$$

$$A = A$$

$$a = a$$

$$\gamma = \epsilon \text{ (Empty string)}$$

$$\therefore \quad \delta(q, a, A) = (q, E)$$

Consider the last production,

$$B \rightarrow b$$

$$A = B$$

$$a = b$$

$$\gamma = \epsilon$$

$$\therefore \quad \delta(q, b, B) = (q, \epsilon)$$

The PDA for the given grammar is,

$$M = (\{q\}, \{a, b\}, \{S, A, B\}, \delta, q, S, \phi)$$

Where 'δ' is defined as,

$$\delta(q, a, S) = (q, A)$$

$$\delta(q, b, S) = (q, B)$$

$$\delta(q, a, A) = (q, B)$$

$$\delta(q, a, A) = (q, \epsilon)$$

$$\delta(q, b, B) = (q, \epsilon)$$

## SOLVED PROBLEM 2

**Construct the PDA for the following grammar, S → AA|a A → SA|b**

**SOLUTION**

Let, PDA be $P = (Q, \Sigma, \Gamma, \delta, S, F)$

Where,

$$Q = \{p, q\}$$

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

$$\Gamma = \{S, A, a, b\}$$

$$S = \{P\}$$

$$F = \{q\}$$

oufastupdates.com

$\delta$ is given as follows,

$R_1 : \delta(p, \epsilon, \epsilon) = \{(q, S)\}$

$R_2 : \delta(q, \epsilon, S) = \{(q, AA)\}$

$R_3 : \delta(q, \epsilon, S) = \{(q, a)\}$

$R_4 : \delta(q, \epsilon, A) = \{(q, SA)\}$

$R_5 : \delta(q, \epsilon, A) = \{(q, b)\}$

$R_6 : \delta(q, a, a) = \{(q, \epsilon)\}$

$R_7 : \delta(q, b, b) = \{(q, \epsilon)\}$

Let us apply this on string aabb accepted by the given grammar.

| States | Input | Stack | Rule Used |
|---|---|---|---|
| P | aabb | $\epsilon$ | - |
| q | aabb | S | $R_1$ |
| q | aabb | AA | $R_2$ |
| q | aabb | SAA | $R_4$ |
| q | aabb | aAA | $R_3$ |
| q | abb | AA | $R_6$ |
| q | abb | SAA | $R_4$ |
| q | abb | aAA | $R_3$ |
| q | bb | AA | $R_6$ |
| q | bb | bA | $R_5$ |
| q | b | A | $R_7$ |
| q | b | b | $R_5$ |
| q | $\epsilon$ | $\epsilon$ | $R_7$ |

oufastupdates.com

## SOLVED PROBLEM 3

**Construct the PDA corresponding to the grammar,**

**$S \rightarrow aABB|aAA.$**

**$A \rightarrow aBB|a.$**

**$B \rightarrow bBB|A.$**

**(or)**

**Let G be the grammar given by,**

**$S \rightarrow aABB|aAA$**

**$A \rightarrow aBB|a$**

**$B \rightarrow bBB|A$**

**Construct the PDA that accepts the language generated by this grammar G.**

**SOLUTION**

Let PDA be,

$$P = (Q, \Sigma, \Gamma, \delta, D, F)$$

Where,

$Q = \{p, q\}$

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

$\Gamma = \{S, A, B, a, b\}$

$S = \{p\}$

$F = \{q\}$

$\delta$ is given as follows,

$R_1 : \delta(p, \epsilon, \epsilon) = \{(q, S)\}$

$R_2 : \delta(q, \epsilon, S) = \{(q, aABB)\}$

$R_3 : \delta(q, \epsilon, S) = \{(q, aAA)\}$

$R_4 : \delta(q, \epsilon, A) = \{(q, aBB)\}$

$R_5 : \delta(q, \epsilon, A) = \{(q, a)\}$

$R_6 : \delta(q, \epsilon, B) = \{(q, bBB)\}$

$R_7 : \delta(q, \epsilon, B) = \{(q, A)\}$

$R_8 : \delta(q, a, a) = \{(q, \epsilon)\}$

$R_9 : \delta(q, b, b) = \{(q, \epsilon)\}$

oufastupdates.com

Let us apply this on string aaaaa accepted by the given grammar.

| States | Input | Stack | Rule Used |
|---|---|---|---|
| p | aaaaa | ∈ | - |
| q | aaaaa | S | $R_1$ |
| q | aaaaa | aAA | $R_2$ |
| q | aaaa | AA | $R_8$ |
| q | aaaa | aBBA | $R_4$ |
| q | aaa | BBA | $R_8$ |
| q | aaa | ABA | $R_7$ |
| q | aaa | aBA | $R_5$ |
| q | aa | BA | $R_8$ |
| q | aa | AA | $R_7$ |
| q | aa | aA | $R_5$ |
| q | a | A | $R_8$ |
| q | a | a | $R_5$ |
| q | ∈ | ∈ | $R_8$ |

oufastupdates.com

# DETERMINISTIC PUSHDOWN AUTOMATA

According to the definition of PDA which is an nondeterministic but deterministic is an sub case and also it is quite important. So, in this section, we shall define deterministic PDA's and find out some of things they can and cannot do.

## Definition of Deterministic PDA

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ be a PDA.

Where,

$Q$ = A finite set of states i.e., $(q_0, q_1, q_2 \ldots)$.

$\Sigma$ = A finite set of input.

$\Gamma$ = A finite set of stack symbols.

$\delta$ = Transistion function (or) mapping function.

$q_0$ = An initial state.

$z_0$ = Initial symbol in stack.

$F$ = The set of final states.

If the PDA is deterministic, it should satisfy following two conditions,

(1) $\delta(q, a, z_0)$ has only one member for any $q$ in $Q$, $a \in \Sigma$ or $a = \varepsilon$ an a $z$ in $\Gamma$.

(2) If $\delta(q, \varepsilon, z_0)$ is non empty i.e., must be defined then $\delta(q, a, z_0)$ should empty.

These two conditions should be satisfied for the PDA is *deterministic* and if one of the condition is failed then the PDA is *non deterministic*.

**NOTE :** *Not empty i.e., transitions is defined empty i.e., transition is not defined.*

## EXAMPLE PROBLEM 6.6

**The following language is an deterministic PDA or non deterministic PDA,**

$$L = (M) = \{\omega \mid \omega \in (a + b)^* \text{ and } n_a(\omega) = n_b(\omega)\}.$$

**SOLUTION**

The trasitions are as follows,

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

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

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

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

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

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

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

oufastupdates.com

The PDA should satisfy the two conditions shown in the definition of deterministic,

(1) The first condition is $\delta(q_0, a, z_0)$ should have only one member. In this use for each $q \in Q$, $a \in \Sigma$ and $z \in \Gamma$. There exist only one member. So, the first condition is satisfied.

(2) The second condition states that $\delta(q, \varepsilon, z_0)$ is not empty then $\delta(q, a, z_0)$ should be empty. Consider the transition,

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

Since the transition is defined, the transition $\delta(q_0, a, z_0)$ where $a \in \Sigma$ should not be defined but there are two transitions,

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

$$\delta(q_0, b, z_0) = (q_0, b, z_0) \text{ defined from state } q_0$$

When the top of the stack is $z_0$. Since the second condition is not satisfied. So, the given PDA is *non deterministic.*

## Regular Languages and Deterministic PDA's

The DPDA's accept a class of languages that is between the regular languages and the CFL's. So we have to prove that the DPDA languages include all the regular languages.

**THEOREM**

**If L is a regular language, then L = L(P) for some DPDA P.**

**PROOF**

Essentially, a DPDA can simulate a deterministic finite automaton. The PDA keeps some stack symbol $Z_0$ on its stack, because a PDA has to have a stack, but really the PDA ignores its stack and just uses its state.

Formally, let $A = (Q, \Sigma, \delta_A, q_0, F)$ be a DFA.

Construct DPDA $P = (Q, \Sigma, \{Z_0\}, \delta_P, q_0, Z_0, F)$

By defining $\delta_P(q, a\ Z_0) = \{(p, Z_0)\}$ for all states p and q in Q, such that $\delta_A(q, a) = p$.

We assume that $(q_0, w, Z_0) \overset{*}{\underset{P}{\vdash}} (p, \in, Z_0)$ if and only if $\hat{\delta}_A\ (q_0, w) = P$.

That is, P simulates A using its state. The proofs in both directions are easy inductions on |w| and we leave them for the reader to complete. Since both A and P accept by entering one of the states of F, we conclude that their languages are the same.

If we want the DPDA top accept by empty stack, then we find that our language recognizing capability is rather limited. Say that a language L has the prefix property if there are no two different strings x and y in L such that x is a prefix of y.

oufastupdates.com

## DPDA's and Context Free Languages

Like a finite automaton, a PDA can be deterministic or nondeterministic. A deterministic PDA (DPDA) is one in which every transition $\delta(q_i, a, Y)$ has at the most one implication. However, in case of PDA there is a possibility of the non real input symbol $\in$ as well. A transition function $\delta(q_i, a, Y)$ can always be preceded by another transition of the form $\delta(q, \in, Y)$.

Thus, even if the output of the transition function $\delta(q_i, a, Y)$ is singleton the behaviour of the PDA can still be nondeterministic depending upon whether $\delta(q_i, \in, Y)$ is executed or not. Hence, for a PDA to be deterministic it is necessary that for every $a \in \Sigma$, for every $q_i \in Q$, and for every $Y \in \Gamma$,

(1) $\delta(q_i, a, Y)$ has at the most one implication.

(2) If $\delta(q_i, a, Y)$ is nonempty, then $\delta(q_i, \in, Y)$ is empty.

A CFL that is accepted by a deterministic PDA is known as deterministic CFL. A deterministic language must have deterministic grammar as well. A production of the form $S \rightarrow XSX \mid Y$ in a CFG creates a nondeterministic PDA with transition functions as follows.

$T_1: (q_0, \in, S) \vdash (q_0, XSX)$

$T_2: (q_0, \in, S) \vdash (q_0, Y)$

oufastupdates.com

**Obtain the PDA for the following grammar,**

**S → a ABC**

**A → a B/a**

**B → b A/b**

**C → a.**

**SOLUTION**

**STEP 1 :** For each non-transition A, introduce a transition as,

$$\delta(q, \varepsilon, A) = \{(q, B) \mid A \to B \text{ is in } p\}$$

| A → B Production | δ (q, ε , A) = (q, A) Transitions |
|---|---|
| S → a ABC | δ (q, ε , S) = (q, aABC) |
| A → aB | δ (q, ε , A) = (q, aB) |
| B → bA | δ (q, ε , B) = (q, bA) |
| A → a | δ (q, ε , A) = (q, a) |
| B → b | δ (q, ε , B) = (q, b) |
| C → a | δ (q, ε , C) = (q, a) |

**STEP 2 :** For each terminal a, introduce the transitions as,

δ(a, a, a) = (q, ε) as shown below,

| a Terminal | δ (q, a , a) = (q, ε ).Transitions |
|---|---|
| a | δ (q, a, a) = (q, ε ) |
| b | δ (q, b, b) = (q, ε ) |

So the find PDA of machine,

$$M = (\{q\}, \{a, b\}, \{S, A, B, C, z_0\}, \delta\{q\}, S\phi)$$

oufastupdates.com

$\delta$ is shown below,

$$\delta(q, \varepsilon, S) = (q, aABC)$$

$$\delta(q, \varepsilon, A) = (q, aB)$$

$$\delta(q, \varepsilon, A) = (q, a)$$

$$\delta(q, \varepsilon, B) = (q, bA)$$

$$\delta(q, \varepsilon, B) = (q, b)$$

$$\delta(q, \varepsilon, C) = (q, a)$$

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

$$\delta(q, b, b) = (q, \varepsilon)$$

Consider the sentence aabba generated by grammar G using the following derivation,

$$S \Rightarrow aABC$$

$$\Rightarrow aaBBC$$

$$\Rightarrow aabBC$$

$$\Rightarrow aabbC$$

$$\Rightarrow aabba$$

Construction of M on aabba is,

(Initial ID)

$$q(aabba, S) \vdash (q, abba, a\,ABC)$$

$$\vdash (q, abba, ABC)$$

$$\vdash (q, abba, aBBc)$$

$$\vdash (q, bba, BBc)$$

$$\vdash (q, bba, bBc)$$

$$\vdash (q, ba, Bc)$$

$$\vdash (q, ba, bc)$$

$$\vdash (q, a, c)$$

$$\vdash (q, a, a)$$

$$\vdash (q, \varepsilon, \varepsilon) \text{ (Accepting configuration)}$$

oufastupdates.com

# *Long Questions ( UNIT 3)*

-> Normal Forms for Context free Grammars

-> Pumpping lemma

-> Chomsky Normal Form

-> Reduce grammar to CNF

oufastupdates.com

# NORMAL FORMS FOR CONTEXT FREE GRAMMARS

In this section we should prove every CFL (with out ε) is generated by a CFG in which all productions are of the form E → TF or E → a where E, T and F are variables and a is a terminal. This representation form is called *Chomsky Normal Form.*

Initially, we need to study different preliminary simplification are useful in different ways,

(1) Substitution method.

(2) Left recursion.

(3) Eliminating of useless symbols (grammar makes more complex).

(4) Eliminating of *unit productions* and (derivation longer).

(5) Eliminating of *ε-productions* and (derivation longer).

## Eliminating Useless Symbols

A symbol of G is useful, when it appears on R.H.S of a production and can generate some terminal string. Otherwise, it is said to be useless symbol.

**Example**

Consider the grammar,

$$S \rightarrow AB|C$$

$$A \rightarrow a$$

$$B \rightarrow b$$

Here, the non-terminals are S, A and B.

Let us derive a string from this grammar. We start with start symbol S,

$$S \Rightarrow AB$$

$$\Rightarrow aB$$

$$\Rightarrow ab$$

Thus, we are able to derive certain string.

But, let us take S → C. We cannot move further, becuase there are no rules for C, which can derive a terminal string.

That means, C is a useless symbol and we can remove it.

Then the grammar becomes,

$$S \rightarrow AB$$

$$A \rightarrow a$$

$$B \rightarrow b.$$

oufastupdates.com

THEOREM

**Let $G = (V, T, P, S)$ be a context free grammar and assume that $L(G) \neq 0$ i.e., grammar generates at least one string. An equivalent grammar $G^1 = \{V^1, T^1, P^1, S^1\}$ can be obtained by eliminating useless symbols and productions such that $L(G) = L(G^1)$.**

PROOF

Consider symbol A is said to be useful if $A \overset{*}{\Rightarrow} T^*$ and $S^* \overset{*}{\Rightarrow} \alpha AB \overset{*}{\Rightarrow} w$ where w in $T^*$. This means that A can generate terminal strings or variable or symbols and is reachable from start symbol S. But these symbols are not used while deriving them these symbol are useless if it is not useful and all these productions are also useless.

The grammar $G^1$ can be obtained in two stages from grammar G.

**Stage 1 :** We compute $V^1$, the set of variable A for which can derive terminal strings. $A \overset{*}{\Rightarrow} w$ where $w \in T^*$. At the end we eliminate all symbols in $V - V^1$. The set $V^1$ is computed by following procedure,

(i) Initialize $V^1 \leftarrow \{A \mid A \overset{*}{\Rightarrow} T^*\}$

(ii) Each and every production of the form $A \rightarrow w$, $w \in (V^1 \cup T)^*$,

- Copy V to $V^1$.
- Add all the elements $V^1$ to V.

(iii) Repeat step 2, until the V can not be added to $V^1$.

In this algorithm we should start with non terminals whose right hand side (RHS) is a terminal symbol in step-1 where as in the step-2, each and every production of the form $A \rightarrow w$ where w is used for generating terminal strings and production A is copied to $V^1$.

**NOTE :** *Variable $V^1$ contains only variables which contains string of terminals. The productions are obtained $V^1$ is added to $P^1$ and terminal of these productions are added to $T^1$. The grammar $G^1 = (V^1, S^1, P^1, S)$ contain only variable of A in $V^1$ such that $A \overset{*}{\Rightarrow} w$. Where $w \in T^*$. Therefore the derived grammar $G^1$ is $L(G^1) = L(G)$.*

**Stage 2 :** We collect all group of variable and terminals reachable from start symbol S. The production is not used are useless. The procedure is follows,

(i) Select reachable states from start $V^1 \leftarrow \{s\}$.

(ii) For each A in $V^1$. If there is a production $A \rightarrow \alpha X \beta$ then,

- Add variable X to $V^1$.
- Add terminal in $\beta$ to $T^*$.

(iii) Repeat the step 2 while there is no modification in $V^1$.

In this all the variables or terminals are not reachable from start symbol is eliminated.

oufastupdates.com

**Eliminate useless symbols from the following grammar,**

$S \rightarrow aA \mid B$

$A \rightarrow aB \mid B$

$B \rightarrow aB \mid b \mid bC$

$D \rightarrow Ea$

$E \rightarrow a \mid aE \mid bC.$

**SOLUTION**

**Stage 1 :** Apply the algorithm given in stage (1), we can obtain set of variables and the productions from which we are getting string of terminal as follows.

| $V^1$ | V | Productions |
|---|---|---|
| $\{\phi\}$ | B, E | $B \rightarrow b$<br>$E \rightarrow a$ |
| B, E | B, E, A, D | $A \rightarrow aB \mid B$<br>$B \rightarrow aB$<br>$D \rightarrow Ea$<br>$E \rightarrow aE$ |
| B, E, A, D | B, E, A, D, S | $S \rightarrow a\,A \mid B$ |
| B, E, A, D, S | B, E, A, D, S | - |

So, the grammar from which, we get only string of terminals is given by,

$G^1 = \{V^1, T^1, P^1, S\}$

Where,

$V^1 = \{B, E, A, D, S\}$

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

$P^1 = \{$

$B \rightarrow b \mid aB$

$E \rightarrow a \mid aE$

$A \rightarrow aB \mid B$

$D \rightarrow Ea$

$S \rightarrow a\,A \mid B$

$\}$

oufastupdates.com

**Stage 2 :** Eliminate the symbol is productions which are not reachable from the start symbol.

| $V^1$ | $T^1$ | Productions |
|---|---|---|
| S | | $S \rightarrow aA \mid B$ |
| A, B | a, b | $A \rightarrow aB \mid B$<br>$B \rightarrow b \mid aB$ |

So, the final grammar obtained after eliminating useless symbols and productions is given by,

$$G^1 = \{V^1, T^1, P^1, S\}$$

Where,

$V^1 = \{S, A, B\}$

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

$P^1 = \{$

$S \rightarrow aA \mid B$

$A \rightarrow aB \mid B$

$B \rightarrow b \mid aB$

$\}$

## AMPLE PROBLEM 7.2

**Eliminate useless symbols from the following grammar,**

**$S \rightarrow AB \mid aAC$**

**$A \rightarrow aA \mid b$**

**$B \rightarrow bB \mid a$**

**$D \rightarrow aA \mid bB$**

**$E \rightarrow aD \mid bD.$**

**SOLUTION**

**Stage 1 :** Find the set of variable and the productions from which we get only string of terminals using the following algorithm.

oufastupdates.com

| $V^1$ | V | Productions |
|---|---|---|
| $\{\phi\}$ | A, B | $A \to b$<br>$B \to a$ |
| A, B | A, B, S, D | $S \to AB$<br>$A \to aA$<br>$B \to bA$<br>$D \to aA$<br>$D \to bB$ |
| A, B, S, D | A, B, S, D, E | $E \to aD$<br>$E \to bD$ |
| A, B, S, D, E | A, B, S, D, E | - |

So, the grammar from which, we get only string of terminals is given by,

$$G^1 = \{V^1, T^1, P^1, S\}$$

Where,

$V^1 = \{A, B, S, D, E\}$

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

$P^1 = \{$

$S \to AB$

$A \to b \mid aA$

$B \to a \mid bB$

$D \to aA \mid bB$

$E \to aD \mid bD$

$\}$

***Stage 2 :*** Eliminate the symbol is productions which are not reachable from the start symbol.

oufastupdates.com

| $V^1$ | $T^1$ | Productions |
|---|---|---|
| S | - | $S \to AB$ |
| S, A, B | a, b | $A \to aA \mid b$<br>$B \to bB \mid a$ |

So, the final grammar obtained after eliminating useless symbols and productions is given by,

$$G^1 = \{V^1, T^1, P^1, S\}$$

Where,

$V^1 = \{S, A, B\}$

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

$P^1 = \{$

$S \rightarrow AB$

$A \rightarrow aA \mid b$

$B \rightarrow bB \mid a$

$\}$

Therefore useless symbols are aAC, D and E are eliminated.

## EXAMPLE PROBLEM 7.3

**Eliminate useless symbols from the following grammar,**

**$S \rightarrow aA \mid bB$**

**$A \rightarrow aA \mid a$**

**$B \rightarrow bB$**

**$D \rightarrow ab \mid Ea$**

**$E \rightarrow ac \mid d.$**

**SOLUTION**

**Stage 1 :** Find the set of variable and the productions from which we get only string of terminals using the following algorithm.

| $V^1$ | V | Productions |
|---|---|---|
| $\{\phi\}$ | A, D, E | $A \rightarrow a$<br>$D \rightarrow ab$<br>$E \rightarrow d \mid ac$ |
| A, D, E | A, D, E, S | $S \rightarrow aA$<br>$A \rightarrow aA$<br>$D \rightarrow Ea$ |
| A, D, E, S | A, D, E, S | - |

oufastupdates.com

So, the grammar from which, we get only string of terminals is given by,

$$G^1 = \{V^1, T^1, P^1, S\}$$

Where,

$V^1 = \{A, D, E, S\}$

$T^1 = \{a, b, c\}$

$P^1 = \{$

$A \rightarrow a \mid aA$

$D \rightarrow ab \mid Ea$

$E \rightarrow d \mid ac$

$S \rightarrow aA$

$\}$

S is start symbol.

**Stage 2 :** Eliminate the symbol is productions which are not reachable from the start symbol.

| $V^1$ | $T^1$ | **Productions** |
|---|---|---|
| S | a | $S \rightarrow aA$ |
| S, A | a, b | $A \rightarrow a \mid aA$ |

So, the final grammar obtained after eliminating useless symbols and productions is given by,

$$G^1 = \{V^1, T^1, P^1, S\}$$

Where,

$V^1 = \{S, A\}$

$T^1 = \{a\}$

$P^1 = \{$

$S \rightarrow aA$

$A \rightarrow a \mid aA$

$\}$

S is start symbol.

oufastupdates.com

## Computing the Generating and Reachable Symbols

X is generating if $X^* \Rightarrow w$ for some terminal string w.

X is reachable if there is a derivation $s^* \Rightarrow \alpha X \beta$ for some $\alpha$ and $\beta$

(1) **Generating Symbols :** Let G = (V, T, P, S) be a grammar.

To compute the generating symbols of G, we need to follow the below steps,

(i) Every symbol of T is generating therefore it generates itself.

(ii) If $A \rightarrow \alpha$, then A is also generating for $\alpha$ CT or $\alpha = \epsilon$

**Example :** Consider the grammar,

$$E \rightarrow eB/e$$

$$A \rightarrow a$$

Compute generating symbols.

**Solution :** The set of generating symbols are,

(i) All terminals are generating by itself, so {e, a}.

(ii) Since $E \rightarrow eB$ has no generating symbols for B. So B has to be eliminated. Then E has one more production $E \rightarrow e$ which is having generating symbol e and like so for A.

Therefore the generating symbols are {e, a, E, A}.

(2) **Reachable Symbols :** Let G = (V, T, P, S) be a grammar. The following steps are used to find reachable symbols,

(i) S is reachable. [S Start symbol]

(ii) If A is reachable, then all productions with A in the head, all symbols of those productions are also reachable.

**Example :** Consider the grammar $E \rightarrow AB \mid a$

Find the reachable symbols.

**Solution :** Since E is the start symbol. It has to be include. Then E has production bodies AB and a. So E, A, B and a are reachable.

Therefore the reachable set = {E, A, B, a}.

oufastupdates.com

## Eliminating ε-Productions

A production $A \to \varepsilon$ is called a *null production*. If a grammar contains ε-productions then these can be removed without changing the meaning of the grammar.

**Example**

Consider the grammar,

$$S \to as \mid bs \mid \in$$

The above grammar contains only one $\in$ production i.e., $S \to \in$. It can be removed by placing $S \to \in$ at the place of S,

When we place $S \to \in$ in $S \to aS$, we get $S \to a$.

When we place $S \to \in$ in $S \to bS$, we get $S \to b$.

After eliminating null production the grammar becomes,

$$S \to aS \mid bS \mid a \mid b$$

### THEOREM

**Let G = {V, T, P, S} be a CFG where L(G) ≠ e, we can compute effectively new grammar $G^1$ without e productions such that $L(G^1) = L(G) - e$.**

**PROOF**

The grammar $G^1$ can be obtained from G as follows,

**Algorithm :** Finding NULLABLE productions.

**STEP 1 :** Initialize : NULLABLE ← $\{A \mid A \to \varepsilon\}$.

**STEP 2 :** For each and every productions $A \to w$, where $w \in$ NULLABLE* then,

(i) Copy V NULLABLE to $V^1$.

(ii) Add all the elements $V^1$ to V NULLABLE.

**STEP 3 :** Repeat step 2, until the V NULLABLE cannot be added to $V^1$.

*In this algorithm* we should start with NULLABLE is initialized to the set of *productions which directly* can derive a empty string. Where as in the step 2, If there *is an production* $A \to w$ and w can derive empty string so the A can also derive an *empty string*. Repeat this process until not getting any more NULLABLE.

oufastupdates.com

## Eliminating Unit Productions

A production in which one non-terminal derives another non-terminal, i.e., $A \rightarrow B$ is called a unit production.

**Example**

$S \rightarrow Aa \mid B$

$A \rightarrow a \mid bc \mid B$

$B \rightarrow A \mid bb$

In this example,

$S \rightarrow Aa$

$A \rightarrow a \mid bc$

$B \rightarrow bb$ are not an unit productions.

But unit productions are,

$S \rightarrow B$

$A \rightarrow B$

$B \rightarrow A$

Since B can be generated from S itself or B can be generated from A and A can be generated from B i.e., vice versa,

$S \rightarrow Aa \mid bb$

$A \rightarrow bb$

$B \rightarrow a \mid bc$

Hence the production,

$S \rightarrow B$

$A \rightarrow B$

$B \rightarrow A$ can be deleted.

THEOREM

**Let $G = \{V, T, P, S\}$ be a context free grammar without e productions and has a unit productions. Then we can obtain context free grammar $G^1 = \{V^1, S^1, P, S\}$ without unit productions such that $L(G^1) = L(G)$ i.e., any language generated by G also can be generated by $G^1$ without unit productions.**

PROOF

The unit production in grammar G can be removed from the following steps.

(i) Any unit production of the form $A \rightarrow A$ can be removed from the grammar without effect.

oufastupdates.com

(iii) Consider the special use

$A \rightarrow B$

$B \rightarrow A$

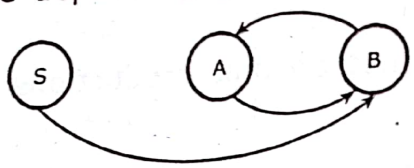
These unit productions are not removed just like that, But first find for each A, all variable B such that $A \overset{*}{\Rightarrow} B$.

(iv) Consider the unit productions and obtain the dependency graph for example if we have the productions of the form

$S \rightarrow B$

$B \rightarrow A$

$A \rightarrow B$ then the depending graph can be written as,




(v) From the dependency graph, it is clear that all non unit productions generated from B can be generated from A as well as S for all non unit productions.

(vi) Finally delete all unit productions from the grammar G.

The resulting grammar $G^1$ generates the same language as generated by G without any unit productions.

## EXAMPLE PROBLEM 7.6

**Eliminate unit productions form the following grammar,**

**$S \rightarrow Aa \mid B \mid Ca$**

**$B \rightarrow aB \mid b$**

**$C \rightarrow Db \mid D$**

**$D \rightarrow E \mid d$**

**$E \rightarrow ab.$**

**SOLUTION**

The non unit productions of the grammar G are as follows,

$S \rightarrow Aa$

$S \rightarrow Ca$

$B \rightarrow aB$

$B \rightarrow b$

$C \rightarrow Db$

$D \rightarrow d$

$E \rightarrow ab$

oufastupdates.com

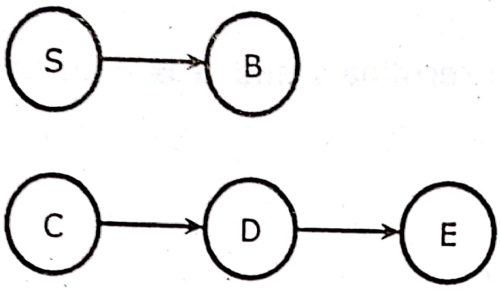
The unit productions of the grammar G are as follows,

$S \rightarrow B$

$C \rightarrow D$

$D \rightarrow E$

The dependency graph is shown below,




It is clear from the first dependency graph $S \overset{*}{\Rightarrow} B$ and so all the non unit production of B can be generated by S,

$S \overset{*}{\Rightarrow} B$ i.e., $S \rightarrow aB \mid b$

All non unit productions of D can be generated by C,

$C \overset{*}{\Rightarrow} D$ i.e., $C \rightarrow d$

$C \overset{*}{\Rightarrow} E$ i.e., $C \rightarrow ab$

All non unit productions of E can be generated by D,

$D \overset{*}{\Rightarrow} E$ i.e., $D \rightarrow ab$.

The final grammar without unit productions can be obtained as,

$G^1 = \{V^1, \Sigma^1, P^1, S\}$

$V^1 = \{S, A, B, C, D, E\}$

Where,

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

P1 = {

$S \rightarrow Aa \mid Ca \mid aB \mid b$

$B \rightarrow aB \mid b$

$C \rightarrow Db \mid d \mid ab$

$D \rightarrow d \mid ab$

$E \rightarrow ab$

}

oufastupdates.com

## Chomsky Normal Form

CNF puts restrictions on number of symbols on the right side of a production. In other words the strings, on the right side of a production, consist of not more than two symbols.

Formally, a context-free grammar G is said to be is CNF, if all productions are of the form,

$$A \rightarrow BC$$

(or) $$A \rightarrow a$$

Where, A, B, C are non-terminals and a is a terminal.

**Example**

(1) The grammar,

$$S \rightarrow BS \mid a$$
$$B \rightarrow SB \mid b$$

Is in CNF.

(2) The grammar,

$$S \rightarrow BS \mid BSB$$
$$B \rightarrow SB \mid aa$$

Is not in CNF, because both the productions $S \rightarrow BSB$ and $B \rightarrow aa$ violate the conditions of CNF.

(3) The grammar,

$$S \rightarrow AB \mid CA$$
$$B \rightarrow BC \mid AB$$
$$A \rightarrow a$$
$$C \rightarrow aB \mid b$$

Is not in CNF, because $C \rightarrow aB$ violates the conditions.

## Reduction of CFG to CNF

**THEOREM**

**For every CFL, without $\epsilon$ generated by a grammar $G_1$, there is an equivalent grammar $G_2$ in CNF.**

**PROOF**

Let, $G_1 = (V, T, P, S)$ be the CFG generating a language not containing $\epsilon$. We construct $G_2 = (V, T, P, S)$ as follows,

***STEP 1* (Eliminate $\epsilon$-Productions and Unit Productions) :** Eliminate all $\epsilon$ and unit productions from $G_1$, by using an appropriate method. If a production contains a single terminal symbol at right, it is in acceptable form. Add all such productions to P′ and all variables to V′.

oufastupdates.com

**STEP 2 (Eliminate Terminals of R.H.S) :** All productions in P of the form $A \to BC$ or $A \to a$ are included in P′ and all variables of such productions are included in V′. Consider a production of the form $A \to x_1x_2 \ldots x_n$, $n \geq 2$. If $x_i$ is a terminal 'a', introduce a new variable '$T_a$' and replace the terminal $x_i$ by $T_a$. These new productions are added to P′ and new variables are added to V′.

**STEP 3 (Restrict the Number of Variables on RHS) :** All productions, that are already in acceptable form of P′, are added to P″ and all respectively variables are included in V″. Consider the production $A \to A_1A_2 \ldots A_n$, $n \geq 3$. Now, introduce new production $A \to A_1K_1$, $A_2K_2$, ... $K_{n-2} \to A_{m-1}A_m$ to P″ and respective new variables $K_1$, $K_2$, ... $K_{n-2}$ are added to V′.

Thus, we get $G_2$ in CNF.

Hence proved.

## EXAMPLE PROBLEM 7.7

**Consider the grammar ({S, A, B}, {a, b}, P, S) that has the productions,**

**$S \to bA|aB$, $A \to bAA|aS|a$, $B \to aBB|bS|b$ and find an equivalent grammar in CNF.**

**SOLUTION**

**STEP 1 :** There are no $\epsilon$-productions and no unit productions.

**STEP 2 :** The only productions already in proper form are $A \to a$ and $B \to b$.

So we begin by replacing terminals on the right by variables, except in the case of $A \to bA$ is replaced by $S \to C_bA$ and $C_b \to b$.

Similarly,

$A \to as$ is replaced by $A \to C_aS$ and $C_a \to a$

$B \to bS$ is replaced by $B \to C_bS$ and $C_b \to b$

$B \to aBB$ is replaced by $B \to C_aBB$

$A \to bAA$ is replaced by $A \to C_bAA$, $S \to aB$ by $S \to C_aB$.

**STEP 3 :** The production $A \to C_bAA$ is replaced by $A \to C_bD_1$ and $D_1 \to AA$ production $B \to C_aBB$ is replaced by $B \to C_aD_2$ and $D_2 \to BB$.

The productions for the grammar in CNF are,

$S \to C_bA \mid C_aB$

$A \to C_aS \mid C_bD_1 \mid a$

$B \to C_bS \mid C_aD_2 \mid b$

$D_1 \to AA$

$D_2 \to BB$

$C_a \to a$

$C_b \to b$

oufastupdates.com

**Find a grammar in CNF equivalent to,**

**$S \rightarrow aAbB$**

**$A \rightarrow aA|a$**

**$B \rightarrow bB|b$.**

**SOLUTION**

**STEP 1 :** There are no unit productions or null productions.

**STEP 2 :** Let $G_1 = (V_N', \{a, b\}, P_1, S)$,

Where, $P_1$ and $V_N'$ are constructed as follows,

(i) $A \rightarrow a$, $B \rightarrow b$ are added to $P_1$.

(ii) $S \rightarrow aAbB$, $A \rightarrow aA$, $B \rightarrow bB$ yield,

$$S \rightarrow C_aAC_bB$$
$$A \rightarrow C_aA$$
$$B \rightarrow C_bB$$
$$C_a \rightarrow a$$
$$C_b \rightarrow b$$
$$V_N' = \{S, A, B, C_a, C_b\}$$

**STEP 3 :** $P_1$ consists of,

$$S \rightarrow C_aAC_bB$$
$$A \rightarrow C_aA$$
$$B \rightarrow C_bB$$
$$C_a \rightarrow a$$
$$C_b \rightarrow b$$
$$A \rightarrow a$$
$$B \rightarrow b$$
$$S \rightarrow C_aAC_bB$$

*Is replaced by,*

$$S \rightarrow C_aC_1$$
$$C_1 \rightarrow AC_2$$
$$C_2 \rightarrow C_bB$$

*The remaining productions in $P_1$ are added to $P_2$.*

*Let,*

$$G_2 = (\{S, A, B, C_a, C_b, C_1, C_2\}, \{a, b\}, P_2, S)$$

oufastupdates.com

Where, $P_2$ consists of,

$$S \rightarrow C_aC_1$$
$$C_1 \rightarrow AC_2$$
$$C_2 \rightarrow C_bB$$
$$A \rightarrow C_aA$$
$$B \rightarrow C_bB$$
$$C_a \rightarrow a$$
$$C_b \rightarrow b$$
$$A \rightarrow a$$
$$B \rightarrow b$$

$G_2$ is in CNF.

## EXAMPLE PROBLEM 7.9

**Reduce the following grammar G to CNF,**

**$S \rightarrow aAD$**

**$A \rightarrow aB|bAB$**

**$B \rightarrow b$**

**$D \rightarrow d$.**

**SOLUTION**

Let, $G_1 = (V_N', \{a, b, d\}, P_1, S)$,

Where, $P_1$ and $V_N'$ are constructed as follows,

$B \rightarrow b$, $D \rightarrow d$ are included in $P_1$.

$S \rightarrow aAD$ gives rise to $S \rightarrow C_aAD$ and $C_a \rightarrow a$.

$A \rightarrow aB$ gives rise to $A \rightarrow C_aB$.

$A \rightarrow bAB$ gives rise to $A \rightarrow C_bAB$ and $C_b \rightarrow b$.

$P_1$ consists of,

$$S \rightarrow C_aAD$$
$$A \rightarrow C_aB \mid C_bAB$$
$$B \rightarrow b$$
$$D \rightarrow d$$
$$C_a \rightarrow a$$
$$C_b \rightarrow b$$

oufastupdates.com

| S.No. | CNF | GNF |
|---|---|---|
| (1) | CNF is named after Noam Chomsky. | GNF is named after Sheila Greibach. |
| (2) | CNF puts restrictions on number of symbols, on the right side of production. | GNF puts restrictions not on the length of R.H.S, of production, but on the positions, in which terminals and variables can appear. |
| (3) | CFG is in CNF, only if all productions are of form $A \rightarrow BC$ or $A \rightarrow a$. | CFG is in GNF, only if all the productions are of the form $A \rightarrow a\alpha$. |
| (4) | CNF is important, because it yields efficient algorithms. | GNF is used,<br>(i) To prove that every CFL can be accepted by non-deterministic PDA.<br>(ii) To construct a PDA equivalent of CFG.<br>(iii) And because it is convinent to use in parsing. |

oufastupdates.com

**Eliminate useless symbols for the case given below,**

$S \rightarrow aAa \mid aBc$

$A \rightarrow aS \mid bD$

$B \rightarrow aBa \mid b$

$C \rightarrow abb \mid DD$

$D \rightarrow aDa.$

**SOLUTION**

The representation of given grammar is,

$$G = (\{S, A, B, C, D\}, \{a, b\}, P, S\}$$

oufastupdates.com

To simplify G, we have to eliminate useless symbols.

Here, S, A, B, C and D are non-terminals.

Let us derive a string, we have to start from start symbol s.

$S \rightarrow aBC$

$S \rightarrow abC$

$S \rightarrow ababb$

Thus, by following these rules we get certain string,

$S \rightarrow aAa$

$S \rightarrow aSa$

$S \rightarrow aBCa$

$S \rightarrow ababba$

By following these rules we get string,

$S \rightarrow aAa$

$S \rightarrow abDa$

$S \rightarrow abaDaa$

In this derivation the non-terminal D does not give any terminal string. That is by following D's rule, given $D \rightarrow aDa$, we simply get ample number of D's but no significant string.

Hence, we can declare D as a useless symbol.

Thus, the grammar after removing D is,

$S \rightarrow aAa|aBC$

$A \rightarrow aS$

$B \rightarrow aBa|b$

$C \rightarrow abb$

We find that, there is no path for the non-terminal $B \rightarrow aBa$. After removing it, The optimized grammar is,

$S \rightarrow aAa|aBC$

$A \rightarrow aS$

$B \rightarrow b$

$C \rightarrow abb.$

oufastupdates.com

**Find grammar in CNF equivalent to the grammar S → ~ S | [S > S] | p | q (S being the only variable).**

**SOLUTION**

Let, $G_1 = (V'_N, T, P_1, S)$,

Where, $P_1$ and $V'_N$ are constructed as follows,

(i) $S \to p \mid q$ are added to $P_1$.

(ii) $S \to \sim S$ induces $S \to AS$ and $A \to \sim$.

(iii) $S \to [S > S]$ induces,

$S \to BSCSD$

$B \to [\ ,\ C \to >,\ D \to ]$

$V'_N = \{S, A, B, C, D\}$

$P_1$ consists of,

$S \to p \mid q$

$S \to AS$

$A \to \sim$

$B \to [$

$C \to >$

$D \to ]$

$S \to BSCSD$

$S \to BSCSD$, B 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,

$S \to p \mid q \mid AS \mid BC_1$

$A \to \sim,\ C_1 \to SC_2$

$B \to [,\ C_2 \to CC_3$

$C \to >,\ C_3 \to SD$

$D \to ]$

$G_2$ is in CNF equivalent to the given grammar.

oufastupdates.com

**Reduce the following grammar to CNF,**

$$G = (\{S\}, \{a, b, c\}, \{S \rightarrow a|b|cSS\}, S).$$

**SOLUTION**

Let,

$$G_1 = (V_N', \{a, b, c\}, P_1, S)$$

Where $P_1$ and $V_N'$ are constructed as follows,

(i) $S \rightarrow a \mid b$ are added to $P_1$.

(ii) $S \rightarrow cSS$ yield $S \rightarrow C_cSS$ and $C_c \rightarrow c$.

$$V_N' = \{S, C_c\}$$

$P_1$ consists of,

$$S \rightarrow C_cSS$$

$$C_c \rightarrow c$$

$$S \rightarrow a \mid b$$

$S \rightarrow C_cSS$ is replaced by,

$$S \rightarrow C_cC_1$$

$$C_1 \rightarrow SS$$

The remaining productions in $P_1$ are added to $P_2$.

Let,

$$G_2 = (\{S, C_c, C_1\}, \{a, b, c\}, P_2, S)$$

Where, $P_2$ consists of,

$$S \rightarrow C_cC_1$$

$$C_1 \rightarrow SS$$

$$C_c \rightarrow c$$

$$S \rightarrow a \mid b$$

$G_2$ is in CNF and is equivalent to G.

oufastupdates.com

# PUMPING LEMMA FOR CONTEXT FREE LANGUAGES

The pumping Lemma for context free languages state that every sufficiently long string in a regular set contains a short substring that can be pumped many times as we like and the result is a string in the same regular set. The pumping lemma for CFL states for every sufficiently long string in a context free language, there are always two short substrings close together that can be pumped same number of times and the result is a string in same context-free language.

## Statement of the Pumping Lemma

The pumping lemma for CFL's is similar to the pumping lemma for regular languages, but we break each string Z in the CFL into the firve parts, and we pump the second and fourth in tandem.

**THEOREM**

**Let L be any CFL, then there exists an interger m such that, for any string w ∈ L with |W| ≥ m, the string w can be split into five parts as w = uvxyz such that,**

**(i) |vy| ≥ 1.**

**(ii) |vxy| ≤ m.**

**(iii) For all i ≥ 0 $uv^ixy^iz \in L$.**

**PROOF**

Consider an infinite context-free grammar G with no unit productions and no ∈-productions. Consider a string w ∈ L(G) with length (m),

Where,

m > (number of productions) × (Largest right-side of productions).

This suggests that some variable must be repeated in the derivation of w.

Consider u, v, x, y, z : Strings of terminals.

With

$S \rightarrow uAz$

$A \rightarrow vAy$

$A \rightarrow x$

$w \rightarrow uvxyz.$

oufastupdates.com

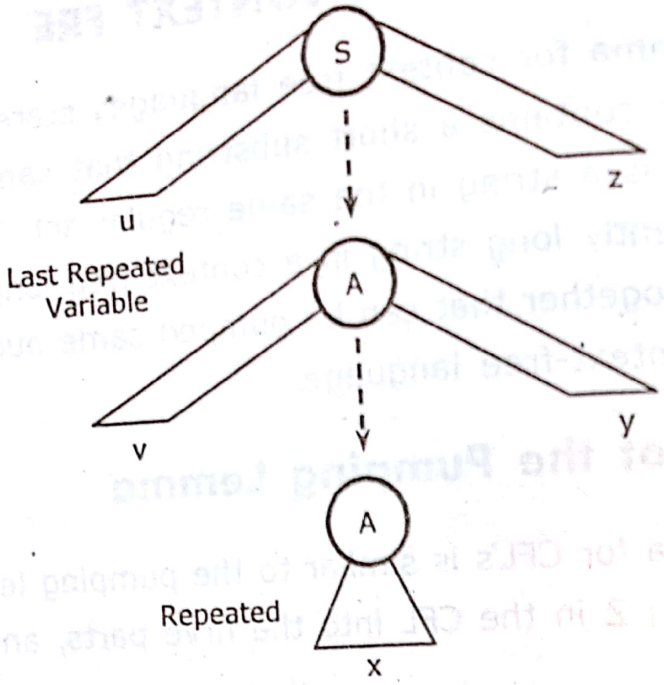



**Fig. 7.2.1** The Tree of w

The possible derivations of w are represented below,

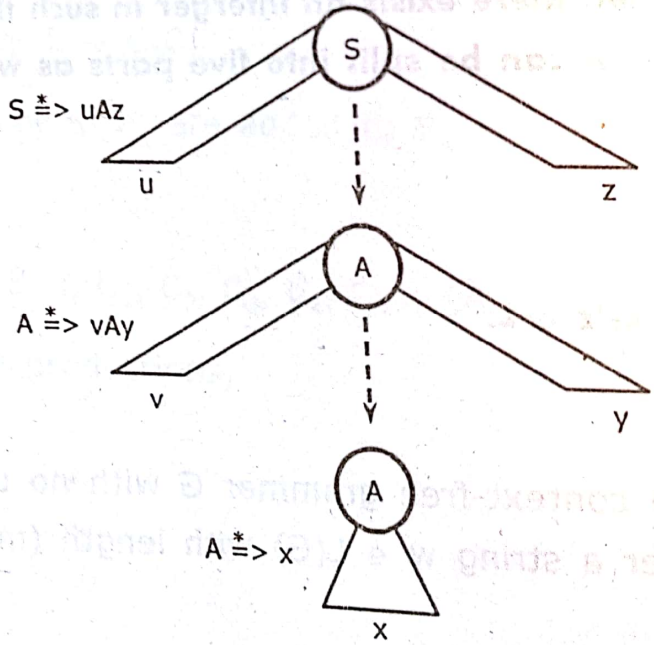



**Fig. 7.2.2** Derivations of w

Following are the strings that can be generated

(i) We know that $S \overset{*}{\Rightarrow} vAz$, $A \overset{*}{\Rightarrow} vAy$ and $A \overset{*}{\Rightarrow} x$

$$S \overset{*}{\Rightarrow} vAz, \overset{*}{\Rightarrow} uxz$$

$$\therefore \quad w = uv^{o}xy^{o}z$$

(ii) We know that $S \overset{*}{\Rightarrow} uAz$, $A \overset{*}{\Rightarrow} vAy$ and $A \overset{*}{\Rightarrow} x$.

$$S \overset{*}{\Rightarrow} uAz, \overset{*}{\Rightarrow} uvAyz \overset{*}{\Rightarrow} uvxyz$$

$$\therefore \quad w = uv^{1}xy^{1}z$$

oufastupdates.com

(iii) We know that $S \overset{*}{\Rightarrow} uAz$, $A \overset{*}{\Rightarrow} vAy$ and $A \overset{*}{\Rightarrow} x$.

$$S \overset{*}{\Rightarrow} uAz, \overset{*}{\Rightarrow} uvAyz \overset{*}{\Rightarrow} uvvAyyz \overset{*}{\Rightarrow} uvvxyyz$$

$$\therefore \quad w = uv^2xy^2z.$$

(iv) We know that $S \overset{*}{\Rightarrow} uAz$, $A \overset{*}{\Rightarrow} vAy$ and $A \overset{*}{\Rightarrow} x$.

$$S \overset{*}{\Rightarrow} uAz, \overset{*}{\Rightarrow} uvAyz \overset{*}{\Rightarrow} uvvAyyyz \overset{*}{\Rightarrow} uvvvAyyyz \overset{*}{\Rightarrow} uvvvxyyyz$$

$$\therefore \quad w = uv^3xy^3z.$$

In general, for productions $S \overset{*}{\Rightarrow} uAz$, $A \overset{*}{\Rightarrow} vAy$ and $A \overset{*}{\Rightarrow} x$, The string generated is,

$$S \overset{*}{\Rightarrow} uAz, \overset{*}{\Rightarrow} uvAyz \overset{*}{\Rightarrow} uvvAyyz \overset{*}{\Rightarrow}$$

$$\overset{*}{\Rightarrow} uvvvAyyyz, \overset{*}{\Rightarrow} \ldots$$

$$\overset{*}{\Rightarrow} uvvv \ldots vAy \ldots yyyz \overset{*}{\Rightarrow}$$

$$\overset{*}{\Rightarrow} uvvv \ldots vxy \ldots yyyz$$

$$\overset{*}{\Rightarrow} uv^ixy^iz.$$

Therefore, a string of the form $uv^ixy^iz$ for $i \geq 0$ is generated by G. Further, $uvxyz \in L(G)$ implies that, $uv^ixy^iz \in L(G)$ with $|vxy| \leq m$ and $|vy| \geq 1$.

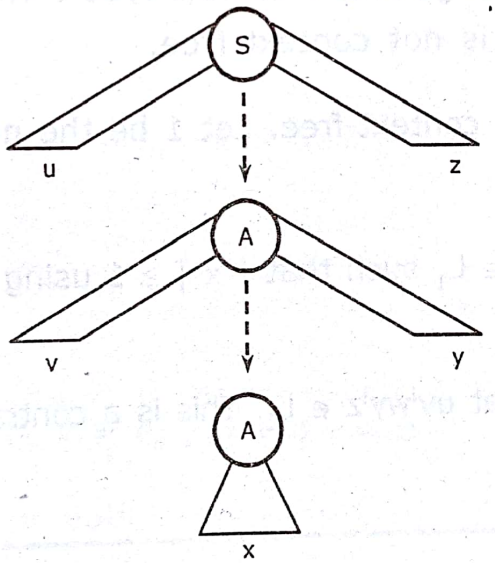
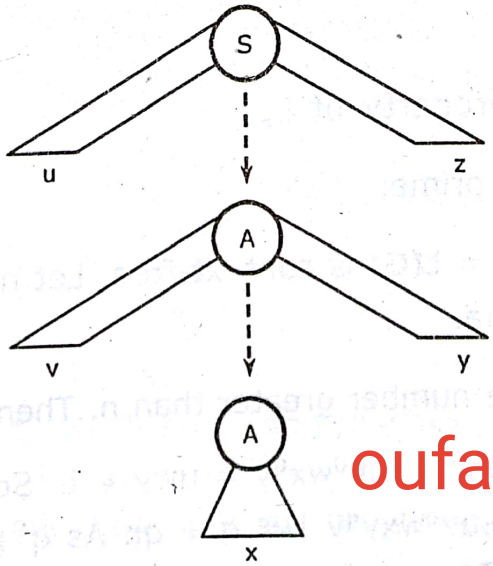



**Fig. 7.2.3** $|vxy| \leq m$, Since A is the Last Repeated Variable




oufastupdates.com

**Fig. 7.2.4** $|vy| \geq 1$ Since There are No Unit and $\in$ Productions

## OGDEN's Lemma

Ogden's Lemma is a stronger foem of pumping lemma for CFL's. Let L be a CFL, then there is a constant 1 (which may infact be the same as for the Pumping Lemma) such that if x is any word in L and we mark any more positions of x 'distinguished' then we write x = uvwyz, such that,

(1) v and y together have atleast one distinguished position.

(2) vwz has atmost n distinguished positions.

(3) For all $i \geq 0$, $uv^iwy^iz$ is in L.

## Applications of Pumping Lemma

Pumping Lemma is useful to prove certain languages are not CFLs.

We assume initially given language (L) is context-free. By applying Pumping Lemma we get a contradiction.

**Procedure for Proving Language is not Context-Free :** The following steps are considered to show a given language is not context-free,

**STEP 1 :** Suppose that $L_1$ is context-free. Let 1 be the natural number obtained by using Pumping Lemma.

**STEP 2 :** Choose a string $x \in L_1$ such that $|x| \geq 1$ using Pumping Lemma principle, write x = uvwyz.

**STEP 3 :** Find suitable i so that $uv^iwy^iz \notin L_1$. This is a contradiction. So $L_1$ is not context-free.

### EXAMPLE PROBLEM 7.11

**Show that $L = \{a^p \mid p \text{ is a prime}\}$ is not a CFL.**

**SOLUTION**

We use the following property of L,

If $w \in L$, then $|w|$ is prime.

**STEP 1 :** Suppose that L = L(G) is context-free. Let n be the natural number obtained by using pumping lemma.

**STEP 2 :** Let p be a prime number greater than n. Then $Z = a^p \in L$. We write z = uvwxy.

**STEP 3 :** By pumping Lemma, $uv^0wx^0y = uwy \in L$. So $|uwy|$ is a prime number say q. Let $|vx| = r$. Then $|uv^qwxy^qy| = q + qr$. As q + qr is not prime, $uv^qwxy^qy \notin L$. This is a contradiction. Therefore, L is not context-free.

oufastupdates.com

**Show that $L = \{a^i b^i c^i \mid i \geq 1\}$ is not context-free.**

**SOLUTION**

Suppose L is context-free and let n be the constant of Lemma.

Consider $z = a^n b^n c^n$.

We write $z = uvwxy$, such that,

(i) $|vx| \geq 1$.

(ii) $|uWx| \leq n$.

(iii) $uv^iWx^iy \in L$ for all $i \geq 0$.

There are two cases,

**CASE (1)**

Let,

$$W = aabbcc$$

By pumping lemma,

$$W = uv^iWx^iy.$$

Where, $u = aa$, $v = b$, $w = b$, $x = c$, $y = c$ if $i = 0$

Then we get, $W = uv^owx^oy = aabc$

$= a^2bc \notin L$ which is a condradiction.

**CASE (2)**

If $i = 2$ then W becomes,

$$W = uv^2Wx^2y$$

$$= aabbbccc$$

$= a^2b^3c^3 \notin L$ which is a contradiction.

Since, in both Case (1) and Case (2) we get contradiction to our assumption, the given language L is not a context free language.

oufastupdates.com

**Show that the language L = {www : w ∈ {a, b}*} is not context-free.**

**SOLUTION**

Consider the string $a^m b^m a^m b^m$.

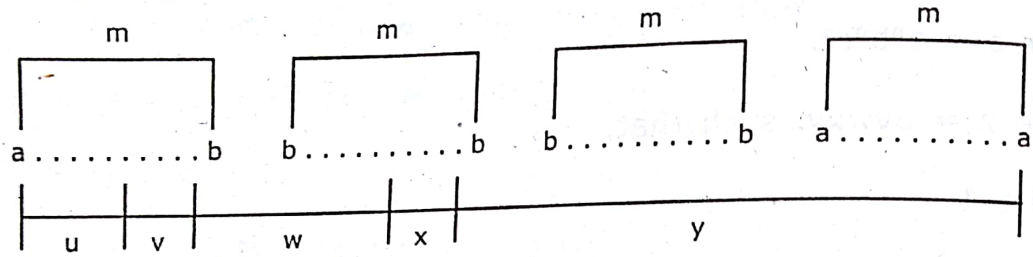



**Fig. 7.2.5** Yielding of String $a^m b^m a^m b^m$

There are many ways in which we can pick vwx, but for all of them we have a winning counter move. For example, for the choice in the Fig. 7.2.5 we can use i = 0 to get a string of them from,

$$a^k b^j a^m b^m,\ k < m \text{ (or) } j < m$$

Which is not in L. For other choices also, similar arguments can be made.

We conclude that L is not context-free.

## Solved Problems

### SOLVED PROBLEM 1

**Prove that the following language is not context free language L = {$a^n b^n c^j$/n ≤ j ≤ 2n}.**

**SOLUTION**

Given that,

$$L = \{a^n b^n c^j / n \le j \le 2n\}$$

Assume 'm' to be a constant. Let us select a long string 'S' such that $S = a^m b^m c^{2m}$ and ensure that S∈L and |S| ≥ m. If S = uvwxyz then the possible values of vx that satisfy |vx| ≤ m and 1 ≤ |vx| ≤ m are,

**(i)** **vx = $a^t$ :** When vx = $a^t$, where 1 ≤ t ≤ m, it means that vx has only a's. Then, the value of $uv^2wx^2y$ is $a^{m+t}\ b^m\ c^{2m}$. In $uv^2wx^2y$, the number of a's ranges from m+1 (putting t = 1) to 2m (putting t = m). Since number of b's is m, which is not equal to the number of a's, $uv^2wx^2y \in L$ is a contradiction to pumping lemma.

**(ii)** **vx = $b^t$ :** When vx = $b^t$, where 1 ≤ t ≤ m, it means that vx has only b's. Then, the value of $uv^2wx^2y$ is $a^m\ b^{m+t}\ c^{2m}$. In $uv^2wx^2y$, the number of b's ranges from m+1 (putting t = 1) to 2m the number of b's, $uv^2wx^2y \notin L$ is a contradiction to pumping lemma.

oufastupdates.com

**vx = $c^t$ :** When vx = $c^t$, $1 \le t \le m$, it means that vx has only c's. Then, the value of $uv^2wx^2y$ is $a^m b^m c^{2m+t}$. In $uv^2wx^2y$, number of c's ranges from 2m + 1 to 3m, which does not satisfy the condition $m \le j \le 2m$. Thus, $uv^2wx^2y \notin L$, which is a contradiction to pumping lemma.

(iv) **vx = $a^t b^p$ :** When vx = $a^t b^p$, where $1 \le t, p \le m$, it means that vx has both a's and b's. Then, the value of $uv^2wx^2y$ is $a^{m+1} b^{m+p}$. In $uv^2wx^2y$, number of a's is less than or equal to number of b's, but not always equal. Thus, $uv^2wx^2y \notin L$, which is a contradiction to pumping lemma.

(v) **vx = $b^t c^p$ :** When vx = $b^t c^p$, where $1 \le t, p \le m$, it means that vx has both b's and c's. Then, the value of $uv^2wx^2y$ is $b^{m+t} C^{2m+P}$. In $uv^2wx^2y$, number of b's is more than number of a's. Thus, $uv^2wx^2y \notin L$, which is a contradiction to pumping lemma.

From the evaluations done above, there is no valve of vx such that $uv^hwx^hy$, where $h \ge 0$, is in L. Thus, the language $L = \{a^n b^n c^j \mid n \le j \le 2n\}$ is not a CFL.

## SOLVED PROBLEM 2

**Show that $L = \{a^i b^j / j = j^2\}$ is not context free language.**

**SOLUTION**

Give that, $L = \{a^i b^j / j = i^2\}$

Assume to be a constant. Let us select a long string 'S', $S = a^m b^{m^2}$ and ensure that $S \in L$ and $|S| \ge m$ (Length of S is greater than or equal to m). If S = uvwxyz then the possible values of vx that satisfy $|vwx| \le m$ and $1 \le |vx|\ m \le$ are,

(i) **vx = $a^t$ :** When vx = $a^t$, where $1 \le t \le m$, it means that vx has only a's. Then, the value of $uv^2wx^2y$ is $a^{m+1} b^{m^2}$. In $uv^2wx^2y$, the number of a's ranges from m + 1 (putting t = 1) to t + m (putting t = m). Since number of b's is $m^2$, which is not equal to the number of a's. Hence $uv^2wx^2y \in L$ and it is a contradiction to pumping lemma.

(ii) **vx = $b^t$ :** When Vx = $b^t$, where $t \le t \le m$, it means that vx has only b's. Then the value of $uv^2wx^2y$ is $a^m b^{m^2+t}$. In $uv^2wx^2y$, the number of b's ranges from $m^2$ + 1 (putting t = 1) to $m^2$ + m (putting t = m), which is not the square of number of a's. Hence $uv^2wx^2y \notin L$ and it is a contradiction to pumping lemma.

(iii) **vx = $a^t b^r$ :** When vx = $a^t b^r$, where $1 \le t$ and $r \le m$, it means that vx has both a's and b's. Then, the value of $uv^2wx^2y$ is $a^{m+t} b^{m^2+r}$. In $uv^2wx^2y$, the minimum number of a's is m + 1 (putting t = 1) and the minimum number of b's is $m^2$ + m (putting r = m). The number of a's and b's are not equal. Hence $uv^2wx^2y \notin L$ and it is a contradiction to pumping lemma.

oufastupdates.com

## CLOSURE PROPERTIES

Like regular language, the context-free languages are closed under the operators such as union, concatenation and star-closure but there are exceptions also like complement and intersections.

Thus, closure properties that hold for regular language need not always hold for context free languages.

After performing that particular operation on those CFLs the resultant language is context free language.

### Substitution

Let $\Sigma$ be an alphabet, and suppose that for every symbol a in $\Sigma$, we can choose language $L_a$ consider another alphabet $\Delta$ can be over an alphabet $\Delta$ i.e, substitution from $\Sigma$ to $\Delta$ is a mapping which maps each symbol a to the language $L_a$ over $\Delta$.

If $w = a_1, a_2 \ldots\ldots a_n$ is a string in $\Sigma^*$ than s (w) is the language of all strings $x1$ $x_2 \ldots\ldots x_n$ such that string $x_i$ is in the language S $(a_i)$ per i = 1, 2, ..... n. In another way S(w) is the concatenation the language S $(a_1)$ S $(a_2)$ ..... S $(a_n)$,

i.e., s (w) = S $(a_1)$ S $(a_2)$ S $(a_3)$ ..... S $(a_n)$

If L is a a language over $\Sigma$, S (L) = {S (w) | w $\in$ L}.

oufastupdates.com

## DECISION PROPERTIES OF CFL'S

There are algorithms to answer whether the given CFL is empty or non-empty, finite infinite. Even they can answer whether the given word x in L or not.

Certain question like,

) Whether two CFG's $G_1$ and $G_2$ are equivalent or not.

) Whether the complement of a finite CFL is finite or not.

) Whether the given CFG G is ambiguous or not cannot be answered by the existing algorithms.

### XAMPLE PROBLEM 7.22

**Test the following grammar is finite or infinite,**

**S → AB**

**A → BC | a**

**B → CC | b**

**C → a.**

**SOLUTION**

Drawing a directed graph from the given productions.

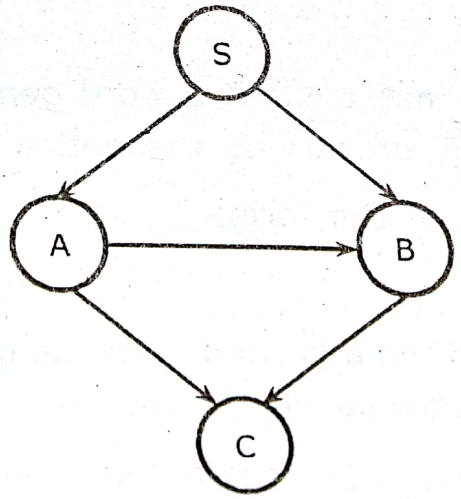



**Fig. 7.4.1** Dependency Graph for the given Grammar

In this graph there are no cycles.

The given grammar is finite.

oufastupdates.com

**Let us consider the following CFG in CNF,**

$$S \rightarrow AA$$

$$A \rightarrow AA$$

$$A \rightarrow a$$

**SOLUTION**

The list starts off easily enough,

| Substring | All producing non terminals |
|---|---|
| $a_1 = a$ | A |
| $a_2 = a$ | A |
| $a_3 = a$ | A |

We can see now that both substrings of length 2 are the same, aa, and are factorable into exactly AA. This is the right side of two productions whose left sides are S and A therefore the list consists.

| Substring | All producing non terminals |
|---|---|
| $a_1a_2$ | S, A |
| $a_2a_3$ | S, A |

There is only one length 3 substring, x itself and it can be factored into $a_1$ times $a_2a_3$ or $a_1a_2$ times $a_3$. The first case gives the non terminal possibilities. AS or AA, and the second gives the possibilities SA or AA. Of these, only AA is the right side of a production. The left sides are S and A.

Therefore, the list concludes with the following,

| Substring | All producing non terminals |
|---|---|
| $w = a_1a_2a_3$ | S, A |

From the list, we see that the word w can be derived from the start symbol S and so it is in the language. It should also be clear that similarly any string of more than three a's can also be produced by this CFG from non terminal S and A.

oufastupdates.com

# *Long Questions ( UNIT 4)*

-> Turing Machine

-> Programming techniques of T. M

-> ID's of T.M

-> Universal T.M

oufastupdates.com

In the previous two model of computation we have seen that neither finite automata nor pushdown automata can be regarded as general models for computers. This is because they are not capable of recognizing even simple languages such as $\{a^n b^n c^n \mid n \geq 0\}$ and $\{ww \mid w \in \Sigma^*\}$. This leads to look beyond regular and context free languages. As a result the study of devices that can recognize this and more complicated languages came into picture. Such devices are called *Turing machines*. It was named after the scientist Alan Turing in 1936.

Historically, the order of invention of these ideas is as follows,

(1) Finite automata and regular languages were developed by Kleene, Moore, Mealy and Scott in the 1950's.

(2) Pushdown automata and context free languages were developed later, by Chomsky, Greibach, Oettinger, Every and Schutzenberger in the 1960's.

(3) Turing machines and their theory were developed by Alan Turing and Emil Post in the 1930's.

The *Turing machines* is a simple mathematical model of a general computer. It is an extremely basic abstract symbol manipulating device which, despite their simplicity, can be adapted to simulate the logic of any computer that could possibly be constructed. Though they were intended to be technically feasible, Turing machines were not meant to be a practical computing technology, but a thought experiment about the limits of mechanical computation, thus they were not actually constructed. Studying their abstract properties yields many insights into computer science and complexity theory.

The languages accepted by Turing machine are said to be recursively enumerable. It is precisely those languages whose strings can be listed by a Turing machine. The Turing machine is studied both for the class of languages it defines and the class of integer functions it computes called the *partial recursive functions*. In particular as today, the Turing machine is equivalent to the computational power of a digital computer.

## PROBLEMS THAT COMPUTERS CANNOT SOLVE

A Turing Machine (TM) is similar to a finite automata with an unlimited and unrestricted memory i.e., modified versions of PDA and it is more powerful than PDA. A Turing machine is however a more accurate model of a general purpose computer. A Turing machine can do everything that a real computer can do.

**NOTE :** ***Even a turing machine cannot solve certain classes of problems.***

**Real Computer :** A real computer is a triple,

Computer = (Processor, Memory, I/O devices) which performs the actions.

oufastupdates.com

**Run Program**

While (plugged in and power on).

execute (pc).

pc = next (pc).

**nplications**

1) Processor = (pc, instructions).

2) Program is stored in memory as a stream of instructions.

3) PC always points to the next instruction to execute.

**ıformate Characterizations**

1) **TM Memory :** Infinite tape.

(2) **TM I/O :** A tape-head that can read/write symbols and move around on the tape.

(3) **TM Processor :** A control device that performs transformations of the symbols written on the tape while moving the head around.

**Initial Condition**

(1) Initially the tape contains only the input string and is blank everywhere else.

(2) If TM needs to store information, it may write it on the tape.

(3) To read the information that it has written, TM can move its head back over the tape.

(4) Machine continues computing until it decides to produce an output.

**The Output :** The computation performed by a TM ends up with an output, which is one of,

(1) Accept (2) Reject (3) Compute forever

## THE TURING MACHINE

turing machine M is defined as a seven tuple $(S, \Sigma, \Gamma, \delta, s_0, B, A)$, consisting of,

S = Non empty finite set of internal states.

$\Gamma$ = Non empty finite set of tape symbols.

$\Sigma$ = Non empty finite set of input symbols, which is a subset of $\Gamma$ but not including B, i.e., $\Sigma \subseteq \Gamma - \{B\}$.

B = Symbol of $\Gamma$, known as blank.

$\delta$ = Transition function from $S \times \Gamma$ to $S \times \Gamma \times \{L, R\}$, i.e., $S \times \Gamma \rightarrow S \times \Gamma \times \{L, R\}$, however $\delta$ may be undefined for some arguments.

$s_0$ = Initial or start state, i.e., $s_0 \in S$.

A = Set of final or accepting states, i.e., $A \subseteq S$.

oufastupdates.com

The transition function $\delta : S \times \Gamma \rightarrow S \times \Gamma \times \{L, R\}$ states that, if the Turing machine is in some state (from set S), by taking a tap symbol (from set $\Gamma$), moves to some next state by replacing the current symbol by another or same symbol and moves the read write head one cell either left (L) or right (R) along the tape. The new state need not be different from the current state.

**Example :** If $\delta(s, x) = (s', y, R)$, then the machine gets into the next state s', writes the symbol 'y' on the tape by replacing the symbol 'x' and moves one cell right. Similarly, the transition $\delta(s, 0) = (s', B, L)$ describes that the Turing machine changes its state from s to s' on replacing the current input symbol '0' by blank 'B'. The read write head moves one cell left.

## Turing Machine Model

The basic Turing machine model can be visualized as an indefinitely long tape coupled to a *finite control* unit. The *tape*, which acts as the machine's memory is divided into cells or squares. Each square may be inscribed with one symbol from a designated finite alphabet or it may be blank. The different tape symbols will usually be represented by numerals or capital letters, with the restriction that the letter B will be reserved to denote a blank cell. A *read-write* head is associated with this tape that can travel right or left on the tape and that can read and write a single symbol on each move.

The *control unit* of a Turing machine is capable of assuming any one of a fixed, finite number of states. At a given time, the state of the control unit together with the tape symbol scanned by the read-write head, uniquely determines how the machine will behave at that time. The actions available to a Turing machine are quite limited, either it may halt, thereby terminating its operation, or a basic move may be carried out. Each move consists in writing a symbol in the currently scanned cell, shifting the tape one cell to the back or forth, and causing the control unit to enter a new state. The symbol that the Turing machine writes on its tape need not be different then the symbol that is already there and the new state need not be different from the current state. A block diagram of the Turing machine model is given in Fig. 8.3.1.

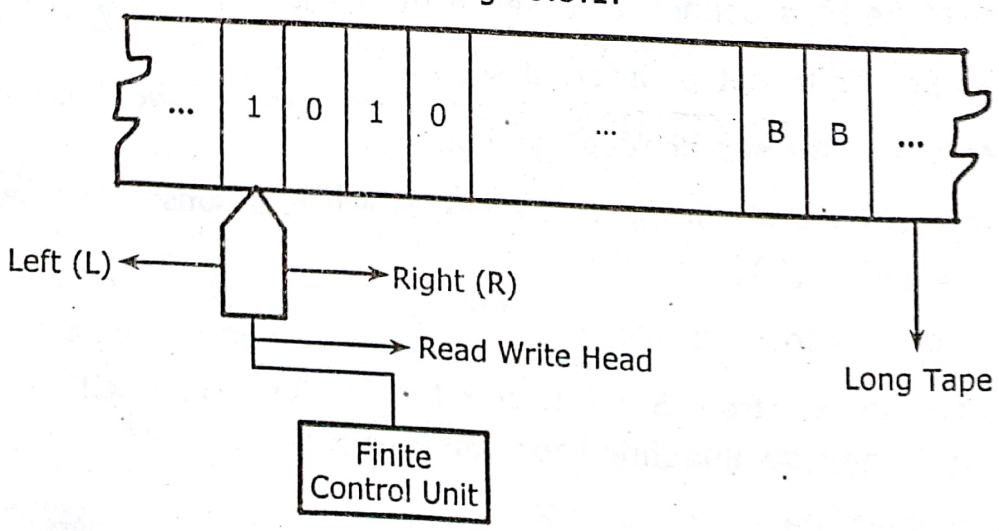



**Fig. 8.3.1** A Basic Turing Machine Model

oufastupdates.com

## Transition Table

If we define a transition system δ is form of table, then that representation is called a *transition table*.

If $\delta(q, a) = (z, x, y)$, we write xyz under column and q row. So if we get xyz in the table, it means that x is written in the current cell, y gives the movement of the head (L or R) and z denotes the new state into the turing machine enters.

**Example :** Let us consider a TM with five stages $q_0, q_1, q_2, q_3, q_4$ where $q_0$ is an initial as well as final state. The tape symbols are a, b and #. Then the transition table δ is defined as follows,

**Table 8.3.1** Transition Table

| Present State | Tape Symbols | | |
|---|---|---|---|
| | a | B | # |
| → $q_0$ | $bRq_1$ | $aRq_2$ | $\#Rq_0$ |
| $q_1$ | $bRq_0$ | $aRq_3$ | $\#Rq_4$ |
| $q_2$ | $bRq_3$ | $aRq_0$ | $\#Rq_4$ |
| $q_3$ | $bRq_2$ | $aRq_1$ | $\#Rq_4$ |
| $q_4$ | – | – | $\#Rq_4$ |

This TM is converting all even number of either a's or b's or both a's and b's into even number of b's or a's or b's and a's respectively.

## Instantaneous Description (ID) For Turing Machines

An instantaneous description (ID) of a turing machine is defined in terms of the entire nput string and the current state. We denote an instantaneous description of a Turing machine M by αsβ, where 's' is the present state of M is in S, α and β are substrings of string 'αβ'. The substring 'α' is the processed string whereas the substring 'β' is the remaining string to be processed. The substring 'α' is also known as *left sequence* whereas the substring 'β' is known as *right sequence*. Observe that the blank 'B' may occur in 'αβ'.

**Example :** Consider the turing machine is in state $s_3$ with the input string $a_1a_2a_3a_4a_5a_6$. Let the symbol under read-write head be $a_4$. Therefore, the instantaneous description given as '$a_1a_2a_3a_4a_5a_6$'. Here, we have $\alpha = a_1a_2a_3$ and $\beta = a_4a_5a_6$. A diagrammatic representation is given in Fig. 8.3.2

oufastupdates.com

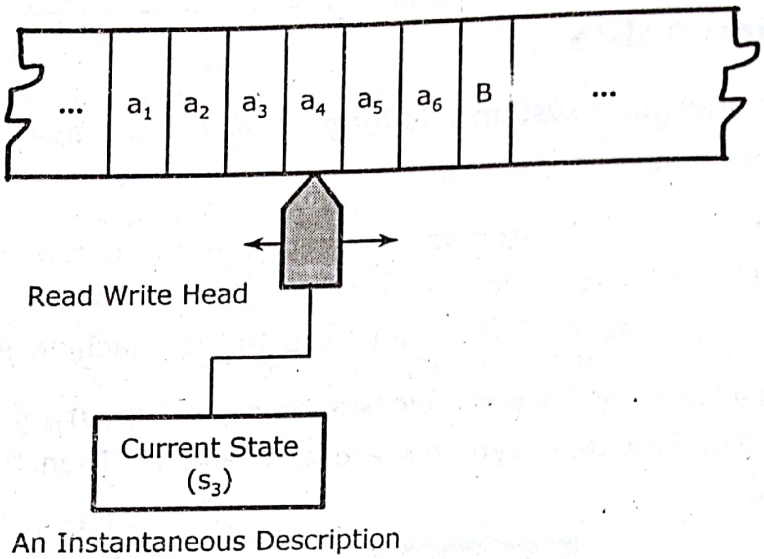



**Fig. 8.3.2** An Instaneous Description of Turing Machine

Turing machine using ***move relation*** is discussed below. By using the string $X_1, X_2, \ldots X_{i-1q}, X_i, X_{i+1} \ldots\ldots\ldots X_n$ to represent an ID in which,

(1) q is the state of the turing machine.

(2) The tape head is scanning the $i^{th}$ symbol from the left.

(3) $X_1, x_2, \ldots\ldots X_n$ is the portion of the tape between the leftmost and the rightmost non-blank.

In this subsection we define the transitions in a turing machine by using move relation. By considering an instantaneous description $X_1X_2 \ldots X_{i-1}\, SX_i \ldots X_n$. Suppose that $\delta(s, X_i) = (q, Y, L)$.

If $(i - 1) = n$, then $X_i$ is taken to be B. If $i = 1$, then the tape head is to fall off the left end of the tape. This is not allowed and there is no next instantaneous description. If $i > 1$, then we have,

$$X_1X_2 \ldots X_{i-1}\, sX_i \ldots X_n \vdash X_1\, X_2 \ldots X_{i-2q}\, X_{i-1}\, Y_q\, X_{i+1} \ldots X_n.$$

In this case, the symbol $X_i$ is replaced by Y and there is a left move. Therefore, the current symbol under read-write head is $X_{i-1}$ and the state s is changed to q.

Alternatively, suppose that $\delta(s, X_i) = (q, Y, R)$. If $i > 1$, then we have,

$$X_1X_2 \ldots X_{i-1}\, s\, X_n \vdash X_1\, X_2 \ldots X_{i-2}\, X_{i-1}\, Y_q\, X_{i+1} \ldots X_n$$

In this case, the symbol $X_i$ is replaced by Y and there is a right move. Therefore, the current symbol under read-write head is $X_{i+1}$ and the state s is changed to q.

oufastupdates.com

This can be represented by a move as shown below,

$$a_1a_2a_3a_4q_2a_5a_6a_7a_8.... \vdash a_1a_2a_3a_4b_1q_3a_6a_7a_8....$$

Similarly if the current ID of the turing machine is,

$a_1a_2a_3a_4q_2a_5a_6a_7a_8....$ and there is a transition

$$\delta(q_2, a_5) = (q_1, c_1, L)$$

means that if the machine is in state $q_2$ and the next symbol to be scanned is $a_5$, then the machine enters into state $q_1$ replacing the symbol $a_5$ by $c_1$ and L indicates that the read-write head is moved one symbol towards left.

The new configuration obtained is,

$$a_1a_2a_3q_1a_4c_1a_6a_7a_8....$$

This can be represented by a move as shown below,

$$a_1a_2a_3a_4q_2a_5a_6a_7a_8.... \vdash a_1a_2a_3q_1a_4c_1a_6a_7a_8....$$

This configuration indicates that the new state is $q_1$, the next input symbol to be scanned is $a_4$. In general, the actions performed by TM depends on,

(1) The current state.

(2) The whole string to be scanned.

(3) The current position of the read-write head.

The action performed by the machine consists of,

(1) Changing the states from one state to another.

(2) Replacing the symbol pointed to by the read-write head.

(3) Movement of the read-write head towards left or right.

## Acceptance of a Language by TM

A string w in the turing machine M is said to be accepted if $q_0w \vdash^* \alpha_1P\alpha_2$ for some $P \in F$ and $\alpha_1$ and $\alpha_2 \in \Gamma^*$

The turing machine does not accept string w if the machine M either halts in a non accepting state (non-final state) or does not halt.

***Strings Not Accepted by Turing Machines :*** When a string is not accepted by a turing machine, that is when a turing machine does not halt on a string, one of the following three things happens,

(1) The turing machine goes into a infinite loop.

(2) No transition is specified for the current configuration.

(3) The head is at the left end and it is instructed to move left

oufastupdates.com

In cases (2) and (3), the operation of the turing machine is aborted. For example the following turing machine accepts the language a+, but it goes into an infinite loop for any strings that are not in the language.

**Acceptors and Transducers :** We have defined acceptors, which produce only a binary (accept/reject) output and transducers, which can produce more complicated results. However, all our work so far has been with acceptors. A turing machine also accepts or rejects its input. More importantly, the results left on the tape when the turing machine finishes can be regarded as the "output" of the computation, thus, a turing machine is a transducer.

**Turing Machine as Transducers :** A turing machine can be used as a transducer. The most obvious way to do this is to treat the entire nonblank portion of the initial tape as input and to treat the entire nonblank portion of the tape when the machine halts as output.

In other words, a turing machine defines a function $y = f(x)$ for strings $x, y \in \Sigma^*$ if,

$$q_0 x \vdash^* q_f y$$

Where $q_f$ is a final state. A function f is turing computable if there exists a turing machine that can perform the above task. If a turing machine halts, the sequence of configurations leading to the half state is called a computation.

**Turing Machines as Acceptors :** A turing machine halts when it no longer has any available moves. If it halts in a final state, it accepts its input, otherwise, it rejects its input.

This is too easy, so let's repeat it in symbols,

A turing machine $T = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ accepts a language L(M).

Where,

$$L(M) = (w \in \Sigma^+ : q_0 w \vdash^* x_i q_f x_j)$$

For some $q_f \in F$, $x_i$, $x_j \in \Gamma^*\}$

We said a turing machine accepts its input if it halts in a final state. There are two ways this could fail to happen,

(1) The turing machine could halt in a non-final state.

(2) The turing machine could never stop (in which case we say it is in an infinite loop).

## EXAMPLE PROBLEM 8.2

**Describe the processing of,**

**(i) 011**

**(ii) 0011**

**(iii) 001 by the turing machine M.**

oufastupdates.com

**Which of the above string are accepted by M?**

**Table 8.3.2** Transition Table

| Present State | Tape Symbol | | | | |
|---|---|---|---|---|---|
| | 0 | 1 | x | y | b |
| $q_1$ | $xRq_2$ | - | - | - | $bRq_5$ |
| $q_2$ | $0Rq_2$ | $yLq_3$ | - | $yRq_2$ | - |
| $q_3$ | $0Lq_4$ | - | $xRq_5$ | $yLq_3$ | - |
| $q_4$ | $0Lq_4$ | - | $xRq_1$ | - | - |
| $q_5$ | - | - | - | $yxRq_5$ | $bRq_6$ |
| $q_6$ | - | - | - | - | - |

**SOLUTION**

For the solution of this problem we make use of both the moves explained above,

(i) $q_1011 \vdash xq_211 \vdash q_3xy1 \vdash xq_5y1 \vdash xyq_51$

As $\delta(q_5, 1)$ is not defined so the input string 011 is not accepted.

(ii) $q_10011 \vdash xq_2011 \vdash x0q_211 \vdash xq_30y1 \vdash q_4x0y1 \vdash xq_10y1 \vdash xxq_2y1 \vdash xxyq_21 \vdash xxq_3yy \vdash xq_3xyy \vdash xxq_5yy \vdash xxyq_5y \vdash xxyyq_5b \vdash xxyybq_6$ M halts.

(iii) $q_1001 \vdash xq_201 \vdash x0q_21 \vdash xq_30y \vdash q_4x0y \vdash xq_10y \vdash xxq_2y \vdash xxyq_2$

M halts and no transition $\delta(q_2, b)$ and $q_2$ is not an accepting state, so 001 is not accepted by M.

## Construction of Turing Machine (TM)

The basic guidelines which are used for construction of turing machine are discussed below,

(1). When the read-write head reads a symbol, we should know about what to do in future. Here, the machine must remember the past symbol. This can be remembered by turing machine by going to the next unique state.

(2) In this, we have to minimize the number of states while designing a turing machine. This can be achieved by changing the state only when there is a change in the written symbol or when there is any change in the movement of the read-write head.

oufastupdates.com

**Obtain a turing machine to accept the language $L = \{0^n1^n \mid n \geq 1\}$.**

**SOLUTION**

It is given that the language accepted by TM should have n number of 0's followed by n number of 1's. For this let us take an example of the string w = 00001111. The string w should be accepted as it has four 0's followed by four 1's.

**General Procedure :** Let $q_0$ be the start state and let the read-write head points to the first symbol of the string to be scanned. The general procedure to design TM for this case is shown below,

(1) Replace the leftmost 0 by X and change the state to $q_1$ and then move the read-write head towards right. This is because, after a 0 is replaced, we have to replace the corresponding 1 so that number of 0's matches with number of 1's.

(2) Search for the leftmost 1 and replace it by the symbol Y and move towards left (so as to obtain the leftmost 0 again). Steps 1 and 2 can be repeated.

**Design :** Consider the situation,

XX00YY11

↑

$q_0$

Where first two 0's are replaced by X's and first two 1's are replaced by Y's. In this situation, the read-write head points to the leftmost 0 and the machine is in state $q_0$. With this as the configuration, now let us design the TM.

**STEP 1 :** In state $q_0$, replace 0 by X, change the state to $q_1$ and move the pointer towards right. The transition for this can be of the form,

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

The resulting configuration is shown below,

XXX0YY11

↑

$q_1$

**STEP 2 :** In state $q_1$, we have to obtain the leftmost 1 and replace it by Y. So, let us move the pointer to point to leftmost 1. When the pointer is moved towards 1, the symbols encountered may be 0 and Y. Irrespective what symbol is encountered, replace 0 by 0, Y by Y, remain in state $q_1$ and move the pointer towards right.

oufastupdates.com

The transitions for this can be of the form,

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

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

When these transitions are repeatedly applied, the following configuration is obtained,

XXX0YY11

$\uparrow$

$q_1$

**STEP 3 :** In state $q_1$, if the input symbol to be scanned is a 1, then replace 1 by Y, change the state to $q_2$ and move the pointer towards left.

The transition for this can be of the form,

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

The resulting configuration is shown below,

XXX0YYY1

$\uparrow$

$q_2$

Note that the pointer should be moved towards left. This is because, 0 is replaced by X and the corresponding 1 is replaced by Y. Now, we have to scan for the leftmost 0 and so, the pointer should be moved towards left.

**STEP 4 :** Note that to obtain leftmost 0, we need to obtain rightmost X first. So, we scan for the rightmost X. During this process we may encounter Y's and 0's. Replace Y by Y, 0 by 0, remain in state $q_2$ only and move the pointer towards left. The transitions for this can be of the form,

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

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

The following configuration is obtained.

XXX0YYY1

$\uparrow$

$q_2$

**STEP 5 :** Now, we have obtained the rightmost X. To get leftmost 0, replace X by X, change the state to $q_0$ and move the pointer towards right.

oufastupdates.com

The transition for this can be of the form,

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

And the following configuration is obtained,

XXX0YYY1

↑

$q_0$

Now, repeating the steps 1 through 5, we get the configuration shown below,

XXXXYYYY

↑

$q_0$

**STEP 6 :** In state $q_0$, if the scanned symbol is Y, it means that there are no more 0's. If there are no 0's we should see that there are no 1's. For this we change the state to $q_3$, replace Y by Y and move the pointer towards right.

The transition for this can be of the form,

$$\delta(q_0, Y) = (q_3, Y, R)$$

And the following configuration is obtained,

XXXXYYYY

↑

$q_3$

In state $q_3$, we should see that there are only Y's and no more 1's. So, as we scan replace Y by Y and remain in $q_3$ only. The transition for this can be of the form,

$$\delta(q_3, Y) = (q_3, Y, R)$$

Repeatedly applying this transition, the following configuration is obtained,

XXXXYYYYB

↑

$q_3$

Note that the string ends with infinite number of blanks and so, in state $q_3$ if we encounter the symbol B, means that end of string is encountered and there exists n number of 0's ending with n number of 1's. So, in state $q_3$, on input symbol B, change the state to $q_4$, replace B by B and move the pointer towards right and the string is accepted.

oufastupdates.com

The transition for this can be of the form,

$$\delta(q_3, B) = (q_4, B, R)$$

Where $q_4$ is the final state and the following configuration is obtained,

XXXXYYYYBB

$\uparrow$

$q_4$

So, the Turing machine to accept the language $L = \{0^n1^n \mid n \geq 1\}$ is given by,

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

Where,

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

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

$\Gamma = \{0, 1, X, Y, B\}$

$q_0 \in Q$ is the start state of machine.

$B \in \Gamma$ is the blank symbol.

$F = \{q_4\}$ is the final state.

$\delta$ is shown below,

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

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

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

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

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

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

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

$\delta(q_0, Y) = (q_3, Y, R)$

$\delta(q_3, Y) = (q_3, Y, R)$

$\delta(q_3, B) = (q_4, B, R)$

oufastupdates.com

**Table 8.3.3** Transition Table of TM to Accept the Language $L = \{0^n, 1^n \mid n \geq 1\}$

| $\delta$ | Tape symbols ($\Gamma$) | | | | |
|---|---|---|---|---|---|
| **States** | **0** | **1** | **X** | **Y** | **B** |
| $\rightarrow q_0$ | $(q_1, X, R)$ | - | - | $(q_3, Y, R)$ | - |
| $q_1$ | $(q_1, 0, R)$ | $(q_2, Y, L)$ | - | $(q_1, Y, R)$ | - |
| $q_2$ | $(q_2, 0, L)$ | - | $(q_0, X, R)$ | $(q_2, Y, L)$ | - |
| $q_3$ | - | - | - | $(q_3, Y, R)$ | $(q_4, B, R)$ |
| $*q_4$ | - | - | - | - | - |

**Transition Diagram**

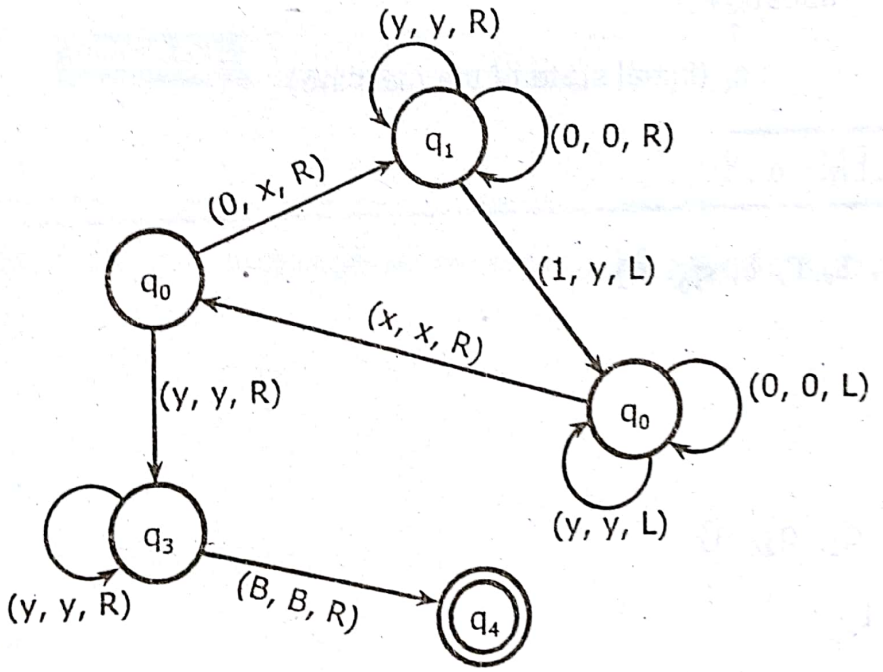



**Fig. 8.3.4** Transition Diagram For $L = \{0^n1^n \mid n \geq 1\}$

## Transition Diagram for Turing Machine (TM)

In some situations, graphical representation of the next move partial function $\delta$ of Turing machine may give better idea of the behaviour of TM in comparison to the tabular representation of $\delta$.

A transition diagram of the next move functions $\delta$ of a TM is a graphical representation consisting of a finite number of nodes and directed labelled arcs between the nodes. Each node represents a state of the TM and a label on an arc from one state (say $q_1$) to a state ($q_2$) represents the information about the required input symbol say 'a' for the transition from $q_1$ to $q_2$ to take place and the action on the part of the control of TM.

oufastupdates.com

The action part consists of,

(1) The symbol say 'b' to be written in the current cell.

(2) The movement of the tape head.

The label of an arc is generally written as a/(b, m), where m is L, R or N.

During the designing procedure of turing machine Tm, any string/input is assumed to be written on input tape in following form.

#abbabb#

That is head is on extreme right blank in the starting and when machine starts processing the string of $q_0$ (starting state) then it is supposed to read first input symbol of the string from right to left, as follows,

#abbabb#
↑
$q_0$ (initial state of the machine)

## EXAMPLE PROBLEM 8.4

**Let TM = {Q, Σ, Γ, δ, $q_0$, h}**

**SOLUTION**

Where,

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

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

$\Gamma = \{0, 1, b\}$

And δ be given by the following Table.

| | 0 | 1 | b |
|---|---|---|---|
| → $q_0$ | - | - | $(q_2, b, R)$ |
| $q_1$ | $(q_2, 0, R)$ | $(q_1, b, R)$ | $(h, b, N)$ |
| $q_2$ | $(q_2, 0, L)$ | $(q_1, 1, R)$ | $(h, b, N)$ |
| *h | - | - | - |

oufastupdates.com

The transition diagram for above TM will be as shown below, where we assume that '$q_0$' is the initial state and 'h' is a final state/half state.

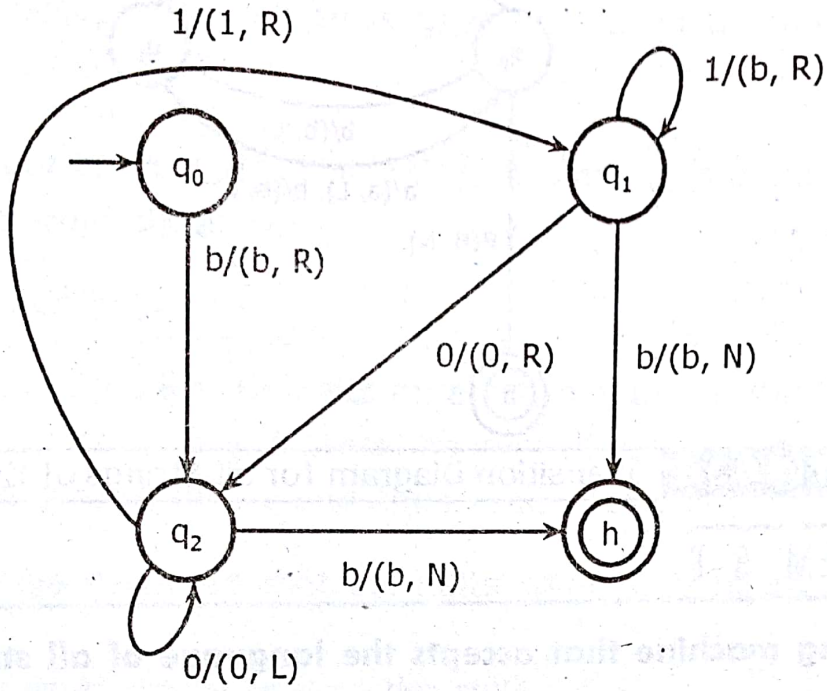



Fig. 8.3.5 Transition Diagram for TM

## AMPLE PROBLEM 8.5

**Design a TM that recognize the language of all strings of even length over alphabet {a, b}.**

**OLUTION**

Let turing machine is,

$$TM = (Q, \Sigma, \Gamma, \delta, q_0, h)$$

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

$$\Gamma = \{a, b, B\}$$

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

Here h is half state when machine halts after accepting the language $q_0$ is initial state. Here next move partial function $\delta$ is given as follows.

| | a | b | B |
|---|---|---|---|
| $\rightarrow q_0$ | $(q_1, a, L)$ | $(q_1, b, L)$ | $(h, B, N)$ |
| $q_1$ | $(q_0, a, L)$ | $(q_0, b, L)$ | - |
| *h | - | - | Accept |

oufastupdates.com

Here '*' is for undefined move. We can construct the transition diagram as follows,

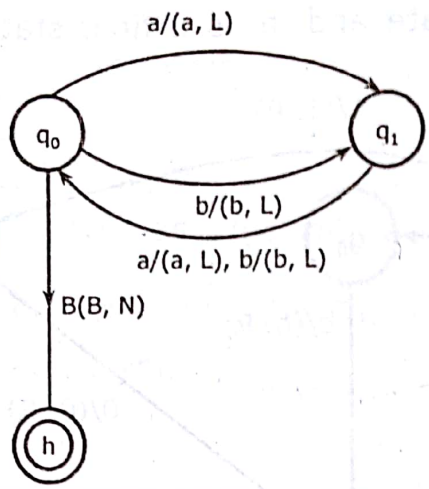



**Fig. 8.3.6** Transition Diagram for all Strings of Even Length

## EXAMPLE PROBLEM 8.6

**Design a turing machine that accepts the language of all strings which contain aba as a substring.**

**SOLUTION**

Let turing machine is,

$$T \quad M \quad \Sigma, \Gamma, \delta, q_0, h)$$

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

$$\Gamma = \{a, b, B\}$$

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

And $\delta$ is defined as follows.

| | a | b | B |
|---|---|---|---|
| $\rightarrow q_0$ | $(q_1, a, L)$ | $(q_0, b, L)$ | - |
| $q_1$ | $(q_1, a, L)$ | $(q_2, b, L)$ | - |
| $q_2$ | $(h, a, L)$ | $(q_0, b, L)$ | $(h, B, N)$ |
| *h | - | - | Accept |

Now we can also design transition diagram as follows.

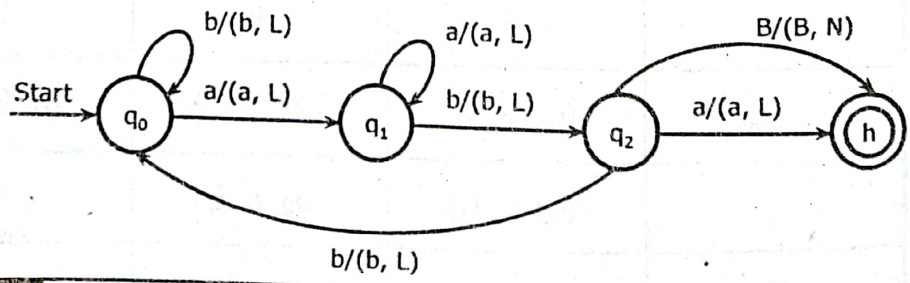



**Fig. 8.3.7** Transition Diagram for all Strings Contains aba as Substring

oufastupdates.com

## Turing Machines and Halting

Another notion of "acceptance" which is used for turing machine is acceptance by *halting*.

Turing machine is said to be *halts* if it enters a state q, scanning a tape symbol X, and there is no move in this situation.

That is $\delta(q_1\ X)$ is undefined.

**Example :** In the above Example 8.6 TM halts on all strings of a's and b's, since no matter what string TM finals on its tape, it will cancel its second group of a's, if it can find such a group, against its first group of a's and thus reach state h and halts.

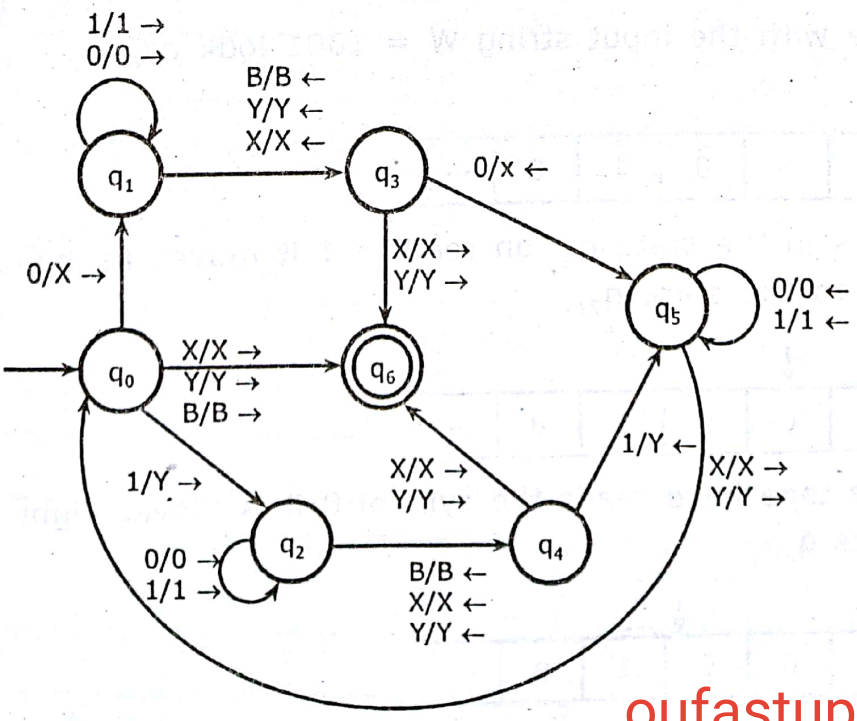
Therefore, it is always assumed that TM halts when it is in an *accepting state*.

**NOTE :** *TM always halts when it is in an accepting state.*

Unfortunately, TM halts even if it does not accept. The languages with turing machines that do halt eventually, regardless of whether or not they accept, are called *recursive*.

### SOLVED PROBLEM 1

**Show the sequence of moves in terms of IDs along with the details of state and tape contents at various instants before and after the moves of the turing machine given below for the input string w = 1001.**




oufastupdates.com

**Design a TM that recognizes the set,**

**$\{0^{2n}\}^n \geq n = 0\}$.**

**SOLUTION**

The given language is not meaningful. The modified language is $L = \{0^{2n}\ 1^n | n \geq 0\}$.

It accepts strings that begins with 0's and ends with 1's where number of 0's are double the number of 1's.

**Example :** $L = \{\epsilon, 001, 000011, ...\}$

The tuple representation of TM is,

$$TM = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, x, y, B\}, \delta, q_0, B, \{q_4\})$$

Where, $\delta$ is defined as,

$$\delta(q_0, 0) = (q_1, x, R)$$

$$\delta(q_1, 0) = (q_2, x, R)$$

$$\delta(q_2, 0) = (q_2, 0, R)$$

$$\delta(q_2, 1) = (q_3, y, R)$$

$$\delta(q_2, y) = (q_2, y, R)$$

$$\delta(q_3, 0) = (q_3, 0, L), (q_0, 0, R)$$

$$\delta(q_3, 1) = (q_3, 1, L), (q_3, y, R)$$

$$\delta(q_3, x) = (q_3, x, R)$$

$$\delta(q_3, y) = (q_3, y, L)$$

$$\delta(q_3, B) = (q_4, B, R)$$

A TM that recognizes L is shown in Fig. 8.3.8.

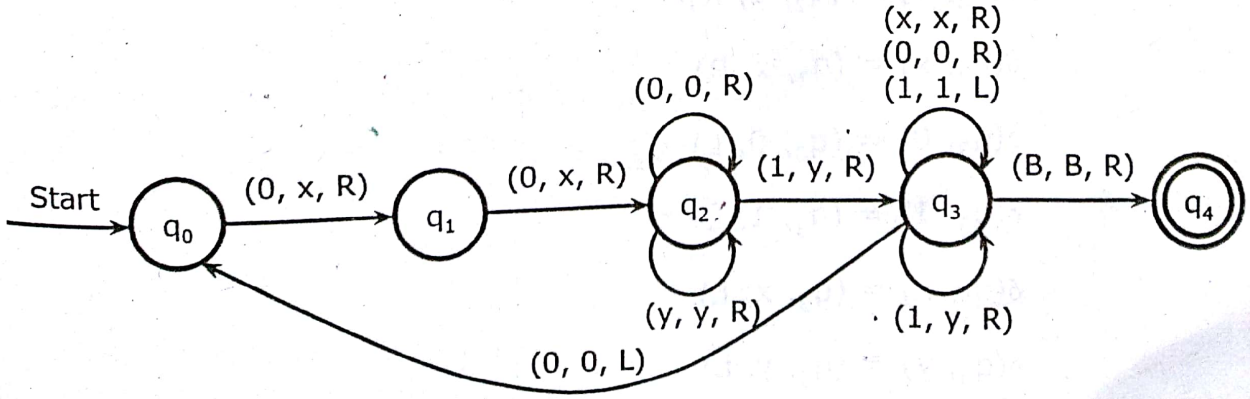



**Fig. 8.3.8** TM for Given Language

oufastupdates.com

**Design turing machine which will recognize strings containing equal number of 0's and 1's.**

**SOLUTION**

The tuple representation of TM is,

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

Where $\delta$ is defined as follows,

$$\delta(q_0, 0) = (q_1, M, R), \qquad \delta(q_0, 1) = (q_2, M, R)$$
$$\delta(q_0, M) = (q_0, M, R), \qquad \delta(q_0, B) = (q_4, B, R)$$
$$\delta(q_1, M) = (q_1, M, R), \qquad \delta(q_1, 0) = (q_1, 0, R)$$
$$\delta(q_1, 1) = (q_3, M, L), \qquad \delta(q_2, M) = (q_2, M, R)$$
$$\delta(q_2, 1) = (q_2, 1, R), \qquad \delta(q_2, 0) = (q_3, M, L)$$
$$\delta(q_3, 0) = (q_3, 0, L), \qquad \delta(q_3, 1) = (q_3, 1, L)$$
$$\delta(q_3, M) = (q_3, M, L), \qquad \delta(q_3, B) = (q_0, B, R)$$

The turing machine works as follows,

(i) In state $q_0$, the first occurrence of 0 or 1 is replaced by M. Then the machine moves to $q_1$ or $q_2$ depending on which symbol was changed.

(ii) If machine moves to $q_1$, it implies that previous input symbol was 0, so it replaces the first 1 by M and switches to state $q_3$.

(iii) If machine moves to $q_2$, it implies that previous input symbol was 1, so it replaces the first 0 by M and switches to state $q_3$.

(iv) In state $q_3$, the machine moves back to the beginning of the string and the machine state is changed to $q_0$.

(v) In state $q_4$, the turing machine accepts the input if all input symbols are replaced by M.

The transition diagram for turing machine as shown in Fig. 8.3.10.

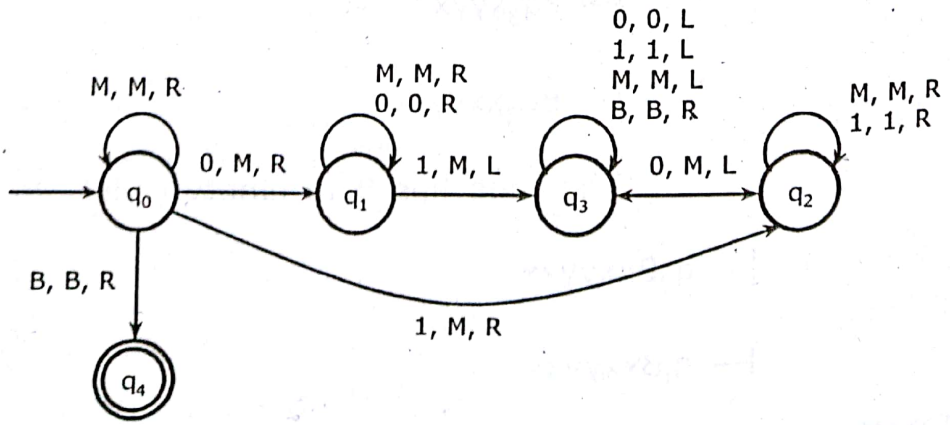



**Fig. 8.3.10** Transition Diagram for Turing Machine

oufastupdates.com

| | |
|---|---|
| Δ10000$01Δ | Move left till 1 and replace 1 with Y. |
| ΔY0000$01Δ | Move right till 1 and replace 1 with X. |
| ΔY0000$0XΔ | Move left till Δ. |
| ΔY0000$0XΔ | Move right and replace Y with 1 and X with '0'. |
| Δ10000$00Δ | Move right till Δ. |
| Δ10000$00Δ | Halt (∵ One of the side contains all zero's). |

## SOLVED PROBLEM 6

**Design turing machine for L = {$a^n b^n c^n$ | n ≥ 1}.**

**SOLUTION**

Consider the turing machine,

$$M = (Q, \Sigma, \Gamma, \delta, q_0, b, f)$$

Where,

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

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

$\Gamma = \{a, b, c, x, y, z, b\}$

$f = \{q_6\}$

Consider the transition diagram to recognize the language,

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

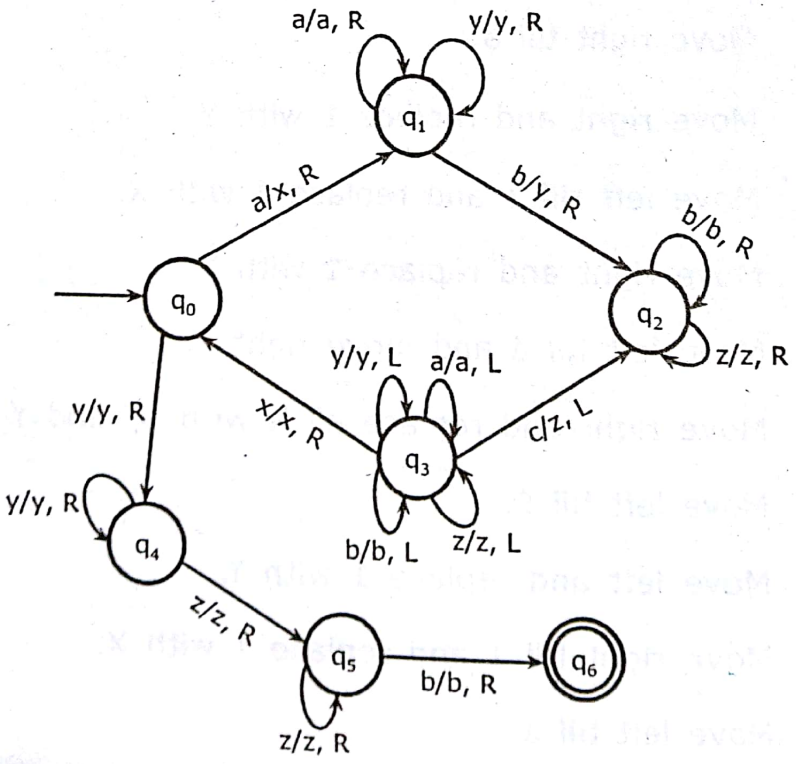



**Fig. 8.3.12** Transition State Diagram

oufastupdates.com

**Table 8.3.5** Transition

| Present State | Tape Symbol | | | | | | |
|---|---|---|---|---|---|---|---|
| | a | b | c | x | y | z | b |
| $\rightarrow q_0$ | $(q_1, x, R)$ | - | - | - | $(q_4, y, R)$ | - | - |
| $q_1$ | $(q_1, a, R)$ | $(q_2, y, R)$ | - | - | $(q_1, y, R)$ | - | - |
| $q_2$ | - | $(q_2, b, R)$ | $(q_3, z, L)$ | - | - | $(q_2, z, R)$ | - |
| $q_3$ | $(q_3, a, L)$ | $(q_3, b, L)$ | - | $(q_0, x, R)$ | $(q_3, y, L)$ | $(q_3, z, L)$ | - |
| $q_4$ | - | - | - | - | $(q_4, y, R)$ | $(q_5, z, R)$ | - |
| $q_5$ | - | - | - | - | - | $(q_5, z, R)$ | $(q_6, b, R)$ |
| $q_6$ | - | - | - | - | - | - | - |

Initial by the read/write head of the T.M. is at $q_0$ when it reads a, it replace a with x and advances to right and enters into $q_1$. At $q_1$ whenever it reads b, it replace it by y and advances to right and enters into $q_2$. At $q_1$ when it reads c, it replace c by z and advances to left. During moving in the left direction when it reads x, y, z it replaces them by y, z respectively. Whenever it reads x at $q_3$ it moves to right and enters into $q_0$. After replacing all strings it moves to right until it reads blank symbol b and enters into final state $q_6$.

## SOLVED PROBLEM 7

**Design turing machine to recognize the palindromes of digits {0, 1}. Give its state transition diagram also.**

**SOLUTION**

The construction is made by defining moves in the following manner,

(i) The machine scans the first input symbol (either 0 or 1), erases (but remembers) it, writes a blank symbol in place and changes state ($q_1$ or $q_2$).

(ii) It scans the remaining part without changing the tape symbol until it encounters b. It then moves the read/write head a step left. If the rightmost symbol tallies with the leftmost symbol, the rightmost symbol is erased. Otherwise TM halts. The read/write head moves to the left until b is encountered.

The above steps are repeated after changing the states suitably.

oufastupdates.com

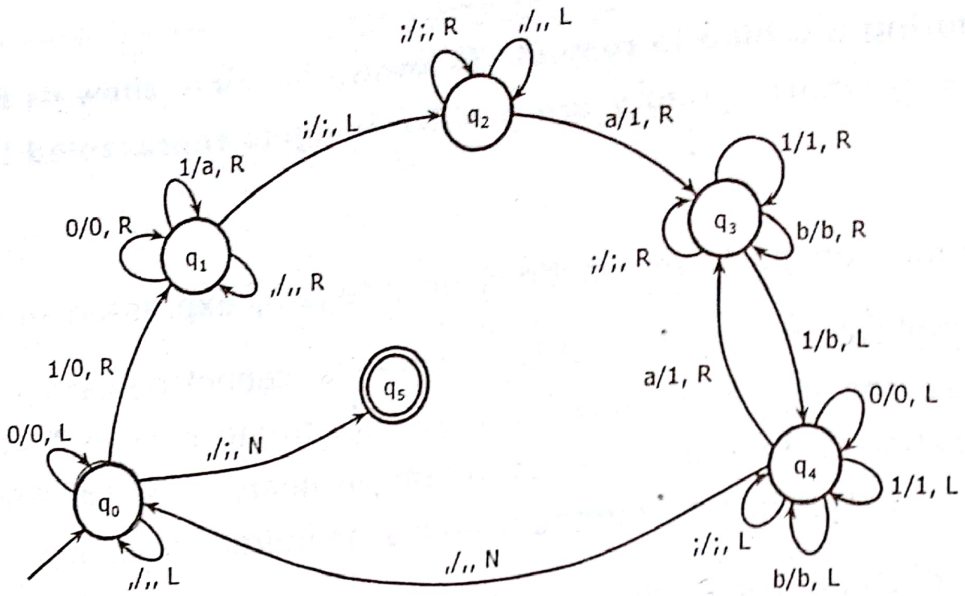


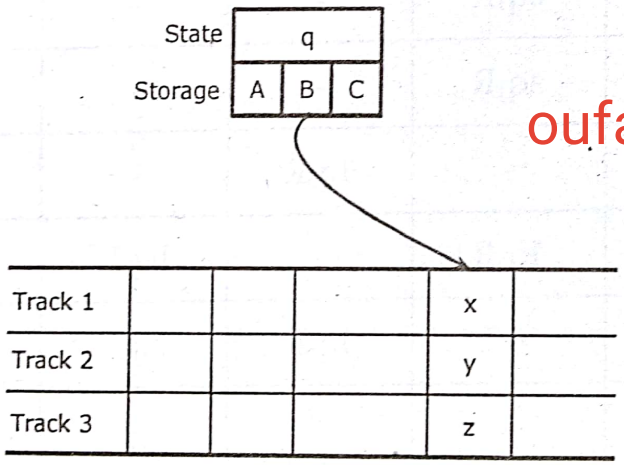

**Fig. 8.3.14** State Transition Diagram

## PROGRAMMING TECHNIQUES FOR TURING MACHINES

A turing machine can be used to compute in a manner not unlike that of conventional, eventually, TM is exactly as powerful as a conventional computer. In particular we shall learn that the turing machine can perform the sort of calculations.

### Storage in the State

The model of turing machine has a finite control. This finite control can be used to hold some amount of information. In the Fig. 8.4.1 shows this technique (other idea, multiple to access). There we see the finite control consisting of not only a "control" state q, but three data elements, A, B and C. Using this technique no expression to the TM model but we will think interms state as a tape. In Fig. 8.4.1, we will consider the number of states as [q, A, B, C].




oufastupdates.com

**Fig. 8.4.1** Turing Machine for Finite Control and Multiple Tracks

**Example :** Consider the following example for design a TM,

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

That stores in its finite control the first symbol (a or b) and checks that it does not appear else where on the input. So, the machine M accept the large L(M) = 01* + 10*.

The set of states Q is $\{q_0, q_1,\} \times \{a, b, B\}$ i.e., these states may considered as pairs with two components.

(1) A control position is either $q_0$ or $q_1$, first stores what the sequence of operation in TM,

(i) If control state is $q_0$ means M has not yet read its first symbol.

(ii) If control state is $q_1$ means it has read the input symbol and achieving that same symbol does not appear else where moving right and reach blank all hopefully.

(2) A data portion, which stores the first symbol will be looked and it should be a or b. The symbol B in this component means that no symbol has read.

The transition facilities $\delta$ of M is as follows,

(1) $\delta([q_0, B], x) = ([q_0, x], x, R)$ for x = a or x = b, Initially, $q_0$ is the control state and data portion of the state is B. The input symbol read is copied into second component of the state and M moves right, entering control state $q_1$ on it does so.

(2) $\delta([q_1, a], \bar{x}) = ([q_1, a], \bar{x}, R)$ where $\bar{x}$ is an "complement" of a i.e., a if x = b and b if x = a. In state $q_1$, it skips over every symbol a or b i.e., different from the one it has stored in its state and moves right continuously.

(3) $\delta([q_1, x], B) = ([q, B], B, R)$ for x = a or x = b if M reaches an first empty, it reaches accepting state [q, B].

## Multiple Track

In the standard turing machine, the tape was divided into squares where each square holds only one symbol. It can be extended so that each tape consist of several tracks. This can be done by dividing the tape into number of tracks and each track is divided into a number of squares with each square holding only one symbol. If there are n-tracks and the read/write head points to $m^{th}$ square, then all the symbols under different tracks beneath that read/write head are the symbols to be scanned. So, all the symbols under the read/write head can be considered as set of characters under one square. Such a TM is called turing machine with multiple tracks.

## Subroutines

In the high level languages use of subroutines built the modularity in the program development process. The same type of concept can be introduced in construction of TM. We can write the subroutines as a turing machine.

First, a TM program for the subroutine is written. This will have an initial state and a 'return' state. After reaching the return state, there is a temporary halt. For using a subroutine, new states are introduced. When there is a need for calling the subroutine, moves are effected to enter the initial state for the subroutine (when the return state of the subroutine is reached) and to return to the main program of TM.

oufastupdates.com

## SOLVED PROBLEM 1

**Design a turing machine to find the two's complement of a given binary number.**

**SOLUTION**

Method for finding two's complement of number. Read the binary number from LSB to MSB, from least significant bit we keep all zero's as it is and move left until we do not get 1. As we get the first 1 from LSB we move left and we convert 0 to 1 and 1 to 0 on moving to the left. The process is continued until we get the left most blank symbol b.

**Table 8.4.1**

| | |
|---|---|
| b$\underset{\uparrow}{0}$110b | Move to right most end until we get blank symbol b. |
| b0110$\underset{\uparrow}{b}$ | Move to left. |
| b011$\underset{\uparrow}{0}$b | Keep this 0 as it is and move left. |
| b01$\underset{\uparrow}{1}$0b | As the first 1 from LSB has scanned to keep it as it is and move left. |
| b0$\underset{\uparrow}{1}$10b | Complement it to 0 and move left. |
| b$\underset{\uparrow}{0}$010b | Convert it to 1 and move left. |
| $\underset{\uparrow}{b}$1010b | Final or halt state is reached so stop. |

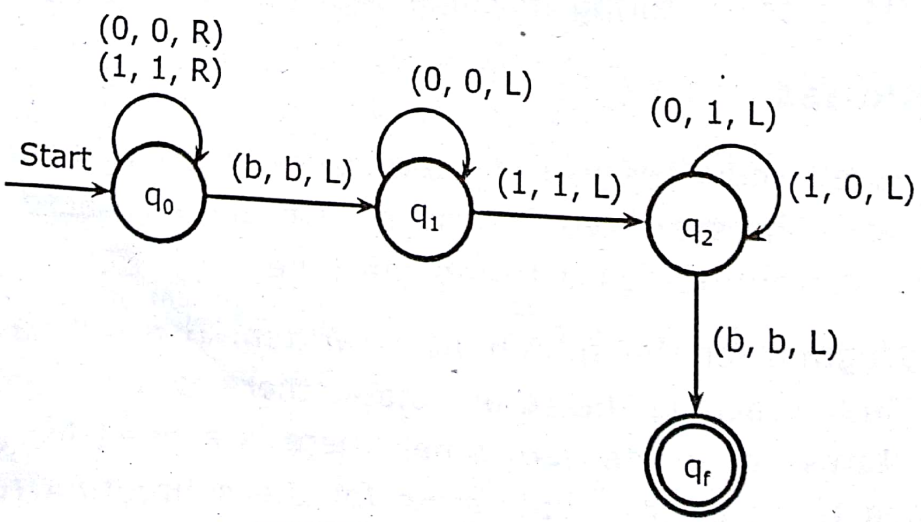



**Fig. 8.4.2** Amplifier

oufastupdates.com

**Table 8.4.2** Transition

| Present State | Tape Symbol | | |
|---|---|---|---|
| | 0 | 1 | b |
| $\rightarrow q_0$ | $0Rq_0$ | $1Rq_0$ | $bLq_1$ |
| $q_1$ | $0Lq_1$ | $1Lq_2$ | - |
| $q_2$ | $1Lq_2$ | $Lq_2$ | $bLq_f$ |
| $q_f$ | - | - | - |

## LVED PROBLEM 2

**Construct TM for checking well firmness of the parenthesis.**

**OLUTION**

This is problem which can be solved by both turing machine and pushdown automata. Here we are concerned with turing machine. We solve this problem by taking an example as follows,

**Example**

$\Delta\underset{\uparrow}{(}\,)(\,)\Delta$ Move right search of ')', skipping '('(.

$\Delta\underset{\uparrow}{(}\,)(\,)\Delta$ Move it as * and move left, in search of '('.

$\Delta\underset{\uparrow}{(}\,*(\,)\Delta$ Now '(' has encountered mark it as * move left in search of '(', mark it.

$\Delta*\underset{\uparrow}{*}(\,)\Delta$ Move right, skip*.

$\Delta**\underset{\uparrow}{(}\,)\Delta$ Move right.

$\Delta**(\underset{\uparrow}{*}\Delta$ Convert to * and move left.

$\Delta****\underset{\uparrow}{}$ Move right till Δ.

$\Delta*****\underset{\uparrow}{\Delta}$ HALT

oufastupdates.com

The TM will look like this,

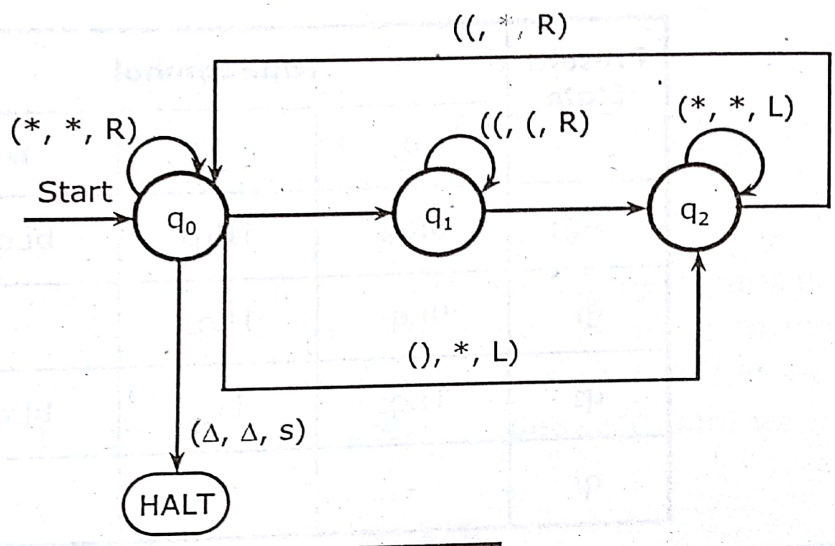



Fig. 8.4.3

The transition table for this TM will be,

Table 8.4.3

| State | ( | ) | * | Δ |
|---|---|---|---|---|
| $q_0$ | $(q_1, (, R)$ | $(q_2, *, L)$ | $(q_0, *, L)$ | (HALT, Δ, S) |
| $q_1$ | $(q_1, (, R)$ | $(q_2, *, L)$ | - | - |
| $q_2$ | $(q_2, *, (, R)$ | - | $(q_2, *, L)$ | - |
| HALT | - | - | - | - |

oufastupdates.com

## RESTRICTED TURING MACHINES

So far in the previous sections, we have discussed various concepts of Standard Turing Machines and other variations of TM. Now, we shall concentrate on the other variations of TM by imposing certain restriction on TM. Instead of providing the complete simulation, we shall provide only broad outline to show that the machines are equivalent. We can have so many variations of standard turing machines with minor modification.

We can have the following turing machines,

(1) Turing machines with semi-infinite tape.

(2) Multi-stack machines.

(3) Counter machines.

(4) Off-line Turing machine.

(5) Linear bounded automata.

### Turing Machine with Semi Infinite Tape

In the standard turing machine, there was no boundary specified for the left side as well as right side of the tape. The read/write head can be moved infinitely towards left as well as towards right i.e., the tape was unbounded in both the directions. Now, if we restrict the read/write head to move only in one direction say towards right (i.e., bounded on the left and unbounded on the right), then the turing machine is called TM with semi-infinite tape and it can be visualized as shown in Fig. 8.6.1.

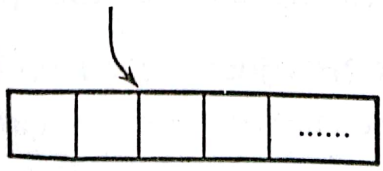



**Fig. 8.6.1** TM with Semi-Infinite Tape

oufastupdates.com

It is clear from the Fig. 8.6.1 that there are no cells to the left hand side of the initial head position and this restriction does not affect the power of the machine. The TM M can simulate the semi-infinite TM $M_S$ using two tracks for a tape as shown in Fig. 8.6.2.

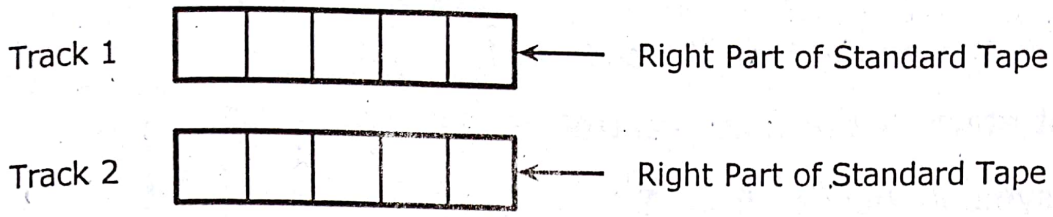



**Fig. 8.6.2** Simulation of Semi-infinite Tape

The upper part represents track1 and in these cells let us store the information which lies to the right of some reference point. Initially, the reference point could be the position of the read-write head. The lower part i.e., track 2 contains the left part of the reference point in reverse order. The machine M which is simulating $M_S$ will use the information of track 1 as long as the read-write head of the $M_S$ is towards right of the reference point. When $M_S$ moves towards left of the reference point, the machine M uses cells of track 2 from right to left. We can partition the states of M into $Q_1$ and $Q_2$ so that the states of $Q_1$ are used when working with respect to track 1 and the states of $Q_2$ are used when working with respect to track 2. Some end markers such as #'s can be placed at the left hand side of track 1 and track 2 to enable the programmer to switch between the tracks. Following in this direction, we can show that M and $M_S$ are equivalent.

## Multistack Machines

All the languages which are accepted by PDA are accepted by TM. At the same time, any language which is not accepted by PDA is also accepted by TM. TM is a generalized machine which can accept all languages. The multi-stack machine is generalized model of PDA. A machine with three stacks is shown in Fig. 8.6.3.

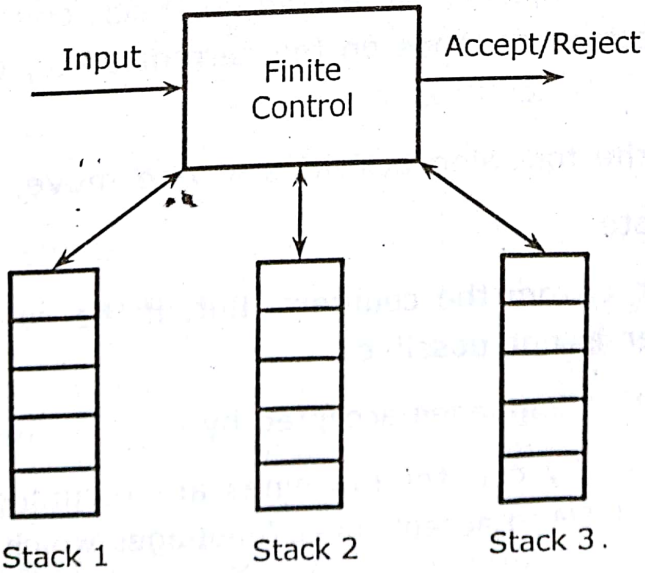



**Fig. 8.6.3** A Machine with Three Stack Ks

oufastupdates.com

The pictorial representation of 3-stack machine will have 3 stacks. In general, a k-stack machine is a deterministic PDA with k stacks. Similar to the PDA, it accepts input chosen from the alphabets $\Sigma$ and has a finite control. All the k stacks will have the alphabets chosen from stack alphabet $\Gamma$.

The move of the multi-stack depends on,

(1) The current state of the finite control.

(2) The input symbol choosen from $\Sigma$.

(3) The symbol on top of each stack.

Once the transition is applied for the current state, input symbol and the current top of each stack, the multistack machine may,

(1) Change the state (or) it may remain the same state.

(2) It may replace the top of the stack.

Thus, the general transition for a k-stack machine may take the form,

$\delta(q, a, X_1, X_2, \ldots X_k) = (p, \alpha_1, \alpha_1, \alpha_1 \ldots \alpha_k)$ where $X_i$ is on top of the stack for $1 \leq i \leq k$.

It is clear from this transition that each symbol on top of the stack can be replaced by different symbols. It can be shown that any language accepted by turing machine, is also accepted by two-stack machine.

### 8.6.3 Counter Machines

The counter machine is a restricted multi-stack machine which can be interpreted in several ways.

The counter machine is similar to that of multistack machine with some modifications. Each stack in the machine is replaced by a counter. Each counter holds a nonnegative number. The move of a machine depends on the current state, current input symbol and if one of the value of the counter is zero.

The machine can do the following activities in one move,

(1) It can change the state.

(2) It can add or subtract 1 from the counters. But, if the counter is zero, subtracting of 1 from that counter is not possible.

Some observations of the languages accepted by counter machines are shown below,

(1) The languages accepted by counter machines are recursively enumerable i.e., we can obtain an equivalent TM to accept those languages which are accepted by counter machines.

(2) A language accepted by only one counter machine is context free language.

oufastupdates.com

## Off Line Turing Machine

A multitape Turing machine whose input tape is read only is an off line Turing machine. Usually, the Turing machine is not allowed to move the input tape head off the region between the end markers ¢ and $. A model of an off line Turing machine is shown below in Fig. 8.6.4.

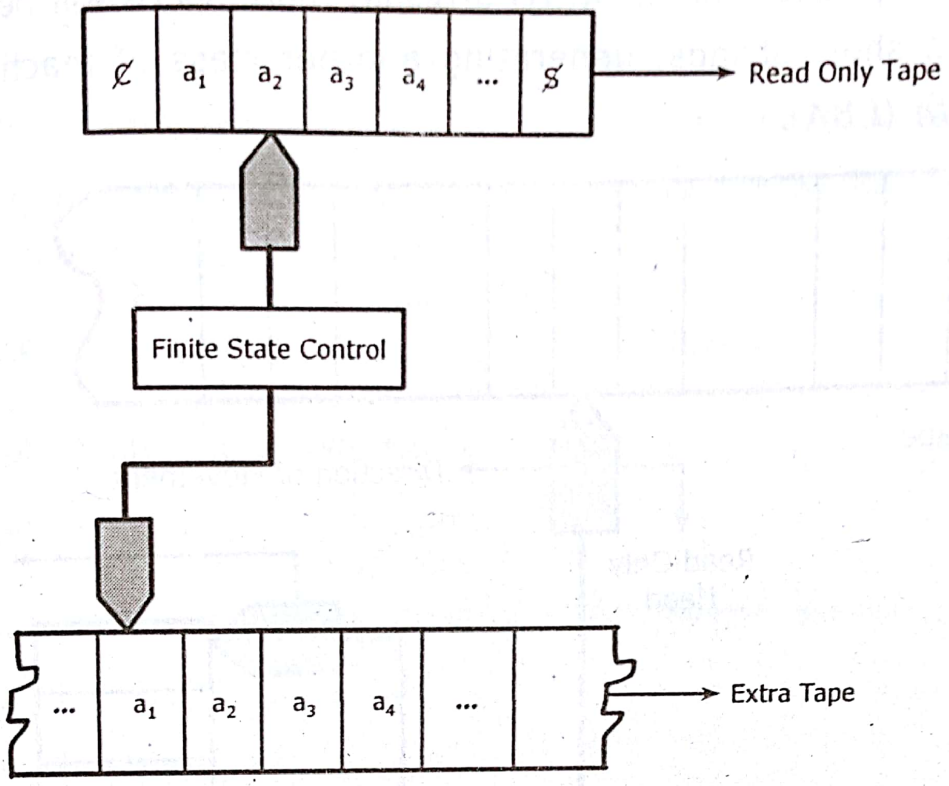



**Fig. 8.6.4** Off Line Turing Machine

This machine is no more efficient than standard turing machine as it is a special case multitape turing machine. On the other hand, an off line turing machine can simulate standard turing machine M by introducing one more tape than M. This is because the f line turing machine makes a copy of its own input into the extra tape, and then it mulates standard turing machine M as the extra tape has only M's input.

The formal definition of off line turing machine is given below,

**efinition :** The turing machine,

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

Where,

$Q$ = Set of finite states.

$\Sigma$ = Set of input alphabets.

$\Gamma$ = Set of tape symbols.

$\delta$ = Transition function from $Q \times \Sigma \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$.

$q_0$ = The start state.

$B$ = A special symbol indicating blank character.

oufastupdates.com

## Linear Bounded Automata (LBA)

The model of computation which we shall examine here is the linear bounded automaton. The power of the standard turing machine cannot be extended by complicating the structure of tape. It is possible to limit it by restricting the way in which the tape can be used. The way of limiting the tape uses it to allow the machine to use only that part of the tape occupied by the input. As a result, more space will be available for long input strings than short strings, generating another class of machines called *linear bounded automata (LBA).*

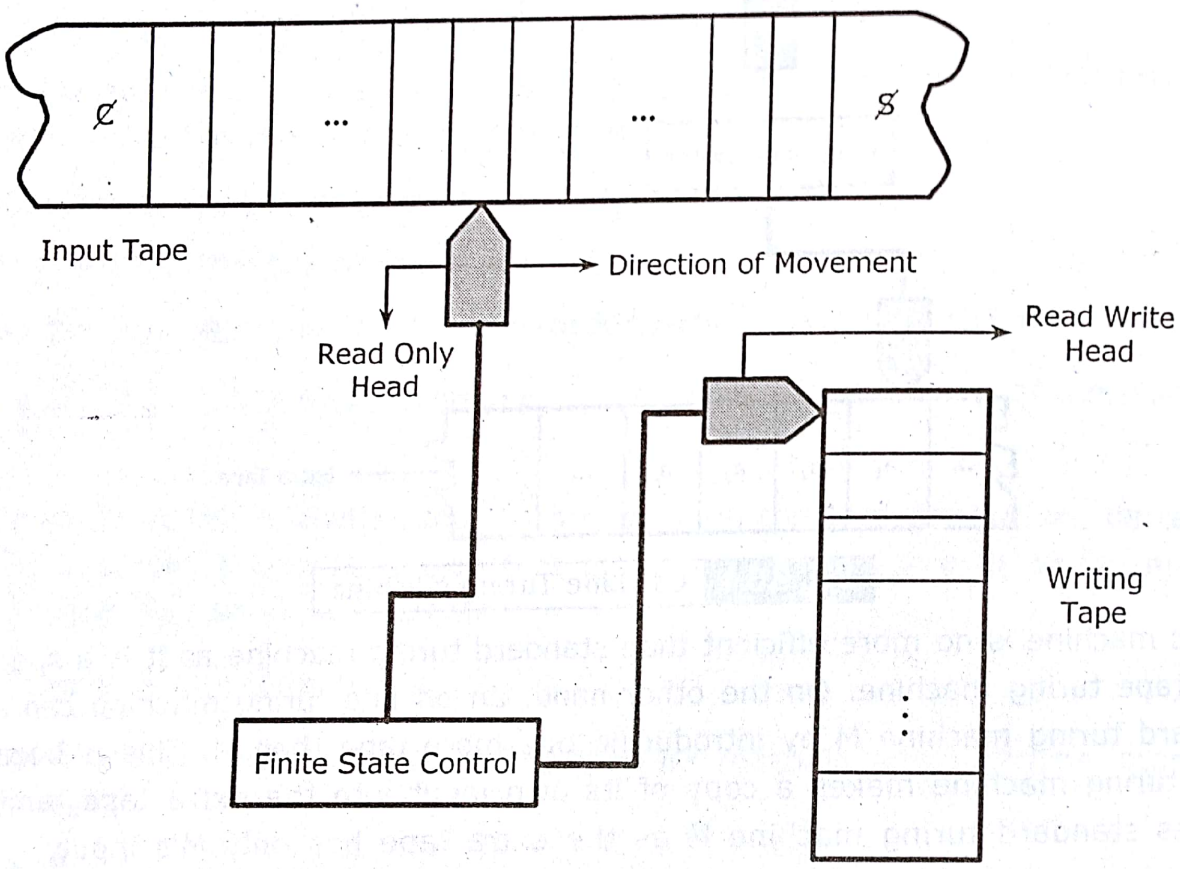



**Fig. 8.6.5** Model of Linear Bounded Automata

The model of linear bounded automata consists of an input tape and a working tape shown in Fig. 8.6.5. A read only head is associated with the input tape and is allowed to move only in right direction. Another read write is associated with the working tape. It can modify the contents of working tape in any way without any restriction. The left end and right end of the input tape is marked with ¢ and $ respectively. The end markers ¢ and $ on the input tape decide the length of input tape and never allow the machine to go pass them. This will ensure that the storage bounds are maintained and help keep our machines from leaving their tapes. If the input tape contains n-cells, then the length of the input string 'w' will be of maximum (n – 2). The string 'w' will be accepted by a linear bounded automaton if there is a Turing machine that accepts 'w' using at most $k_n$ cells of input tape, where k is a constant specified in the description of linear bounded automaton and it is completely independent of the input string.

oufastupdates.com

Some authors define a linear bounded automaton to use only the portion of the tape that is occupied by the input, that is k = 1 in all cases. The definitions lead to equivalent classes of machines, because we can compensate for the shorter tape by having a larger tape alphabet.

**NOTE :** ***That the following definition of linear bounded automaton is assumed to be essentially non-deterministic.***

Formally, a linear bounded automaton is a *nondeterministic turing machine* which has only one tape and this tape is exactly the same length as the input. We define a linear bounded automaton as eight tuple,

$$M = (S, \Sigma, \Gamma, \delta, s_0, ¢, \$, A) \text{ consisting of,}$$

S = Non-empty finite set of internal states.

$\Gamma$ = Non-empty finite set of tape symbols.

$\Sigma$ = Non-empty finite set of input symbols, including end markers ¢ and \$ which is a subset of $\Gamma$, i.e., $\Sigma \subseteq \Gamma - \{B\}$.

$s_0$ = Initial or start state, i.e., $s_0 \in S$.

A = Set of accepting states, i.e., $A \subseteq S$.

$\delta$ = Transition function from $S \times \Gamma \rightarrow S \times \Gamma \times \{L, R\}$.

The language accepted by the linear bounded automata M, L(M) is defined as,

$$L(M) = \{w | w \in (\Sigma - ¢, \$\})^* \text{ and } s_0 ¢ w \$ \vdash^* \alpha s \beta, \alpha, \beta \in \Gamma^*, s \in A\}$$

It differs from a turing machine in that while the tape is initially considered to have unbounded length, only a finite contiguous portion of the tape, whose length is a linear function of the length of the initial input, can be accessed by the read/write head. This limitation makes an LBA a more accurate model of computers that actually exist than a turing machine in some respects.

Linear bounded automata are acceptors for the class of context sensitive languages. The only restriction placed on grammars for such languages is that no production maps a string to a shorter string. Thus no derivation of a string in a context sensitive language can contain a sentential form longer than the string itself. Since there is a one to one correspondence between linear bounded automata and such grammars, no more tape than that occupied by the original string is necessary for the string to be recognized by the automaton.

oufastupdates.com

# TURING MACHINE AND COMPUTERS

In this we will discuss about what computers can do, when the numbers involved exceed normal limits that are built into the architecture of these machines.

## Turing Machine as a Computer of Integer Functions

Turing machine can be used as a computer of integer functions, in addition to using it as a language recognizer. Here integer function means a function whose arguments are integers and result is an integer.

**Example :** $f(i_1, i_2, \ldots\ldots i_k) = m$

Where,

$i_1, i_2, \ldots\ldots, i_k$ = Arguments of integers

m = A result.

For computing integer functions an integer $i \geq 0$ is represented using a string $0^i$, i.e., a string of i 0's. Therefore, if function of k arguments $f(i_1, i_2, \ldots, i_k) = m$ is to be computed, then the arguments $i_1, i_2, \ldots, i_k$, will be represented on the tape of the turing machine as a string $0^i\ 10^{i2}\ 1 \ldots 10^{ik}$, will be represented on the tape of the turing machine then carries out a sequence of moves, and when it halts leaves $0^m$ on its tape.

### EXAMPLE PROBLEM 8.8

**Design a turing machine that computes a function f(m, n) = m + n, i.e., addition of two integers.**

**SOLUTION**

The turing machine will have $0^m 1 0^n$ on its tape initially. Therefore, it will start with the leftmost 0, go on scanning 0's moving right keeping them as it is till it get 1, it replaces this 1 by 0, and then again go on scanning 0's, and moving right keeping them as it is. When it gets a B (blank), it moves left, and replaces the rightmost 0 by B, and halts, thereby leaving finally m + n 0's on the tape. Therefore, the moves of the turing machine are,

**Table 8.7.1** Moves of the TM to Compute f(m, n) = m + n

| | 0 | 1 | B |
|---|---|---|---|
| $\rightarrow q_0$ | $(q_0, 0, R)$ | $(q_1, 0, R)$ | - |
| $q_1$ | $(q_1, 0, R)$ | - | $(q_2, B, L)$ |
| $q_2$ | $(q_3, B, R)$ | - | - |
| $*q_3$ | - | - | - |

oufastupdates.com

Therefore, the turing machine M = ({$q_0$, $q_1$, $q_2$, $q_3$}, {0, 1}, {0, 1, B}, $\delta$, $q_0$, B, {$q_3$}), where $\delta$ is given above.

The transition diagram corresponding to the Table is shown in Fig. 8.7.1.

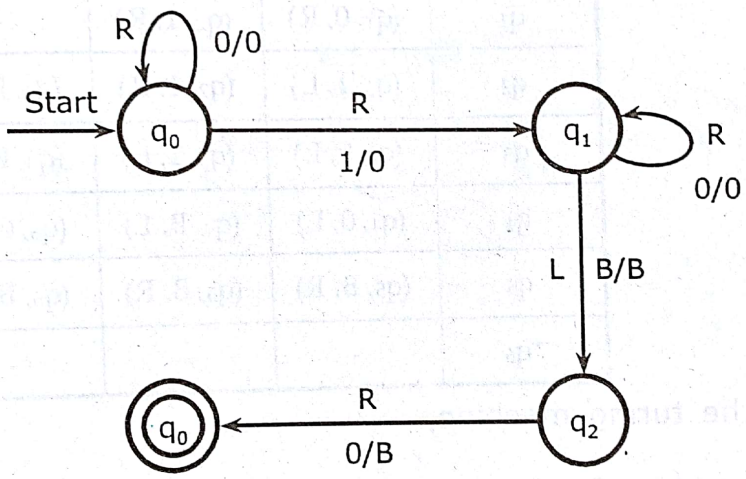



**Fig. 8.7.1** Transition Diagram to Compute f(m, n) = m + n

oufastupdates.com

## Turing Machine as a Generating Device

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 string 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 $\Sigma$ on its output tape, separated by marker symbol #. The set of all those strings that it can eventually write on it's 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 $\Sigma^*$, and W is eventually written on the output tape between pair of #'s$\}$.

oufastupdates.com

# *Long Questions ( UNIT 5)*

-> Undecidability

-> Recursive languages

-> Recursively Enumerable Languages

-> Post Correspondence Problem

-> Halting Problem

oufastupdates.com

# UNDECIDABILITY

The problem solved by the turing machine are classified into two classes,

(1) Those that have an algorithm.

(2) Those that are only solved by turing machines.

In this first case, the TM halts whether or not it accepts in input. In the latter case, the TM may run forever on inputs they do not accept.

**Undecidability :** If a problem cannot be solved or answered based on some algorithm then it is called undecidable.

(or)

Any problem for which no algorithm exist are called undecidable.

(or)

A problem is said to be undecidable if, its language is not recursive or it has no solution or answer or algorithm.

**Recursive Language :** A language is recursive if there exists a turing machine that accepts every string of the language and rejects the string that is not in the language.

**Decidable and Undecidable Problems :** A problem whose language is recursive is said to be decidable, otherwise it is undecidable.

A problem is said to be undecidable if there is no algorithm that takes an input, an instance of the problem and determines whether the output to that instance is 'Yes' or 'No'.

**Example**

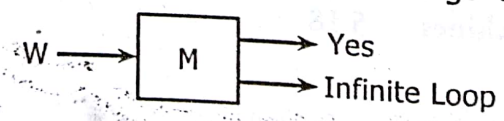
(1) The set of strings of equal number of 0's and 1's is a decidable problem.

(2) Turing machine M halts with empty tape is an undecidable problem.

Does the turing machine halt on input W is an undecidable problem.

## A LANGUAGE THAT IS NOT RECURSIVELY ENUMERABLE

A language is recursively enumerable if there exists a turing machine that accepts every string of the language and does not accept strings that are not in the language.




The long range *goal* of proving the undecidability of the recursively enumerable language W consisting of pairs such that,

(1) M is turing machine (suitably coded, i.e., binary) with input alphabet {0, 1}.

(2) W is a string of 0's and 1's.

(3) M accepts input W.

oufastupdates.com

If this problem with inputs restricted to the binary alphabet is undecidable, then turing machines with any other alphabet, is undecidable.

The inputs given to the turing machine should be in a well-formed representation using binary values.

## DIAGONALIZATION LANGUAGE

Coding a turing machine as a string of 0's and 1's and exhibit a language that is not even recursively enumerable, namely $L_d$ is intermediate goal which is *"diagonalization language"*. It consist of all strings W such that the M represented by W does not accept the input W. Therefore we can show that $L_d$ has no TM at all that accepts it.

**NOTE :** *That showing there is no turing machine at all for a language is showing something stronger than that the language is undecidable (i.e., it has no algorithm or TM that always halts).*

### Enumerating the Binary Strings

It is necessary to assign integer values to the binary strings so that each string corresponds to one integer and each integer corresponds to one string.

**Example :** Let W be the binary string.

Take 1 W as binary integer i.

This implies W as the $i^{th}$ string.

Like that we can say.

Empty (ε) symbol is the first string (and not 0 and 1), 0 is the second, 1 is the third, 00 the fourth, 01 the fifth and so on. Strings are ordered by length and strings of equal length are ordered lexicographically. Therefore the integer i can be represented by $W_i$.

## EXAMPLE PROBLEM 9.1

**What string w37 is?**

**SOLUTION**

The online solution gives 37 is binary (100101) and removes the leading 1 to get 00101.

### Codes for Turing Machines

To represent a TM = (Q, {0, 1}, ⊢, δ, $q_1$, B, F) as a binary string, first assign integers to states, tape symbols and direction L and R,

(1) Assume the states are $q_1, q_2 \ldots q_k$ for some K. Using the integers available in the suffix of each state, the string can be represented as $q_1 \rightarrow 0$, $q_2 \rightarrow 00$, $q_3 = 000$ etc.

oufastupdates.com

(2) Assume the tape symbols 0, 1, B are represented as $x_1, x_2, x_3 \ldots x_m$.

Where,

$x_1 = 0$

$x_2 = 1$

$x_3 = B$

(3) Assume the directions are represented as $D_1$ and $D_2$ where,

After representing each state, symbol and directions using integers, we can encode the transition function.

Suppose $\delta(q_i, x_j)\ (q_x, x_l, D_m)$ for some integers i, j, k, *l*, m,

$$*q_1 \rightarrow 0,\ q_2 \rightarrow 00,\ q_3 = 000$$

Then the encoded string is given by $0^i\ 1\ 0^j\ 1\ 0^k\ 1\ 0^l\ 1\ 0^m$.

The code for the entire turing machine M consists of all the strings for the transition in some order, separated by pairs of 1's.

i.e., $111\ C_1 11\ C_2 11\ C_3 11\ C_4 11 \ldots\ C_{n-1}\ 11 C_n 111$

Where each of the C's is the code for one transition of M.

## EXAMPLE PROBLEM 9.2

**Let the TM in question be,**

**$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$.**

**SOLUTION**

Where $\delta$ is defined by,

$$\delta(q_1, 1) = (q_3, 0, R)$$

$$\delta(q_3, 0) = (q_1\ 1, R)$$

$$\delta(q_3, 1) = (q_2, 0, R)$$

$$\delta(q_3, B) = (q_3, 1, L)$$

And codes for the transitions rules,

0100100010100

0001010100100

00010010010100

0001000100010010

oufastupdates.com

(iii) $\delta(q_3, 1) = (q_2, 0, R)$

$C_3 \Rightarrow 0^3\ 10^2\ 10^2\ 10^1\ 10^2$

(iv) $\delta(q_3, B) = (q_3, 1, L)$

$C_4 \Rightarrow 0^3\ 10^3\ 10^3\ 10^2\ 10^1$

Then the complete code is given by,

M =0100100010100 **11** 0001010100100 **11** 00010010010100 **11**

0001000100010010

Code for <M,1101>, is given by,

= <1110100100010100 **11** 0001010100100 **11** 00010010010100 **11**
0001000100010010111, 1101>

## The Diagonalization Language

Some integers do not belong to any turing machine. If is not a valid TM code, then take the turing machine $M_i$ to be the TM with one state and no transitions. So far these values of i, $M_i$ is a turing machine that halts on any input.

$$\boxed{L(M_i) = \phi \text{ if } w_i \text{ fails to be a valid } T_M \text{ code}}$$

The diagonalization language $L_d$, is the set of strings $w_i$ where is not in $L(M_i)$.

$L_d$ consists of all strings w such that the turning machine M whose code in w does not accept the input w.

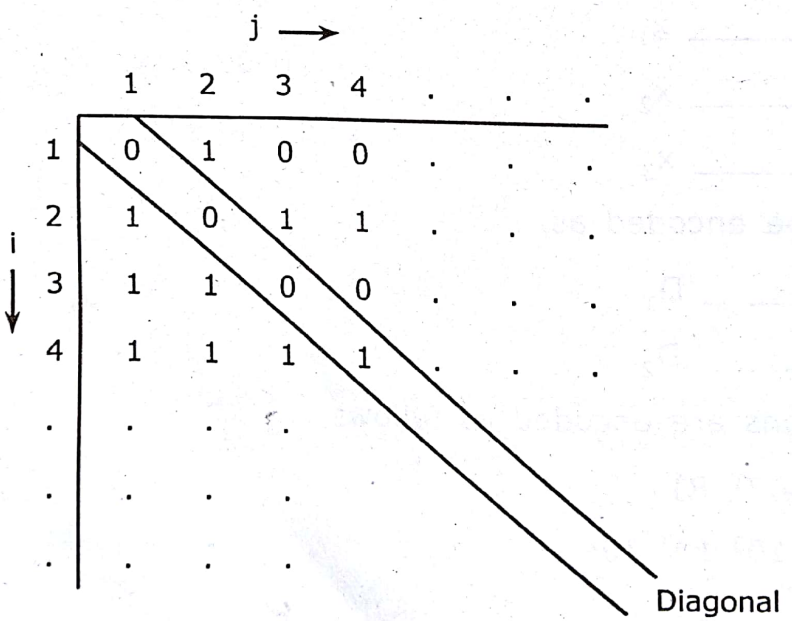



**Fig. 9.2.1** The Table that Represents Acceptance of Strings by Turing Machines

oufastupdates.com

It is clear that whether the $T_m$ $M_i$ accepts the input string or not.

If (i, j) = 1, Yes it is accepted.

If (i, j) = 0 No, it is rejected.

The $i^{th}$ row be called as the characteristic vector for the language $L(M_i)$, where 1's in this row correspond to the members of this language.

In order to find $L_d$, we need to complement the diagonal. In the table the diagonal value be 0001. And then the complemented diagonal = 1110.

Therefore, $L_d$ contains,

$w_1 = e$

$w_2 = 0$

$w_3 = 1$

And does contain $w_4 = 00$ and so on.

**Diagonalization :** The process of complementing the diagonal to construct the characteristic vector of a language that cannot be the language that appears in any row is called Diagonalization.

Then the complement of the diagonal cannot be the characteristic vector of any turing Machine.

## Proof That $L_d$ is Not Recursively Enumerable

Now, we should prove that, there is no turning machine that accepts the language Ld.

THEOREM

**$L_d$ is not a recursively enumerable language. That is, there is no turing machine that accepts $L_d$.**

PROOF

Suppose $L_d$ is accepted by some TM M defined by L(M). Since $L_d$ is a language over alphabet {0, 1}, M would be in the list of turing machines constructed, where it includes all turing machines with input alphabet {0, 1}.

So there may be at least one code for M, say i, that is,

$$M = M_i$$

Then whether $w_i$ is in $L_d$.

By definition,

$L_d = \{w_i \mid M_i \text{ does not accept } w_i\}$

oufastupdates.com

Here we have two possibilities,

**(i)** $w_i \Sigma L_d$ : This means that (i, i) entry is '0' and so $M_i$ does not accept $w_i$. But our assumption here is that there exists a turing machine $M_i$, which accepts $w_i$. There is a contradiction.

**(ii)** $w_i \Sigma L_d$ : This means that (i, i) entry as '1' and so $M_i$ accept $w_i$. But by definition of $L_d$, $M_i$ does not accept $w_i$. So there is a contradiction.

Thus it is clear that $L_d$ is not recursively enumerable, and $L_d$ is not recursive too.

## AN UNDECIDABLE PROBLEM THAT IS RE

The structure of Recursively Enumerable (RE) language is redefined into two classes,

(1) An algorithm which recognizes the language as well as the time of deciding whether the input string is not in the language. Such a turing machine always halts regardless of whether or not it reaches an accepting state.

(2) Languages consisting of RE languages that are not accepted by any turing machine with the guarantee of handling.

### Recursive Languages

A language L is recursive if L = L(M) for some turing machine M such that,

(1) If w is in L, then M accepts and goes to halt state.

(2) If w is not in L, then M halts even though it never enters an accepting state.

A language which is having a definite algorithm is called *Recursive language.*

If a problem is defined using recursive language, then that problem is *decidable* or else it is *undecidable.*

The below diagram suggest the relationship between three classes of languages.

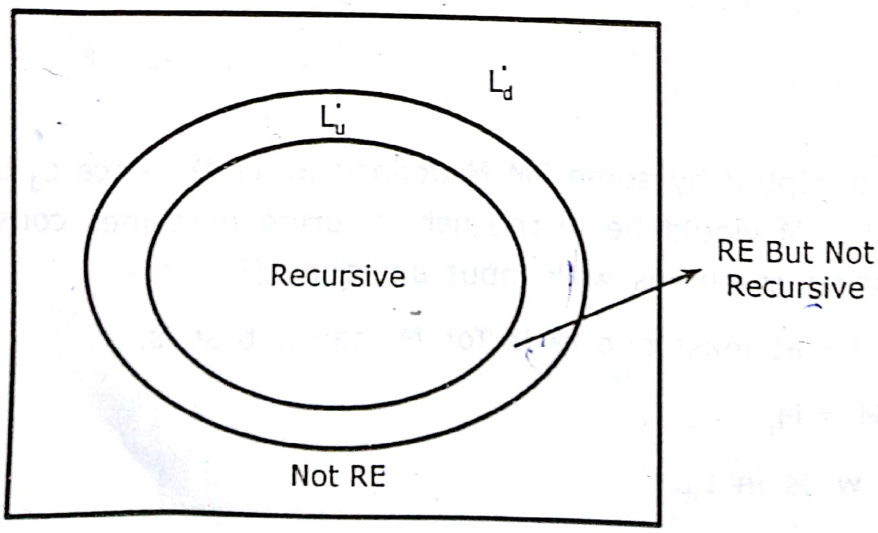



**Fig. 9.3.1** Relationship Between the Recursive, RE and Non-RE languages

oufastupdates.com

## Complements of Recursive and $R_E$ Languages

The recursive languages are closed under complementation.

**THEOREM : 1**

**If L is a recursive language, so is $\bar{L}$.**

**PROOF**

Let L be a recursive language and M a turing machine that halts on all inputs and accepts L. Construct a turing machine M′ from M such that,

(i) The accepting states of M are made non accepting states of M with no transitions.

(ii) Create a new accepting state P which has no transitions.

(ii) For each pair of non accepting state of M and a tape symbol of M such that M has no transition. Make the transition to the accepting state P′.

If M enters a final state on input w, then M halts without accepting.

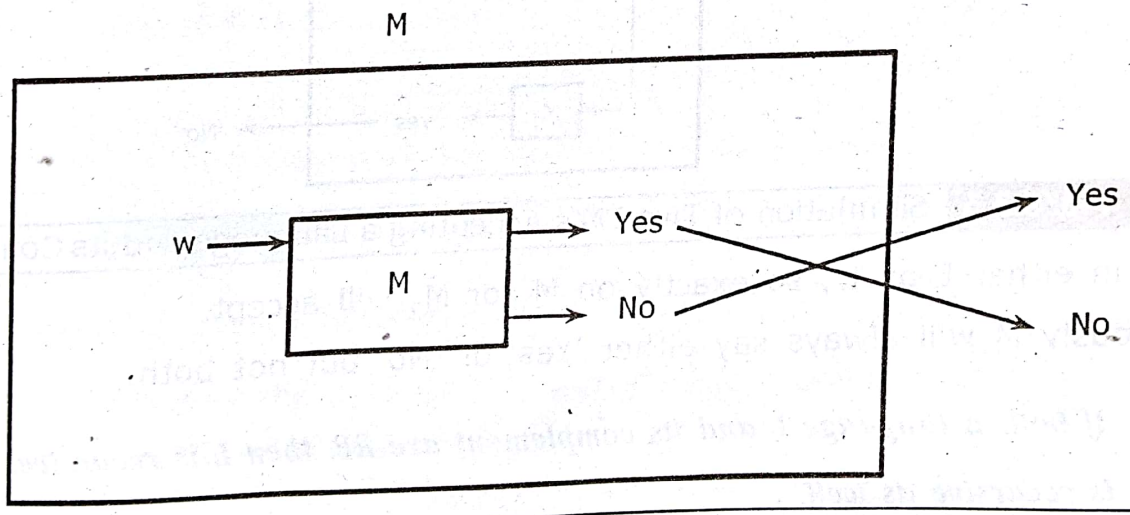



**Fig. 9.32** Construction of a TM Accepting the Complement of a Recursive Language

If M halts without accepting, M enters into final state.

| M | $\bar{M}$ |
|---|---|
| Accept → final state | Rejects |
| Rejects | Accept → final state |

It is clear that $\bar{M}$ has either of the event accepts or rejects. So $\bar{M}$ accepts I which is also recursive.

oufastupdates.com

## THEOREM : 2

**If a language $\bar{L}$ and its complement L are both recursively enumerable, then L is recursive.**

**PROOF**

Let $M_1$ and $M_2$ be the turing machine which accepts the language L and $\bar{L}$ respectively. Construct M to simulate simultaneously $M_1$ and $M_2$.

| $M_2$ | $M_1$ | M |
|---|---|---|
| 0 | 1 | Accepts |
| 1 | 0 | Rejects |

M accepts w if $M_1$ accepts w.

M rejects w if $M_2$ accepts w.

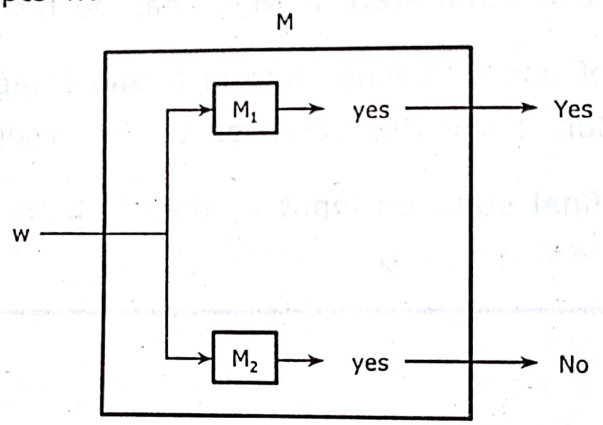



**Fig. 9.3.3** Simulation of Two TM's Accepting a Language and its Complement

w is in either L or $\bar{L}$, so exactly on $M_1$ or $M_2$ will accept.

Obviously M will always say either 'Yes' or 'No' but not both.

**NOTE :** ***If both a language L and its complement are RE, then L is recursive. Then by above theorem $\bar{L}$ is recursive as well.***

## The Universal Language

The *diagonalization language* ($L_d$) is the language consisting of strings such that does not accept,

$$L_d = \{w_i \mid w_i \Sigma L(M_i)\}$$

Consider the binary string representation <M, w> such that M accepts w called as *universal language* ($L_u$),

$$L_u = \{<M, w> \mid M \text{ accepts } w\}$$

$L_u$ is the set of strings representing a TM and an input accepted by that TM. So there is a TM U called a *Universal Turing Machine.*

oufastupdates.com

THEOREM

**$L_u$ is recursively enumerable.**

PROOF

In order to prove this theorem, it is necessary to construct a turing machine U that accepts $L_u$. The turing machine U consists of a three track input tape where the first track holds the input tape (<M, w>), the second track contains the tape of M where tape symbols are written in unary form and the third track represents the state $q_i$ which is also in unary form.

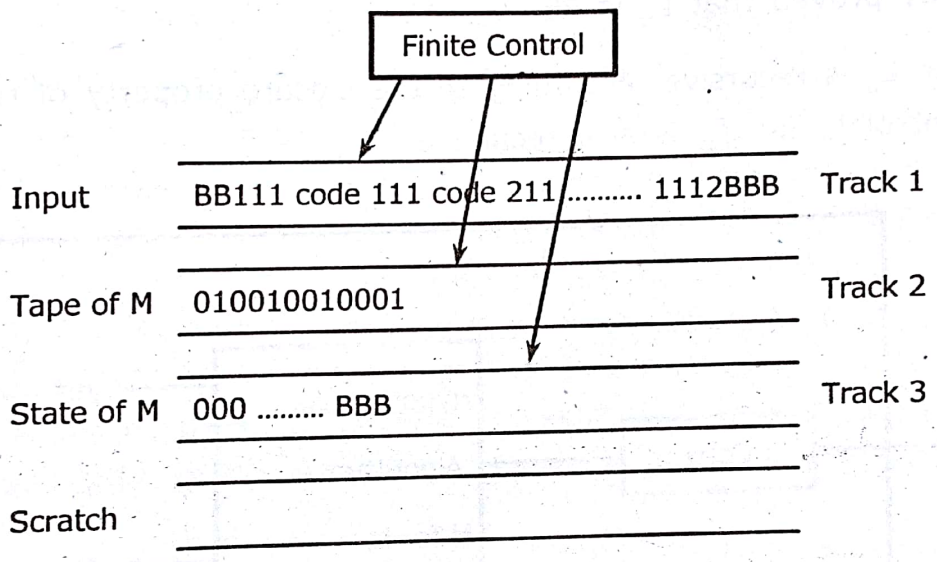



**Fig. 9.3.4** Organization of a Universal Turing Machine

The operation of U are as follows,

(i) First make sure that the code for M is a legitimate code for some turing machine. Otherwise it halts without accepting.

(ii) Initialize the second tape with the input w, in its encoded form. Keep 0 the start state of M on the third tape and move the head of U's second tape of the first stimulated cell.

(iii) If $0^i$ is the current state with $0^j$ the current input symbol appeared on track three and two respectively then U finds the corresponding transition of the form $0^i 10^j 10^k 10^l 10^m$ on track 1 and replaces $0^i$ by $0^k$ and $0^j$ by $0^i$.

(iv) Move the head on tape two to the position corresponding to the value of M.

(v) The Universal Turing Machine U accepts $0^i 10^j 10^k 10^l 10^m$ if M has the transition of the form $\delta(0^i, 0^j) = (0^k, 0^l, 0^m)$. Otherwise M halts without accepting.

Thus U simulates M and accepts w. Thus $L_u$ is recursively enumerable.

oufastupdates.com

## Undecidability of the Universal Language

A problem that is RE but not recursive like $L_u$, is undecidable, because reducing the problem $L_u$ to another problem T can be used to show there is no algorithm to solve T, regardless of whether or not T is RE.

**THEOREM**

**$L_u$ is RE but not recursive.**

**PROOF**

It is already proved that $L_u$ is RE.

Assume $L_u$ is recursive. According to the closure property of recursive sets, the complement of $L_u$ i.e., $\overline{L_u}$ is also recursive.

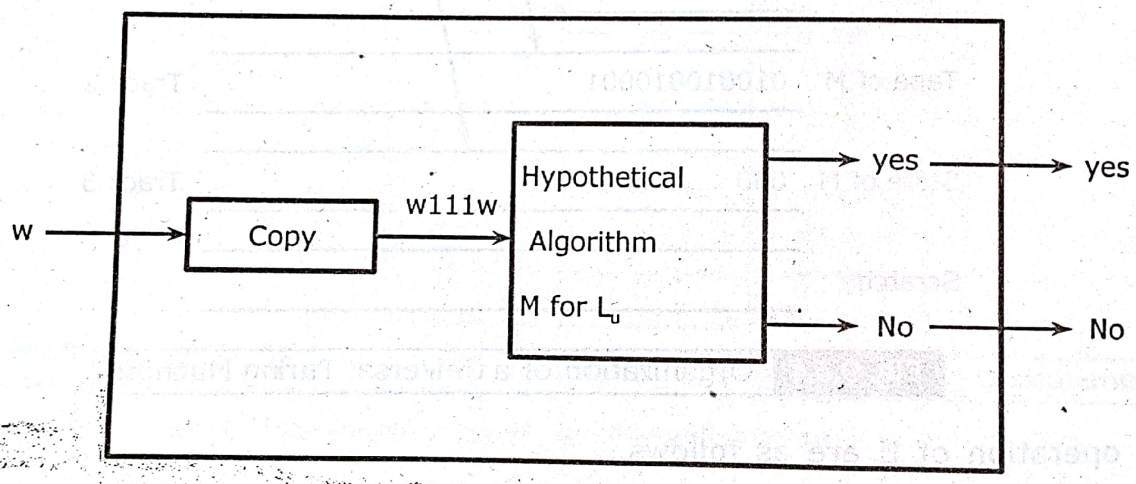



**Fig. 9.3.5** Reduction of $L_d$ to $\overline{L}_u$

Suppose M is an algorithm recognizing $L_u$, $L_d$ can be recognized as follows,

(i) Given string w in (0 + 1) * determined by an easy calculation, the value of i such that $w = w_i$. Integer i in binary is the corresponding code for some TM $M_i$.

(ii) Give the input $<M_i, w_i>$ to algorithm M and accept w if and only if $M_i$ accepts $w_i$.

(iii) So the constructed algorithm also accepts w if and only if $w = w_i$ which is in $L(\overline{M_i})$. This is the algorithm constructed for $L_d$.

But no such algorithm exists and also our assumption of there is an algorithm M for $L_u$ exists is false.

Hence $L_u$ is RE but not recursive.

oufastupdates.com

## Halting Problem

The halting problem is a decision problem which can be informally stated as following, "Given a description of an algorithm and a description of its initial arguments (inputs), determine whether the algorithm, when executed with these arguments, ever halts. The alternative is that the given algorithm runs forever without halting."

Alan Turing proved in 1936 that there is no general method or algorithm which can solve the halting problem. An algorithm may contain loops which may be infinite or finite in length depending on the input and behaviour of the claimed algorithm. The amount of work done in an algorithm usually depends on the input size. Algorithms may consist of various numbers of loops, nested or in sequence. The HP asks the question.

**Example :** Given a program and an input to the program, determine if the program will eventually stop when it is executed that input?

One thing, we can do here to find the solution of HP. Let the program run with the given input and wait for result. If the program stops and we would conclude that problem is solved. But if the program doesn't stop in a reasonable amount of time, we cannot conclude. The question is "how long we can wait....?. the waiting time may be long enough to exhaust the whole life. So, we cannot take it easier as it seems to be. We must have answer, either "Yes" or "NO", and hence some algorithm to decide the exact answer. The importance of the halting problem lies in the fact that it is the first problem to be proved undecidable. Subsequently, many other such problems have been described.

### THEOREM : 1

**HP is undecidable.**

**PROOF**

This proof was devised by Alan Turing in 1936. Initially, we assume that HP is decidable and the algorithm (effective solution) for HP is H. The halting problem solution H takes two inputs.

(i) Description of turing machine M i.e., program P.

(ii) Input I for the program P.

Algorithm H generates an output "YES" if it determines that P stops for input I and it outputs "NO" if it determines that P loops as shown in Fig. 9.3.6.

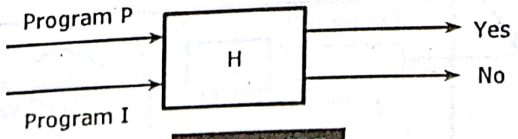



**Fig. 9.3.6**

When an algorithm is coded, it is expressed as a string of characters. Input is also coded into the same format. So, after coding, a program and data have no difference in their format of representation on the storage and so, a program can be treated a data sometime and a data can be treated as a program sometime.

oufastupdates.com

Now algorithm H can be modify to take P as both inputs (the program P and its input I) and H should be able to determine if P will halt on P as shown in Fig. 9.3.7. Here an, algorithm H has to process its program P as program and input.

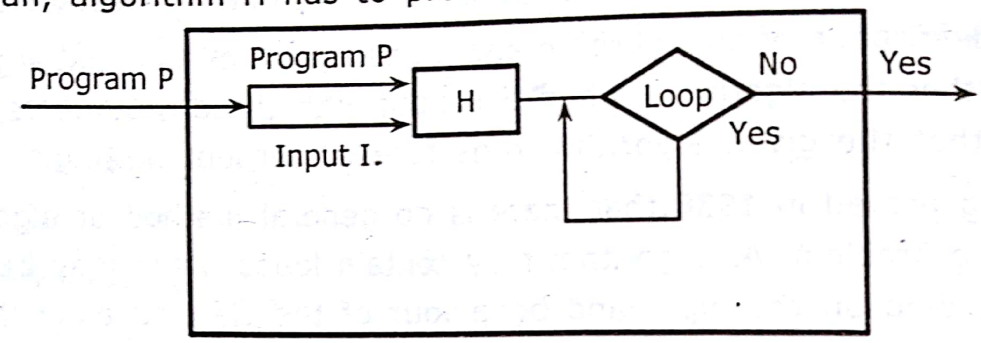



**Fig. 9.3.7**

Let us construct a new, simple algorithm K that takes output of algorithm H as its input and does the following,

(i) If H outputs "NO" then K outputs "Yes" and halts.

(ii) Otherwise if H outputs "Yes" then K loops forever.

It means, K does the opposite what algorithm H does. We define algorithm K as following,

```
Function K( )
{
    if (Function H( ) = "NO"
    {
        return ("YES")
    }
        else
    {
        while(1);                    // Loop for ever
    }
}
                                     // End of the function k
```

Since, algorithm K is a program (Turing machine), so let us use code of algorithm K as the input to itself as shown in Fig. 9.3.8.

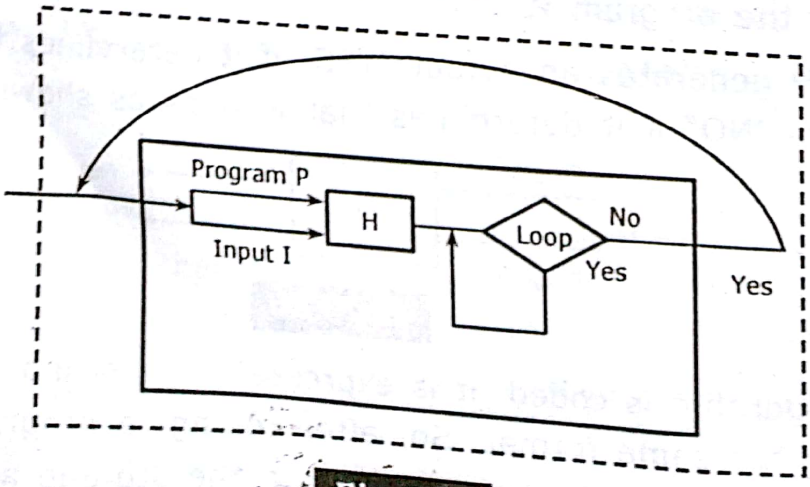



**Fig. 9.3.8**

oufastupdates.com

Now, we analyse the following,

(i) If algorithm H outputs "YES" and tells that algorithm K halts then K itself would loop (that's how we constructed it).

(ii) If algorithm H outputs "NO" and tells that K loops then K outputs "YES" and halts.

Since, in either case algorithm H provides wrong answer for algorithm K. Therefore, H cannot work in all cases and hence cannot provide right solution for all the inputs. This contradicts our assumption made earlier for HP. Hence, HP is undecidable.

THEOREM : 2

**HP of turing machines is undecidable.**

PROOF

HP of turing machine is stated as "To decide whether or not a Turing machine halts for arbitrary input w". We can prove this problem as undecidable by following the similar steps discussed in the Theorem 1.

### 9.3.5.1 Reduction of Problems to Halting Problem of Turing Machine

We reduce a problem X to the HP of turing machine as following,

(1) Assume that we have an effective procedure (Turing machine M or any other kind of algorithm) to solve problem X.

(2) Show how to use the program for X to solve the HP.

(3) Since, HP cannot be solved i.e., undecidable, so conclude that problem X is also undecidable.

EXAMPLE PROBLEM 9.6

**Consider the state entry problem (also known as dead code problem) is to determine whether Turing machine M, when given input w, ever enters state q or not. This is undecidable.**

SOLUTION

We check whether a turing machine M enters state q or not following the steps given below,

(i) Define transition function $\delta(q, a) = \{h, \#)$, for all input symbols 'a'.

(ii) Now, check whether M halts for w or not.

Now, the given problem is same as the HP problem of turing machine that whether M halts for input w or not. Hence, the given problem is undecidable.

oufastupdates.com

## UNDECIDABLE PROBLEMS ABOUT TURING MACHINES

Undecidable problems are all about Turing Machines. From the Rice's theorem, 'Any non trivial property of Turing Machines that depends only on the language the Turing Machine accepts must be undecidable.

### Reduction

**$P_1$ Reduces to $P_2$ :** If there is an algorithm used to convert instance of a problem $P_1$ to instances of a problem $P_2$ with the same answer, then it is said that $P_1$ reduces to $P_2$.

Thus if $P_1$ is not recursive, then $P_2$ cannot be recursive.

If $P_2$ is non RE, then $P_1$ cannot be RE.

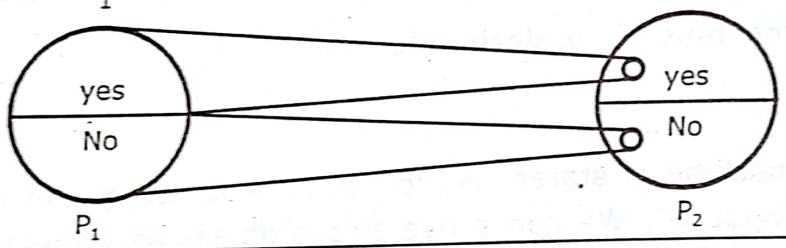



**Fig. 9.4.1** Reductions Turn Positive Instances into Positive and Negative to Negative

So, any instance of $P_1$ that has 'Yes' answer can be turned into an instance of $P_2$ with a 'Yes' answer and every instance $P_1$ of with 'No' answer can be turned into an instance of $P_2$ with 'No' answer.

It is true that only a small fraction $P_1$ of $P_2$ is a target of the reduction.

A *reduction* can be defined as 'A reduction from $P_1$ to $P_2$ is turing machine that takes an instance of $P_1$ written on its tape and halts with an instance of $P_2$ on its tape.

oufastupdates.com

## Rice's Theorem and Properties of the RE Languages

The theorem is named after henry Gordon Rice and is otherwise known as Rice Myhill Shapiro theorem.

A property about turing machines can be represented as the language of all turing machines, encoded as strings, which satisfy that property. The property P is about the language recognized by turing machines if whenever L(M) = L(N) then P contains M if and only if it contains N. The property is non trivial if there is at least one Turing machine that has the property, and at least one Turing machine that hasn't.

THEOREM

**Any non-trivial property about the language recognized by a turing machine is undecidable.**

PROOF

Here we assume that a turing machine that recognizes the empty language does not have the property P. If it does, we just take the complement of P and the undecidability of that complement would immediately imply the undecidability of P. We prove this by the method of contradiction.

oufastupdates.com

In order to get a contradiction, assume that P is decidable. It implies that there is a halting turing machine $M_1$ that recognizes the descriptions of turing machines that satisfy P. Using turing machine $M_1$ we can construct a turing machine $M_2$ that accepts the language.

{(M, w)}|M is the description of a turing machine that accepts the string 'w'}.

Since the latter problem is undecidable, this will show that $M_1$ cannot exist and thus P must be undecidable as well.

The algorithm to perform the reduction takes a pair (M, w) as input and produces a TM $M_i$ the design of $M_1$ is suggested in the Fig. 9.4.4.

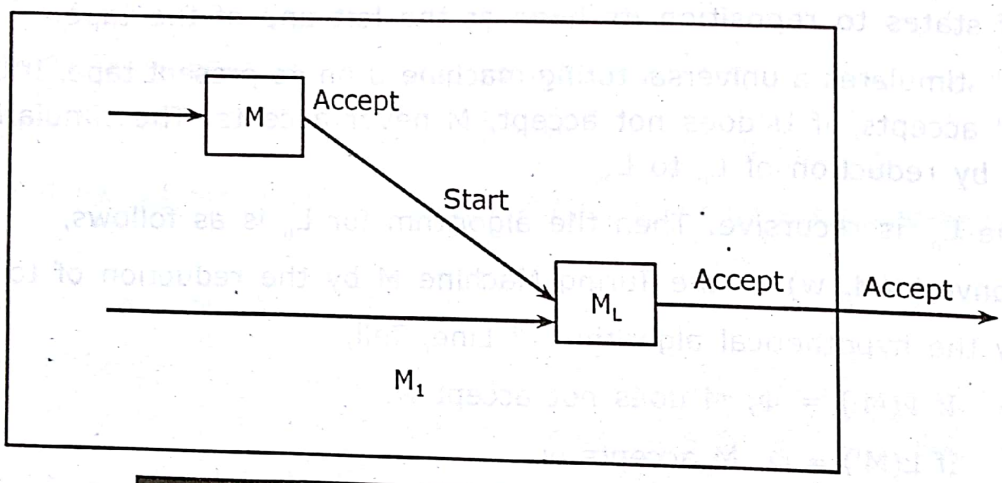



**Fig. 9.4.4** TM Designed to Prove Rice Theorem

Let $M_P$ be a turing machine that satisfies P (as P is non-trivial). Now the turing machine $M_2$ operates as follows,

(i) On input (M, w), create a turing machine C(M, w) as follows,

- On input x, let the turing machine M run on the string w until it accepts. Equivalently if M does not accept, then the turing machine C(M, w) will run forever.
- Next turing machine $M_P$ runs on x. (Accept if and only if $M_P$ does). It is to be noted that turing machine C(M, w) accepts the same language as $M_P$ if M accepts w and C(M, w) accepts the empty language if M does not accept w.

Therefore, if M accepts w the turing machine C(M, w) has the property P, and otherwise it does not.

(ii) The description of C(M, w) is given to turing machine $M_1$. If $M_1$ accepts, then the input (M, w) is accepted otherwise rejected.

**NOTE :** ***It is to be noted that Rice's theorem does not tell anything about properties of machines, only of functions and languages.***

oufastupdates.com

## Problems About Turing Machine Specifications

All problems about turing machines that involve only the language that the TM accepts are undecidable. Some of the undecidable problems of turing machine specification are listed below,

(1) Whether the language accepted by a TM is empty.

(2) Whether the language accepted by a TM is finite.

(3) Whether the language accepted by a TM is a regular language.

(4) Whether the language accepted by a TM is a context free language.

However, Rice's theorem does not imply that everything about a TM is undecidable. For instance, questions that ask about the states of the TM, rather than about the language it accepts, could be decidable.

**Example 1 :** It is decidable whether a TM has five states. The algorithm to decide this question simply looks at the code for the TM and counts the number of states that appear in any of its transitions.

**Example :** It is decidable whether there exists some input such that the TM makes atleast five moves. The algorithm becomes obvious when we remember that if a TM makes five moves, then it does so looking only at the nine cells of its tape surrounding its initial head position. Thus, we may simulate the TM for five moves on any of the finite number of tapes consisting of five or fewer input symbols, preceded and followed by blanks. If any of these simulations fails to reach a halting situation, then we conclude that the TM makes atleast five moves on some input.

## POST'S CORRESPONDENCE PROBLEM (PCP)

In this section, the undecidable problem about turing machine are reduced to undecidable problems about real things. The goal is to prove that problem about strings to be undecidable.

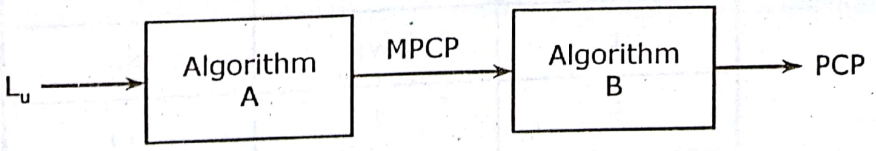



**Fig. 9.5.1** Reductions Proving the Undecidability of PCP

oufastupdates.com

## Definition of Post's Correspondence Problem

An instance of post's correspondence problem (PCP) consists of two list of strings over $\Sigma$ are,

$$A = W_1, W_2, \ldots W_k.$$

$$B = X_1, X_2, \ldots X_K \text{ for some integer } K.$$

The instance of PCP has a *solution* if there is any sequence of integers $i_1, i_2 \ldots i_m$. With $m > 1$ such that $W_{i1}, W_{i2}, W_{i3} \ldots W_{im} = X_1, X_2 \ldots X_k$, is a solution to this instance of PCP.

### EXAMPLE PROBLEM 9.7

**The following instance of PCP has the solution or not whether PCP is undecidable.**

| | List A | List B |
|---|---|---|
| i | $w_i$ | $x_i$ |
| 1 | 11 | 111 |
| 2 | 100 | 001 |
| 3 | 111 | 11 |

**SOLUTION**

Let $\Sigma = \{0, 1\}$ and let the A and B lists of 3 strings has the PCP solution.

For instance, let $m = 3$, $i_1 = 1$, $i_2 = 2$ and $i_3 = 3$ i.e., the solution is the list 1 2 3 we can verify that this list is a solution by concatenating the corresponding strings in order for the two lists, i.e.,

$$w_1\ w_2\ w_3 = x_1\ x_2\ x_3 = 11100111.$$

### EXAMPLE PROBLEM 9.8

**Find the given instance of PCP has solution or not?**

| | List A | List B |
|---|---|---|
| i | $w_i$ | $x_i$ |
| 1 | 1 | 111 |
| 2 | 10111 | 10 |
| 3 | 10 | 0 |

oufastupdates.com

**SOLUTION**

Let $\Sigma = \{0, 1\}$ be a finite set input alphabet and let A and B lists has 3 strings has the solution.

For instance let $m = 1$, $i_1 = 1$, $i_2 = 2$ and $i_3 = 3$ i.e., the solution is the list 2113. We can verify that this list is a solution by concatenating the corresponding strings in order for the two lists i.e.,

$$w_2\ w_1 w_1\ w_3 = x_2\ x_1\ x_1\ x_3 = 101111110.$$

Then the two lists A and B are equal as we get $w_i = x_i$.

Hence (2, 1, 1, 3) is a solution.

## EXAMPLE PROBLEM 9.9

**Find the given instance of PCP has solution or not?**

| | List A | List B |
|---|---|---|
| i | $w_i$ | $x_i$ |
| 1 | 110 | 110110 |
| 2 | 0011 | 00 |
| 3 | 0110 | 110 |

**SOLUTION**

Let $\Sigma = \{0, 1\}$ be a finite set of input alphabets and let A and B lists has 3 strings has the solution.

For instance, let $m = 3$, $i_1 = 2$, $i_2 = 3$ and $i_3 = 1$ i.e., the solution is the list 2 3 1 we can verify that this list is a solution by concatenating the corresponding strings in order for the two lists i.e.,

$$w_2\ w_3\ w_1 = x_2\ x_3\ x_1 = 00110110110.$$

Then the two lists A and B are equal as we get $w_i = x_i$.

Hence (2, 3, 1) is a solution.

oufastupdates.com

**Given an instance of PCP, show that this instance has no solution.**

| | List A | List B |
|---|---|---|
| i | $w_i$ | $x_i$ |
| 1 | 10 | 101 |
| 2 | 011 | 11 |
| 3 | 101 | 011 |

**SOLUTION**

Suppose the instance has a solution $i_1, i_2, \ldots i_m$.

No string beginning with instance of W match with instances of X,

$$W_1 W_3 \neq X_1, X_3$$

$$10101 = 101011$$

Thus, this instance of PCP has no solution.

## EXAMPLE PROBLEM 9.11

**Given an instance of PCP, show that this instance has no solution.**

| | List A | List B |
|---|---|---|
| i | $w_i$ | $x_i$ |
| 1 | 011 | 101 |
| 2 | 11 | 011 |
| 3 | 1101 | 110 |

**SOLUTION**

Suppose that the PCP instance has a solution say $i_1, i_2 \ldots\ldots i_m$ for some $m \geq 1$ we claim $i_1 = 3$. For $i_1 = 1$, then a string beginning with $w_1 = 011$ would have to equal a string that begins with $x_1 = 101$. But that equality is impossible, since the first symbols of these two strings are 0 and 1 respectively similarly it is not possible that $i_1 = 2$, since then a string beginning with $w_2 = 11$ would have to equal a string beginning with $x_2 = 011$.

oufastupdates.com

If $i_1 = 3$, then the two corresponding strings from list A and B would have to begin,

A : 1101

B : 110

Now, let us see what $i_2$ could be,

(i) If $i_2 = 2$, then we have a problem since no string beginning with $w_3\ w_2 = 110111$ can match a string that begins with $x_3\ x_2 = 110011$, they must disagree at the $4^{th}$ position.

(ii) If $i_2 = 3$, we again have a problem because no string that begins with $w_3\ w_2 = 11011101$ can match a string that begins with $x_3\ x_3 = 110110$, they must differ in the fourth position.

(iii) Only $i_2 = 1$ is possible.

If we choose $i_2 = 3$, then corresponding string formed from list of integers $i_1\ i_2$ are,

A : 1101011

B : 110101

There is nothing about these strings that immediately suggests we can extend list 1, 3 to a solution. However, we can argue that it is not possible to do so. The reason is that we are in the same condition we were in after choosing $i_1 = 3$. The string from the list B is same as the string from the A list except that in the B list there is an extra 1 at the end. Thus we are forced to choose $i_3 = 1$, $i_4 = 1$ and so on, to avoid creating mismatch we can never allow the A string to catch up to the B string and the two can never reach a solution.

## The 'Modified' PCP

In order to simplify the reduction of $L_u$ to PCP, an intermediate version of PCP called modified Post's correspondence problem is used.

The modified PCP is the following,

Given lists A and B, of k strings each from $\Sigma^*$

$$A = w_1, w_2, \ldots w_k.$$

$$B = x_1, x_2, \ldots x_k.$$

It has the solution $i_1, i_2, \ldots i_r$ such that,

$$w_1, w_{i1}, \ldots w_{i2}, \ldots\ldots w_{ir} = x_1, x_{i1} \ldots\ldots x_{in}$$

The difference between the MPCP and PCP is that in the MPCP, a solution is required to be stored with the first string on each list.

oufastupdates.com

**Let $\Sigma = \{0, 1\}$. Let A and B be list of strings defined as,**

| | List A | List B |
|---|---|---|
| i | $w_i$ | $x_i$ |
| 1 | 1 | 10 |
| 2 | 110 | 0 |
| 3 | 0 | 11 |

**Find the instance of MPCP.**

**SOLUTION**

MPCP is given by,

$$w_1, w_{i1}, w_{i2} \ldots\ldots w_{ir} = x_1, x_{i1}, \ldots\ldots x_{ir}$$

$$10110 = 10110$$

Where the instance is given 1, 3, 2.

## Construction of PCP from MPCP

Let the lists of MPCP be,

$$A = w_1, w_2, \ldots\ldots w_k.$$

$$B = x_0, x_1, \ldots\ldots x_k$$

Assume * and $ are symbols not present in the alphabet $\Sigma$ of the MPCP instance.

Construct a new PCP instance,

$$C = y_0, y_1, \ldots\ldots\ldots\ldots y_{k-1}$$

$$D = z_0, z_1, \ldots\ldots\ldots\ldots z_{k-1}$$

Designed as follows,

(1) For i = 1, 2, .... k. Let $y_i$ be $w_i$ with a* after each symbol of $w_i$ and let $z_i$ be $x_i$ with a* before each symbol of $x_i$.

(2) $y_0 = *y_1$ and $z_0 = z_i$. The $0^{th}$ pair looks like pair 1, except that there is an extra * at the beginning of the string from the first list. Since we are reducing from MPCP, the $0^{th}$ pair will be the only pair in the PCP instance where both strings begin with the same symbol. So any solution to this PCP instance begin with index 0.

(3) $y_{k-1} = \$$ and $z_{k-1} = *\$$.

oufastupdates.com

## THEOREM

**Post's correspondence problem is undecidable.**

**PROOF**

The proof of this theorem is telling how to reduce $L_u$ to MPCP. It can be proved using the statement.

"M accepts w if and only if the constructed MPCP instance has a solution".

If part : MPCP instance has a solution.

To prove : M accepts w.

Assume a partial solution begins with,

A : #

B : $\#q_0$ w #

State and the tape symbols can only be handled by the pairs of rule (3) and all other tape symbols and # must be handled by pairs from rule (2). After reaching the accepting states, the rules (4) and (5) can be used. Thus, unless M reaches an accepting state, all partial solution have the form,

A : x

B : xy

oufastupdates.com

Wheree,

x = Sequence of ID's of M representing a computation of M on input w possibly followed by # and the beginning of the next ID $\alpha$.

y = Completion of $\alpha$, # and beginning of the ID that follows $\alpha$, upto the point that x ended within $\alpha$ itself.

Thus as long as M does not enter an accepting state, the partial solution is not a solution and also B is longer than A.

Only-if Part : M accepts w.

If w is in L(M), then start with the pair from rule (1) and simulate the computation of M on w. And using the rule (3) pair to copy the state from each ID and simulate one move of M. Then by using rule (2), to copy tape symbols and the marker # as needed. If M reaches an accepting state, then the pairs from rule (4) and rule (5) allow the A string to catch upto the B string and form a solution.

Thus if there is a solution M must enter an accepting state and so M accepts w.

oufastupdates.com

Thanks for downloading

***Good luck***

***~ oufastupdates.com***