

# Theory of Computer Science

## Chapter 1 : Basic Concepts and Finite Automata

**Q. 1 Write note on Chomsky Hierarchy.**

MU - Dec. 2009, Dec. 2012, May 2013, May 2014,  
Dec. 2014, May 2015, Dec. 2016.  
May 2017, Dec. 2017

**Ans. : Chomsky Hierarchy**

A grammar can be classified on the basis of production rules. Chomsky classified grammars into the following types :

1. Type 3 : Regular grammar
2. Type 2 : Context free grammar
3. Type 1 : Context sensitive grammar
4. Type 0 : Unrestricted grammar.

### 1. Type 3 or Regular Grammar

A grammar is called Type 3 or regular grammar if all its productions are of the following forms :

$$A \rightarrow \epsilon$$

$$A \rightarrow a$$

$$A \rightarrow aB$$

$$A \rightarrow Ba$$

Where,  $a \in \Sigma$  and  $A, B \in V$ .

A language generated by Type 3 grammar is known as regular language.

### 2. Type 2 or Context Free Grammar

A grammar is called Type 2 or context free grammar if all its productions are of the following form  $A \rightarrow \alpha$  where  $A \in V$  and  $\alpha \in (V \cup T)^*$ .

$V$  is a set of variables and  $T$  is a set of terminals.

The language generated by a Type 2 grammar is called a context free language, a regular language but not the reverse.

### 3. Type 1 or Context Sensitive Grammar

A grammar is called a Type 1 or context sensitive grammar if all its productions are of the following form.

$$\alpha \rightarrow \beta$$

Where,  $\beta$  is atleast as long as  $\alpha$ .

### 4. Type 0 or Unrestricted Grammar

Productions can be written without any restriction in a unrestricted grammar. If there is production of the  $\alpha \rightarrow \beta$ , then length of  $\alpha$  could be more than length of  $\beta$ .

Every grammar also is a Type 0 grammar.

A Type 2 grammar is also a Type 1 grammar

A Type 3 grammar is also a Type 2 grammar.

**Q. 2 State applications of Finite Automata in brief.**

May 2010

**Ans. :**

### Applications of Finite Automata

Finite automata are used for solving several common types of computer algorithms. Some of them are :

- (i) Design of digital circuit
- (ii) String matching
- (iii) Communication protocols for information exchange.
- (iv) Lexical analysis phase of a compiler.

Finite automata can work as an algorithm for regular language. It can be used for checking whether a string  $w \in L$ , where  $L$  is a regular language.

**Q. 3 What is Finite Automata?**

Dec. 2012

**Ans. :**

### Finite Automata

Finite automata are also called a finite state machine.

A finite state machine is a mathematical model for actual physical process. By considering the possible inputs on which these machines can work, one can analyse their strengths and weaknesses.

Finite automata are used for solving several common types of computer algorithms. Some of them are :

1. Design of digital circuits.
2. String matching.
3. Communication protocols for information exchange.
4. Lexical analyser of a typical compiler.

**Q. 4 Define the term : Unrestricted grammar**

May 2013

**Ans. :**

### Unrestricted grammar

Productions can be written without any restriction in a unrestricted grammar. If there is production of the  $\alpha \rightarrow \beta$ , then length of  $\alpha$  could be more than length of  $\beta$ .

Every grammar also is a Type 0 grammar.

A Type 2 grammar is also a Type 1 grammar

A Type 3 grammar is also a Type 2 grammar.

## Chapter 2 : Finite Automata

**Q. 1 Write short note on Mealy machine.** Dec. 2005

**Ans. :**

### Mealy Machine

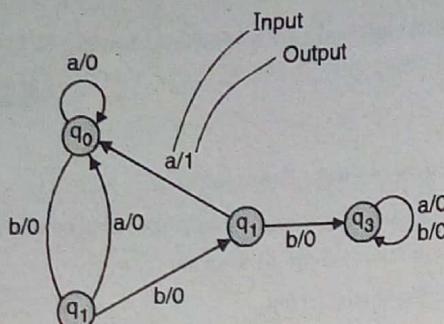


Fig. 2.1 : State diagram of a Mealy machine

State transition function ( $\delta$ ) (or STF) :

	a	b
$\rightarrow q_0$	$q_0$	$q_1$
$q_1$	$q_0$	$q_2$
$q_2$	$q_0$	$q_3$
$q_3$	$q_3$	$q_3$

Fig. 2.2 : State transition function for Mealy machine of Fig. 2.1

Output function ( $\lambda$ ) (or MAF) :

	a	b
$\rightarrow q_0$	0	0
$q_1$	0	0
$q_2$	1	0
$q_3$	0	0

Fig. 2.3 : Output function for mealy machine of Fig. 2.1

State table for both  $\delta$  and  $\lambda$  (both STF and MAF) :

	A	b
$\rightarrow q_0$	$q_0/0$	$q_1/0$
$q_1$	$q_0/0$	$q_2/0$
$q_2$	$q_0/1$	$q_3/0$
$q_3$	$q_3/0$	$q_3/0$

Output → Next state

Fig. 2.4 : State table depicting both transition and output behavior of mealy machine of Fig. 2.1

An arc from state  $q_i$  in a mealy machine is associated with :

1. Input alphabet  $\in \Sigma$
2. An output alphabet  $\in O$ .

An arc marked as 'a/0' in Fig. 2.1 implies that :

1. a is in input
2. 0 is an output.

State transition behavior and output behavior of a mealy machine can be shown separately as in Fig. 2.2 and 2.3; or they can be combined together as in Fig. 2.4.

### Formal Definition of a Mealy Machine

A mealy machine M is defined as :

$$M = (Q, \Sigma, O, \delta, \lambda, q_0)$$

Where,  $Q$  = A finite set of states.

$\Sigma$  = A finite set of input alphabet

$O$  = A finite set of output alphabet

$\delta$  = A transition function  $\Sigma \times Q \rightarrow Q$

$\lambda$  = An output function  $\Sigma \times Q \rightarrow O$

$q_0 = q_0 \in Q$  is an initial state.

**Q. 2 Distinguish between NFA and DFA.**

MU - May 2007, Dec. 2009, May 2011, May 2014,  
May 2015, May 2016, May 2017, Dec. 2017

**Ans. :**

### Difference between NFA and DFA

Parameter	NFA	DFA
Transition	Non-deterministic.	Deterministic
No. of states.	NFA has fewer number of states.	More, if NFA contains $Q$ states then the corresponding DFA will have $\leq 2^Q$ states.
Power	NFA is as powerful as a DFA	DFA is as powerful as an NFA
Design	Easy to design due to non-determinism.	Relatively, more difficult to design as transitions are deterministic.
Acceptance	It is difficult to find whether $w \in L$ as there are several paths. Backtracking is required to explore several parallel paths.	It is easy to find whether $w \in L$ as transitions are deterministic.

**Q. 3 Define DFA.**

May 2010

**Ans. :****Definition of DFA**

A deterministic finite automata is a quintuple.

$$M = (Q, \Sigma, \delta, q_0, F), \text{ where}$$

Q is a set of states.

 $\Sigma$  is a set of alphabet. $q_0 \in Q$  is the initial state, $F \subseteq Q$  is the set of final states, and  $\delta$ , the transition function, is a function from  $Q \times \Sigma$  to  $Q$ .**Q. 4 Obtain a grammar to generate the language**

$$L = \{0^n 1^{2n} \mid n \geq 0\}.$$

May 2010

**Ans. :**

Productions for the required language are as follows.

$$P = \{S \rightarrow 0 S 1 1 \mid \epsilon\}$$

CFG for the above language is  $(\{S\}, \{0, 1\}, P, S)$ **Q. 5 Give deterministic finite automata accepting the following languages over the alphabet  $\{0, 1\}$** 

- (a) Number of 1's is even and number of 0's is even.
- (b) Number of 1's is odd and number of 0's is odd.

May 2010

**Ans. :**

- (a) Number of 1's is even and number of 0's is even.

At any instance of time, we will have following cases for number of 0's and number of 1's seen by the machine.

Situations		State
Number of 0's	Number of 1's	
Even	Even	$q_0$
Even	Odd	$q_1$
Odd	Even	$q_2$
Odd	Odd	$q_3$

An input 0 in state  $q_0$ , will make number of 0's odd.

$$\delta(q_0, 0) \Rightarrow q_2$$

An input 1 in state  $q_0$ , will make number of 1's odd.

$$\delta(q_0, 1) \Rightarrow q_1$$

An input 0 in state  $q_1$ , will make number of 0's odd.

$$\delta(q_1, 0) \Rightarrow q_3$$

An input 1 in state  $q_1$ , will make number of 1's even.

$$\delta(q_1, 1) \Rightarrow q_0$$

An input 0 in state  $q_2$ , will make number of 0's even.

$$\delta(q_2, 0) \Rightarrow q_0$$

An input 1 in state  $q_2$ , will make number of 1's odd.

$$\delta(q_2, 1) \Rightarrow q_3$$

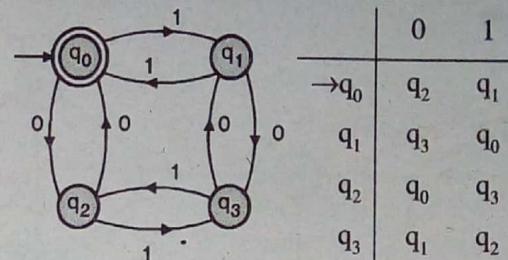
An input 0 in state  $q_3$ , will make number of 0's even.

$$\delta(q_3, 0) \Rightarrow q_1$$

An input 1 in state  $q_3$ , will make number of 1's even.

$$\delta(q_3, 1) \Rightarrow q_2$$

$q_0$  is the starting state. An empty string contains even number of 0's and even number of 1's.  $q_0$  is a final state.  $q_0$  stands for even number of 0's and even number of 1's.



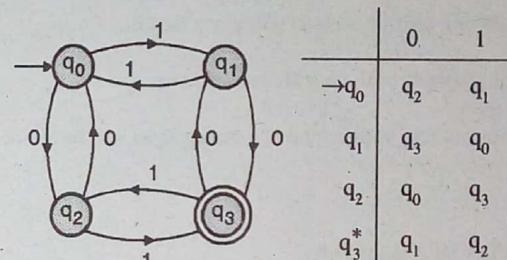
(a) Transition diagram

(b) Transition table

Fig. 2.5 : Final DFA for Q .5(a)

- (b) Number of 1's is odd and number of 0's is odd.

In solution of Q. 5(a), the state  $q_3$  stands for odd number of 0's should be declared as final state.



(c) Transition diagram

(d) Transition table

Fig. 2.5 : Final DFA for for Q .5(b)

**Q. 6 Give the finite automation M accepting  $(a,b)^*(baaa)$ .**

Dec. 2012

**Ans. :**The R.E. =  $(a, b)^*(baaa)$ , represents strings ending in baaa.

The FA is given below

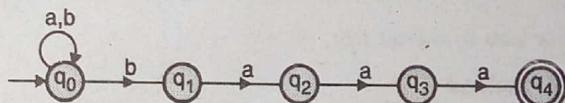


Fig. 2.6

**Q. 7 Give applications of Finite Automata.**

May 2014

**Ans. :****Applications of Finite Automata**

Finite automata are used for solving several common types of computer algorithms. Some of them are :

- (i) Design of digital circuit
- (ii) String matching
- (iii) Communication protocols for information exchange.
- (iv) Lexical analysis phase of a compiler.

Finite automata can work as an algorithm for regular language. It can be used for checking whether a string  $w \in L$ , where  $L$  is a regular language.

**Q. 8 Design a DFA to accept strings over the alphabet set {a, b} that begin with 'aa' but not end with 'aa'.**

Dec. 2014

**Ans. :**

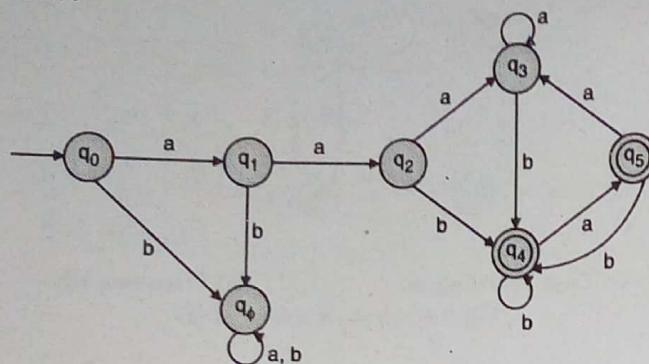


Fig. 2.7

A string not starting with aa will reach the dead state  $q_\phi$ .

A string starting with aa will reach the state  $q_2$ .

A string starting with aa and not ending in aa will be either in  $q_4$  or  $q_5$ .

The DFA is given by,

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_\phi\}, \{a, b\}, \delta, q_0, \{q_4, q_5\})$$

**Q. 9 Design a MOORE and MEALY machine to decrement a binary number.**

**Ans. :**

One can decrement a binary by adding 11...1 (all 1's is 2's complement of 1) to the given number. The addition should start from the least significant digit.

**Mealy machine**

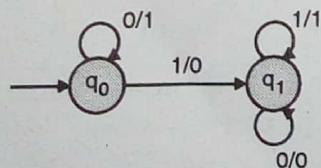


Fig. 2.8

( $q_0$  – Previous carry as 0,  $q_1$  – Previous carry as 1)

i.e., all trailing 0's are written as 1 and the first 1 is written as 0.

**Moore machine :**

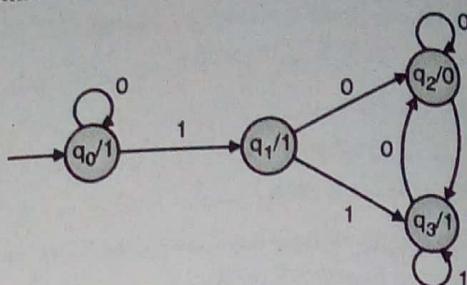


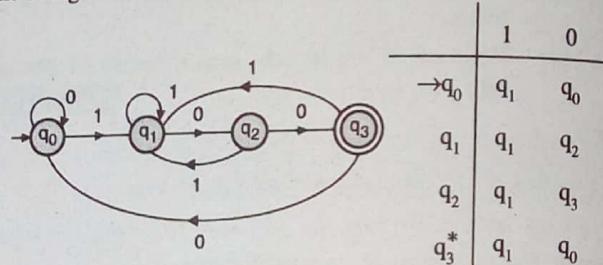
Fig. 2.9

**Q. 10 Design minimized DFA for accepting strings ending with 100 over alphabet (0, 1).** [May 2015]

**Ans. :**

**All strings ending in 100 :**

The substring '100' should be at the end of the string. Transitions from  $q_3$  should be modified to handle the condition that the string has to end in '100'.



(a) State transition diagram

(b) State transition table

Fig. 2.10

**$q_3$  to  $q_1$  on input 1 :**

An input of 1 in  $q_3$  will make the previous four characters as '1001'. Out of the four characters as '1001' only the last character '1' is relevant to '100'.

**$q_3$  to  $q_0$  on input 0 :**

An input of 0 in  $q_3$  will make the previous four characters '1000'. Out of the four characters '1000', nothing is relevant to '100'.

**Q. 11 Design Moore Machine to generate output A if string is ending with abb, B if string ending with aba and C otherwise over alphabet (a, b). and convert it to mealy machine.**

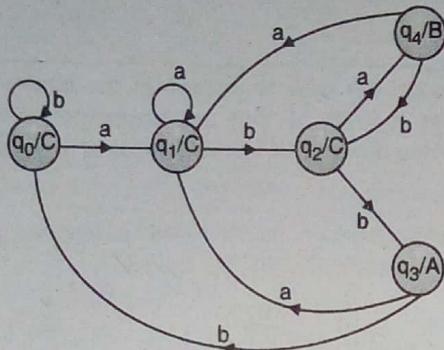
**Ans. :****Design of Moore machine**

Fig. 2.11

**Conversion into Mealy machine :**

**Step 1 :** Construction of a trivial Mealy machine by moving output associated with a state to transition entering into that state.

	a	b
q <sub>0</sub>	q <sub>1</sub> , C	q <sub>0</sub> , C
q <sub>1</sub>	q <sub>1</sub> , C	q <sub>2</sub> , C
q <sub>2</sub>	q <sub>1</sub> , B	q <sub>3</sub> , A
q <sub>3</sub>	q <sub>1</sub> , C	q <sub>0</sub> , C
q <sub>4</sub>	q <sub>1</sub> , C	q <sub>2</sub> , C

**Step 2 : Minimization**

The two states q<sub>1</sub> and q<sub>4</sub> can be merged into a single state, say q<sub>1</sub>.

	a	b
q <sub>0</sub>	q <sub>1</sub> , C	q <sub>0</sub> , C
q <sub>1</sub>	q <sub>1</sub> , C	q <sub>2</sub> , C
q <sub>2</sub>	q <sub>1</sub> , B	q <sub>3</sub> , A
q <sub>3</sub>	q <sub>1</sub> , C	q <sub>0</sub> , C

The two states q<sub>0</sub>, q<sub>3</sub> can be merged into a single state, say q<sub>0</sub>.

	a	b
q <sub>0</sub>	q <sub>1</sub> , C	q <sub>0</sub> , C
q <sub>1</sub>	q <sub>1</sub> , C	q <sub>2</sub> , C
q <sub>2</sub>	q <sub>1</sub> , B	q <sub>0</sub> , A

The final Mealy machine is

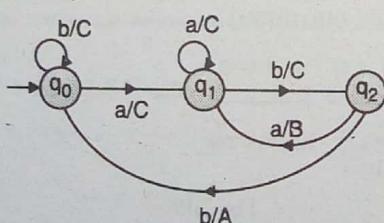


Fig. 2.12

**Q. 12 Convert following  $\epsilon$ -NFA to NFA without  $\epsilon$ .**

Dec. 2015

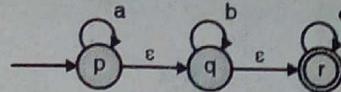


Fig. 2.13

**Ans. :**

To convert  $\epsilon$ -NFA to NFA without  $\epsilon$

**Step 1 :** To remove  $\epsilon$  transition from q state to r state, we do following

- (a) Duplicate transitions of r state on q state
- (b) Since r is the final state, we make q as well as the final state.

**Step 2 :** To remove  $\epsilon$  transition from p state to q state do following :

- (a) Duplicate the transitions of q state on p state
- (b) Since q is a final state we make p as well as the final state.

Thus, the NFA is :

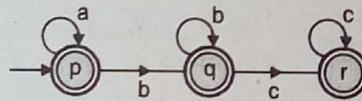


Fig. 2.14

Since all 3 states in the NFA are final states, we can merge all 3 states

$\therefore$  NFA - without  $\epsilon$  is

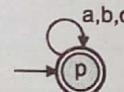


Fig. 2.15

**Q. 13 Design the DFA to accept the language containing all the strings over  $\Sigma = \{a, b, c\}$  that starts and ends with different symbols.****Ans. :**

$$M = \{Q, \Sigma, \delta, q_0, F\}$$

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

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

$$q_0 = \text{initial state}$$

$$F = \{q_3, q_5, q_7\}$$

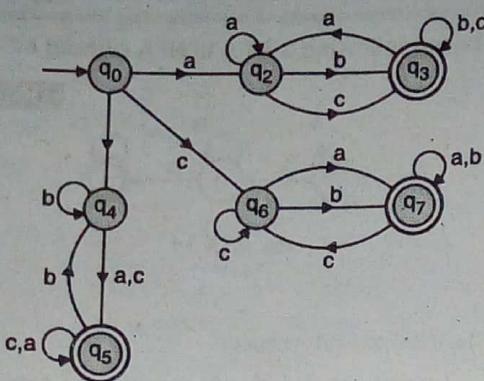


Fig. 2.16

$\delta$  = Transitions are :

$$\begin{aligned}
 \delta(q_0, a) &\Rightarrow q_2 & \delta(q_6, c) &\Rightarrow q_6 \\
 \delta(q_0, b) &\Rightarrow q_4 & \delta(q_3, a) &\Rightarrow q_2 \\
 \delta(q_0, c) &\Rightarrow q_5 & \delta(q_3, b) &\Rightarrow q_3 \\
 \delta(q_2, a) &\Rightarrow q_2 & \delta(q_3, c) &\Rightarrow q_3 \\
 \delta(q_2, b) &\Rightarrow q_3 & \delta(q_5, a) &\Rightarrow q_5 \\
 \delta(q_2, c) &\Rightarrow q_3 & \delta(q_5, b) &\Rightarrow q_4 \\
 \delta(q_4, a) &\Rightarrow q_5 & \delta(q_5, c) &\Rightarrow q_5 \\
 \delta(q_4, b) &\Rightarrow q_4 & \delta(q_7, a) &\Rightarrow q_7 \\
 \delta(q_4, c) &\Rightarrow q_5 & \delta(q_7, b) &\Rightarrow q_7 \\
 \delta(q_6, a) &\Rightarrow q_7 & \delta(q_7, c) &\Rightarrow q_6 \\
 \delta(q_6, b) &\Rightarrow q_7
 \end{aligned}$$

Q. 14 Convert the following grammar into finite automata.

$S \rightarrow aX \mid bY \mid a \mid b$

$X \rightarrow aS \mid bY \mid b$

$Y \rightarrow aX \mid bS$

Ans. :

The above grammar can be converted to FA as follows :

For every non terminating symbol we consider it as a different state

$$M = \{Q, \Sigma, \delta, S, F\}$$

$$Q = \{S, X, Y\}$$

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

$$S = \text{initial state}$$

$$F = \{X, Y\}$$

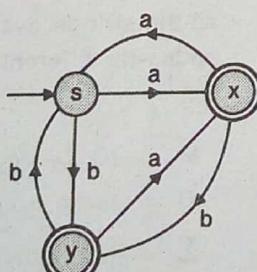


Fig. 2.17

$\delta$  : Transition functions are :

$$\delta(S, a) \Rightarrow X$$

$$\delta(S, b) \Rightarrow Y$$

$$\begin{aligned}
 \delta(X, a) &\Rightarrow S \\
 \delta(X, b) &\Rightarrow Y \\
 \delta(Y, a) &\Rightarrow X \\
 \delta(Y, b) &\Rightarrow S
 \end{aligned}$$

Q. 15 Design the DFA to accept all the binary strings over  $\Sigma = \{0, 1\}$  that are beginning with 1 and having its decimal value multiple of 5. [May 2016]

Ans. :

Running remainder is maintained through the states  $q_0, q_1, q_2, q_3, q_4$ . If the number start with 0, it is rejected.

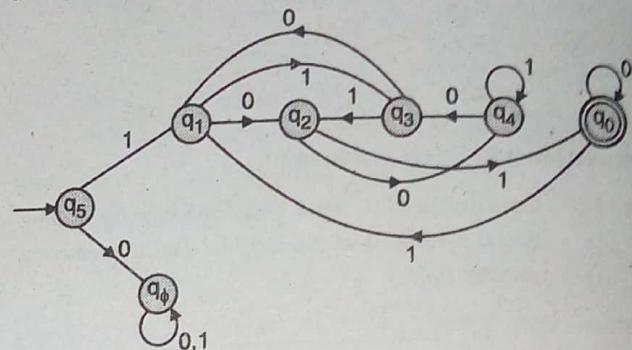


Fig. 2.18

Reminder calculation for finding the next state

State	Binary value of the state	$\delta(q_i, 0)$	$\delta(q_i, 1)$
$q_0$	0	$00 \div 5 = 0 (q_0)$	$01 \div 5 = 1 (q_1)$
$q_1$	1	$10 \div 5 = 2 (q_2)$	$11 \div 5 = 3 (q_3)$
$q_2$	10	$100 \div 5 = 4 (q_4)$	$101 \div 5 = 0 (q_0)$
$q_3$	11	$110 \div 5 = 1 (q_1)$	$111 \div 5 = 2 (q_2)$
$q_4$	100	$1000 \div 5 = 3 (q_3)$	$1001 \div 5 = 4 (q_4)$

The operator + is for reminder.

Q. 16 Design mealy machine to find out 2's complement of a binary number.

Ans. :

2's complement of a binary number

2's complement of a binary number can be found by not changing bits from right end till the first '1' and then complementing remaining bits. For example, the 2's complement of a binary number 0101101000 is calculated as given below :

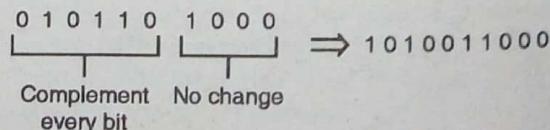


Fig. 2.19

The required mealy machine is given below.

The input is entered from right to left.

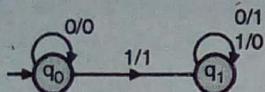


Fig. 2.20

**Q. 17** Convert the following NFA to an equivalent DFA

State	a	b	$\epsilon$
$\rightarrow q_0$	{ $q_0, q_1$ }	{ $q_1$ }	{}
$q_1$	{ $q_2$ }	{ $q_1, q_2$ }	{}
* $q_2$	{ $q_0$ }	{ $q_2$ }	{ $q_1$ }

**Ans. :**

The transition graph of the given NFA is :

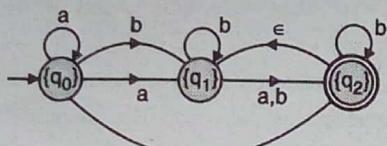


Fig. 2.21

$\epsilon$ -closure of states :

$$q_0 \rightarrow (q_0)$$

$$q_1 \rightarrow (q_1)$$

$$q_2 \rightarrow (q_1, q_2)$$

NFA to DFA using direct method.

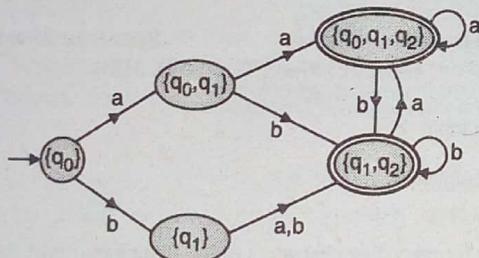


Fig. 2.22

**Q. 18** Design a DFA over an alphabet  $\Sigma = \{a, b\}$  to recognize a language in which every 'a' is followed by 'b'

Dec. 2016

**Ans. :**

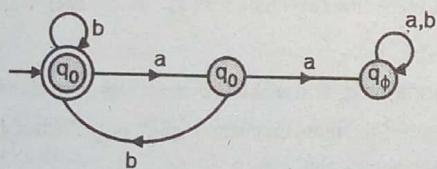


Fig. 2.23

If 'a' is followed by 'a' then the machine enters the failure state  $q_\phi$

A 'b' immediately after 'a' takes the machine to the accepting state  $q_0$

**Q. 19** Design a mealy machine to determine the residue mod 3 of a binary number.

**Ans. :**

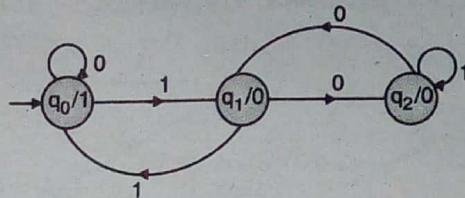


Fig. 2.24

State  $q_0$  is for the running reminder as 0.

State  $q_1$  is for the running reminder as 1.

State  $q_2$  is for the running reminder as 2.

Output 1 indicates divisibility by 3

Output 0 indicate that the number is not divisible by 3.

$\therefore$  Required R.E. =  $(0 + 1(1 + 01)^* 00)^*$

**Q. 20** Convert the following NFA to an equivalent DFA

State	a	b	$\epsilon$
$\rightarrow q_0$	{ $q_0, q_1$ }	$q_1$	{}
$q_1$	{ $q_2$ }	{ $q_1, q_2$ }	{}
* $q_2$	{ $q_0$ }	{ $q_2$ }	{ $q_1$ }

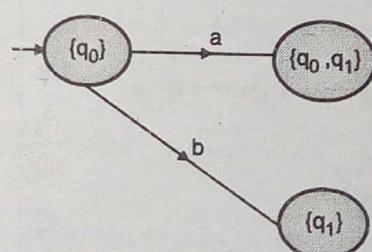
**Ans. :**

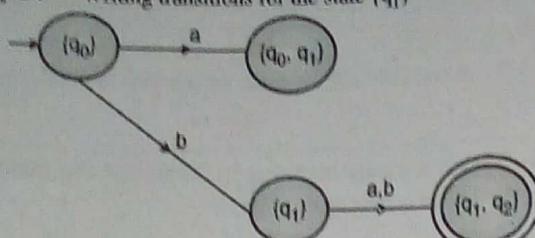
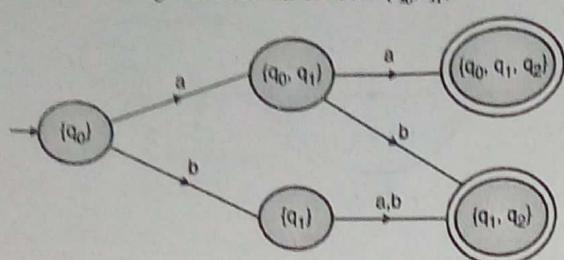
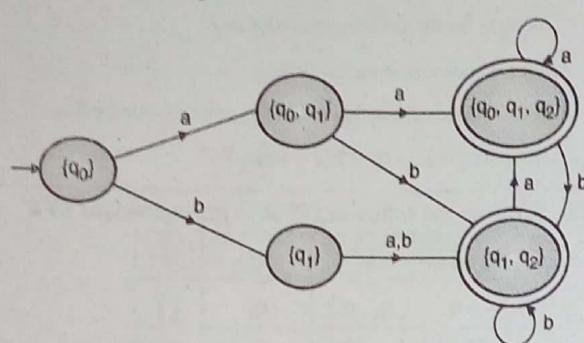
$\epsilon$ -closure of states

State	$\epsilon$ - closure
$q_0$	{ $q_0$ }
$q_1$	{ $q_1$ }
$q_2$	{ $q_1, q_2$ }

Constructing DFA using the direct method

**Step 1 :** Transitions for the state  $\{q_0\}$



**Step 2 :** Writing transitions for the state  $\{q_1\}$ **Step 3 :** Writing transitions for the state  $\{q_0, q_1\}$ **Step 4 :** Writing transitions for the states  $\{q_1, q_2\}$  and  $\{q_0, q_1, q_2\}$ **Q. 21** Draw DFA for the following language over  $\{a, b\}$ :

- All strings starting with abb.
- All strings with abb as a substring i.e., abb anywhere in the string.
- All strings ending in abb.

May'2017

**Ans. :**

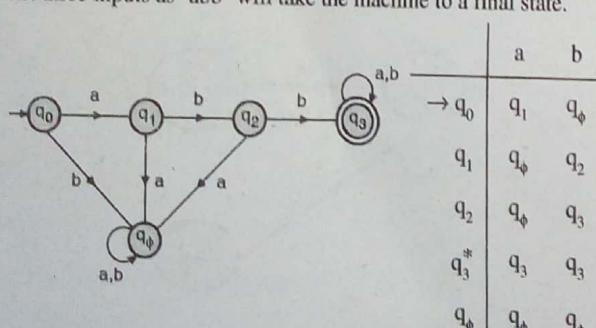
(a) All strings starting with abb

First input as 'b' will take the machine to a failure state.

First two inputs as 'aa' will take the machine to a failure state.

First three inputs as 'aba' will take the machine to a failure state.

First three inputs as 'abb' will take the machine to a final state.

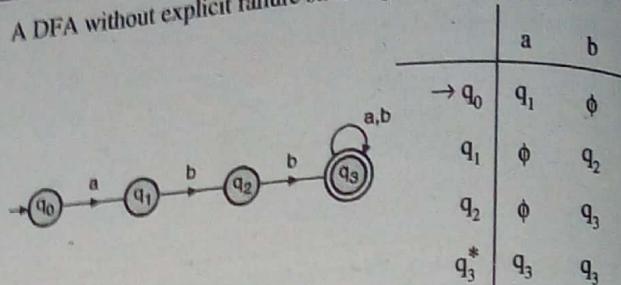


(a) State transition diagram

(b) State transition table

Fig. 2.25 : Final DFA for Q. 21(a)

A DFA without explicit failure state is given in Fig. 2.25(a)

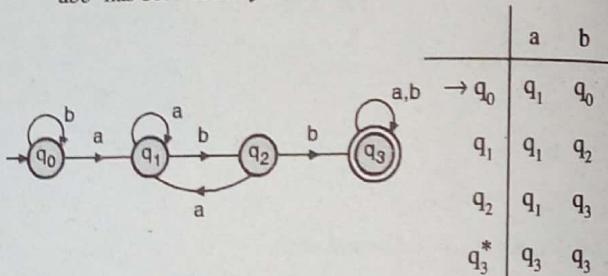


(a) State transition diagram (b) State transition table

Fig. 2.26 : Final DFA for Q. 21(a), without a failure / dead state

(b) All strings with abb as a substring

The machine will have four states:

State  $q_0$  – It is the starting state and indicates that nothing of relevance to complete 'abb' has been seen.State  $q_1$  – preceding character is 'a' and 'bb' is required to complete 'abb'.State  $q_2$  – Preceding characters are 'ab' and 'b' is required to complete 'abb.'State  $q_3$  – Preceding characters are 'abb' and the substring 'abb' has been seen by the machine.

(a) State transition diagram

(b) State transition table

Fig. 2.27 : Final DFA for Q. 21(b)

 $q_0$  to  $q_0$  on input 'b' :

First character in 'abb' is a.

 $q_0$  to  $q_1$  on input 'a' : $q_1$  is for preceding characters as 'a', first character of abb. $q_1$  to  $q_1$  on input 'a' :An input of 'a' in state  $q_1$  will make the preceding two characters as 'aa'. Last 'a' will still constitute the first 'a' of abb. $q_1$  to  $q_2$  on input 'b' : $q_2$  is for preceding two characters as 'ab' of 'abb'. $q_2$  to  $q_1$  on input 'a' :An input 'a' in  $q_2$  will make the preceding three characters as 'aba'. Out of the three characters 'aba', only the last character 'a' is relevant to 'abb'.

$q_2$  to  $q_3$  on input b :

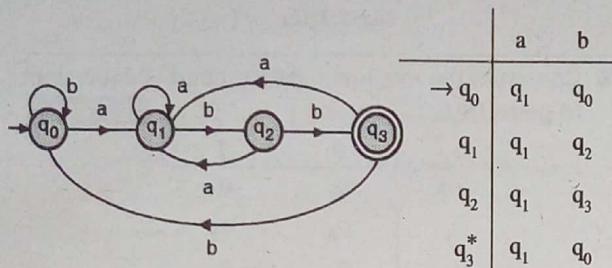
$q_3$  is for preceding three characters as 'abb'.

$q_3$  to  $q_3$  on input a or b :

The substring 'abb' has been seen by the machine and a new input will not change this status.

### (c) All strings ending in abb

As the substring 'abb' should be at the end of the string. Transitions from  $q_3$  should be modified to handle the condition that the string has to end in 'abb'.



(a) State transition diagram

(b) State transition table

Fig. 2.28 : Final DFA for Q. 21(c)

$q_3$  to  $q_1$  on input a :

An input of a in  $q_3$  will make the previous four characters as 'abba'. Out of the four characters as 'abba' only the last character 'a' is relevant to 'abb'.

$q_3$  to  $q_0$  on input b :

An input of b in  $q_3$  will make the previous four characters 'abbb'. Out of the four characters 'abbb', nothing is relevant to 'abb'.

### Q. 22 Design a DFA which can accept a binary number divisible by 3

Or

**Design of a divisibility – by – 3 – tester for a binary number.** [Dec. 2005, May 2014, May 2017]

**Ans. :**

A binary number is divisible by 3, if the remainder when divided by 3 will work out to be zero. We must device a mechanism for finding the final remainder.

We can calculate the running remainder based on previous remainder and the next input.

The running remainder could be :

0 → associated state,  $q_0$

1 → associated state,  $q_1$

2 → associated state,  $q_2$

Starting with the most significant bit, input is taken one bit at a time. Running remainder is calculated after every input. The process of finding the running remainder is being explained with the help of an example.

Number to be divided : 101101.

1	0	1	1	0	1	← Binary number to be divided by 3
1	0					$(1)_2 \text{ MOD } 3 = (1)_2$
1	0	1				$(10)_2 \text{ MOD } 3 = (10)_2$
1	0	1	1			$(101)_2 \text{ MOD } 3 = (10)_2$
1	0	1	1	1		$(101)_2 \text{ MOD } 3 = (10)_2$
1	0	0				$(100)_2 \text{ MOD } 3 = (1)_2$
1	1					$(11)_2 \text{ MOD } 3 = (0)_2$
x	x					

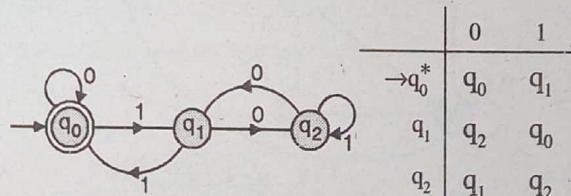
Fig. 2.29

The calculation of next remainder is shown below,

Previous remainder	Next input	Calculation of remainder	Next remainder
0 ( $q_0$ )	0	$00 \% 3 \Rightarrow$	0 ( $q_0$ )
0 ( $q_0$ )	1	$01 \% 3 \Rightarrow$	1 ( $q_1$ )
1 ( $q_1$ )	0	$10 \% 3 \Rightarrow$	10 ( $q_2$ )
1 ( $q_1$ )	1	$11 \% 3 \Rightarrow$	0 ( $q_0$ )
10 ( $q_2$ )	0	$100 \% 3 \Rightarrow$	1 ( $q_1$ )
10 ( $q_2$ )	1	$101 \% 3 \Rightarrow$	10 ( $q_2$ )

↑                      ↑                      ↑

Binary              Binary decimal      Binary



(b) State transition diagram

(c) State transition table

Fig. 2.30 : DFA for Q. 22

### Q. 23 Design a DFA for a mod 5 tester for ternary input.

Dec. 2017

**Ans. :**

A ternary system has three alphabets

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

Base of a ternary number is 3.

The running remainder could be :

- (0)<sub>3</sub> = 0 → associated state,  $q_0$
- (1)<sub>3</sub> = 1 → associated state,  $q_1$
- (2)<sub>3</sub> = 2 → associated state,  $q_2$
- (10)<sub>3</sub> = 3 → associated state,  $q_3$
- (11)<sub>3</sub> = 4 → associated state,  $q_4$
- ↑      ↑

Ternary      Decimal

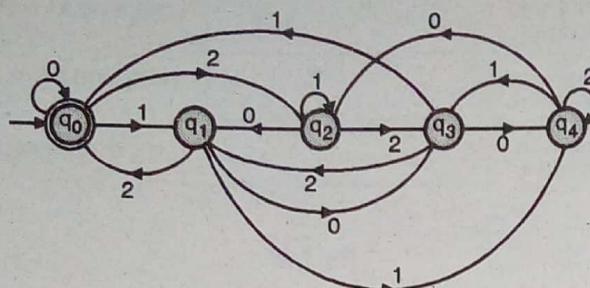


Fig. 2.31

**Q. 24 Design DFA that accepts the following language :**

- Set of all strings with odd number of 1's followed by even number of 0's  $\Sigma = \{0, 1\}$ .
- Set of all strings which begin and end with different letters  $\Sigma = \{x, y, z\}$ .
- Strings ending with 110 or 111.

Dec. 2006, Dec. 2009, Dec. 2010

**Ans. :**

(i)

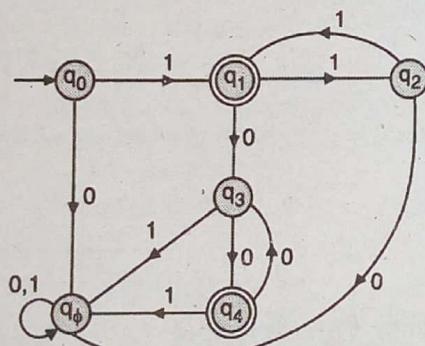


Fig. 2.32(a)

(ii)

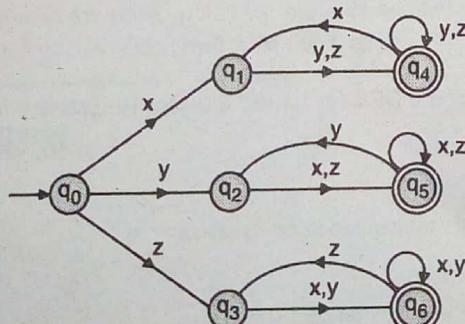


Fig. 2.32(b)

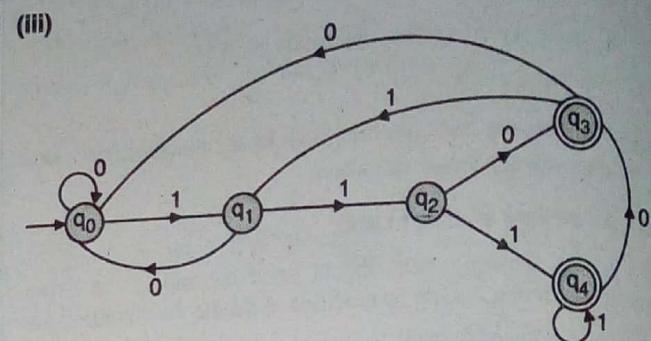


Fig. 2.33(c)

**Q. 25 Construct the minimum state automata equivalent to given DFA.** [May 2011]

	0	1
→ $q_0$	$q_1$	$q_0$
$q_1$	$q_0$	$q_2$
$q_2$	$q_3$	$q_1$
$q_3^*$	$q_3$	$q_0$
$q_4$	$q_3$	$q_5$
$q_5$	$q_6$	$q_4$
$q_6$	$q_5$	$q_6$
$q_7$	$q_6$	$q_3$

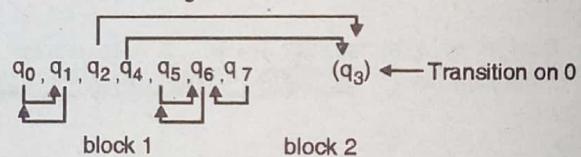
**Ans. :**

**Step 1 :** Finding 0-equivalence partitioning of states by putting final and non-final states into independent block.

$$P_0 = (q_0, q_1, q_2, q_4, q_5, q_6, q_7) \quad (q_3)$$

block 1    block 2

**Step 2 :** Finding 1-equivalence partitioning of states by considering transition on '0' and transition on '1'.



On input 0, block 1 is successor of  $q_0, q_1, q_5, q_6, q_7$ .

On input 0, block 2 is successor of  $q_2, q_4$ .

∴  $q_2, q_4$  are distinguishable from  $q_0, q_1, q_5, q_6, q_7$

$$(q_0, q_1, q_2, q_4, q_5, q_6, q_7) \quad (q_3)$$

block 1    block 2

(q3) ← Transition on 1

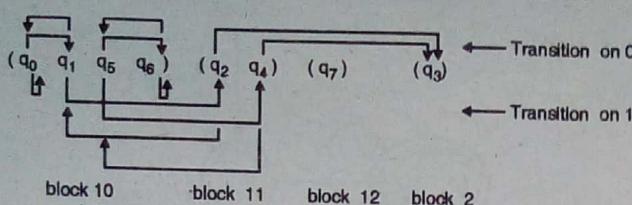
On input 1, block 2 is successor of  $q_7$ .

On input 1, block 1 is successor of  $q_0, q_1, q_2, q_4, q_5, q_6$ .

$q_7$  is distinguishable from  $q_0, q_1, q_2, q_4, q_5, q_6$ .

$$P_1 = (q_0, q_1, q_5, q_6) (q_2, q_4) (q_7)$$

**Step 3 :** Finding 2-equivalence partitioning of states by considering transition on '0' and transition on '1'.



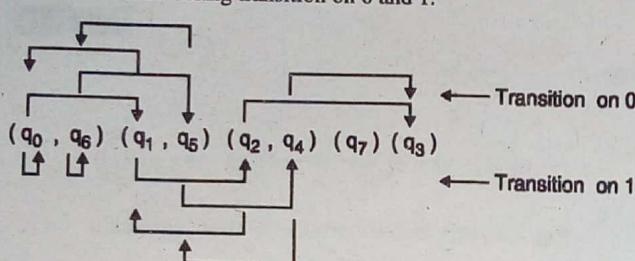
On input 1, block 11 is successor of  $q_1, q_5$ .

On input 1, block 10 is successor of  $q_0, q_6$ .

$q_1, q_5$  is distinguishable from  $q_0, q_6$ .

$$P_2 = (q_0, q_6) (q_1, q_5) (q_2, q_4) (q_7) (q_3)$$

**Step 4 :** Finding 3-equivalence partitioning of states by considering transition on 0 and 1.



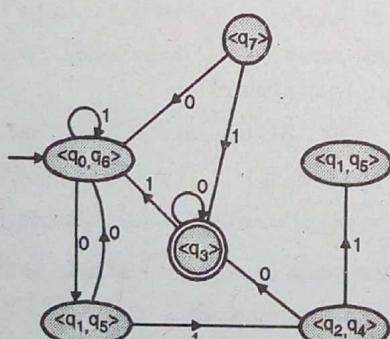
Blocks can not be divided further.

$\therefore P_3 = P_2 = (q_0, q_6) (q_1, q_5) (q_2, q_4) (q_7) (q_3)$  which is final set of blocks of equivalent classes.

**Step 5 :** Construction of minimum state DFA.

	0	1
$\rightarrow(q_0, q_6)$	$(q_1, q_5)$	$(q_0, q_6)$
$(q_1, q_5)$	$(q_0, q_6)$	$(q_2, q_4)$
$(q_2, q_4)$	$(q_3)$	$(q_1, q_5)$
$(q_3)^*$	$(q_3)$	$(q_0, q_6)$
$(q_7)$	$(q_0, q_6)$	$(q_3)$

(a) State transition diagram for minimum-



(b) State transition diagram for minimum-state DFA state DFA

Fig. 2.34

**Q. 26 A language L is accepted by some NFA if and only if it is accepted by some DFA.**

OR

For every NFA, there exists an equivalent DFA.

Dec. 2014

**Ans. :**

**Proof**

Given theorem has two parts :

1. If L is accepted by a DFA  $M_2$ , then L is accepted by some NFA  $M_1$ .
2. If L is accepted by an NFA  $M_1$ , then L is accepted by some DFA  $M_2$ .

First part can be proved trivially. Determinism is a case of non-determinism. Thus a DFA is also an NFA.

Second part of the theorem is proved below :

Construct  $M_2$  from  $M_1$  using subset generation algorithm as explained earlier. We can prove the theorem using induction on the length of  $\omega$ .

**Base case :** Let  $\omega = \epsilon$  with  $|\omega| = 0$ , where  $|\omega|$  is length of  $\omega$ .

Starting state for both NFA and DFA are taken as  $q_0$ . When  $\omega = \epsilon$ , both DFA and NFA will be in  $q_0$ . Hence, the base case is proved.

**Assumption :** Let us assume that both NFA and DFA are equivalent for every string of length n. We must show that the machines  $M_1$  (NFA) and  $M_2$  (DFA) are equivalent for strings of length  $(n + 1)$ . Let  $\omega_{n+1} = \omega_n a$ , where  $\omega_n$  is a string of length n and  $\omega_{n+1}$  is a string of length  $(n + 1)$ . 'a' is an arbitrary alphabet from  $\Sigma$ .

$\delta_2(q_2, \omega_n) = \delta_2(q_0, \omega_n)$ , where  $\delta_2$  is transition function of DFA ( $M_2$ ) and  $\delta$  is transition function of NFA ( $M_1$ ).

If the subset reached by NFA is given by

$$\{p_1, p_2, \dots, p_k\}$$

$$\text{then, } \delta_2(q_0, \omega_{n+1}) = \bigcup_{i=1}^k \delta_2(p_i, a) \quad \dots(i)$$

$$\text{or } \delta_2(\{p_1, p_2, \dots, p_k\}, a) = \bigcup_{i=1}^k \delta_1(p_i, a) \quad \dots(ii)$$

$$\text{also, } \delta_2(q_0, \omega_n) = \{p_1, p_2, \dots, p_k\} \quad \dots(iii)$$

from (i), (ii) and (iii) we get,

$$\begin{aligned} \delta_2(q_0, \omega_{n+1}) &= \delta_2(\delta_2(q_0, \omega_n), a) \\ &= \delta_2(\{p_1, p_2, \dots, p_k\}, a) \\ &= \bigcup_{i=1}^k \delta_1(p_i, a) = \delta_1(q_0, s_{n+1}) \end{aligned}$$

Thus, the result is true for  $|\omega| = n + 1$ , hence it is always true.

**Q. 27** Convert the following NFA to a DFA and informally describe the language it accepts.

Dec. 2011

	0	1
$\rightarrow p$	{p, q}	{p}
q	{r, s}	{t}
r	{p, r}	{t}
s*	$\emptyset$	$\emptyset$
t*	$\emptyset$	$\emptyset$

**Ans. :****Step 1 :** {p} is taken as the first subset.

$$0\text{-Successor of } \{p\} = \delta(\{p\}, 0) = \{p, q\}$$

$$1\text{-Successor of } \{p\} = \delta(\{p\}, 1) = \{p\}$$

**Step 2 :** The new subsets {p, q} is generated. Successors of {p, q} are calculated.

$$\begin{aligned}\delta(\{p, q\}, 0) &= \delta(p, 0) \cup \delta(q, 0) \\ &= \{p, q\} \cup \{r, s\} \\ &= \{p, q, r, s\}\end{aligned}$$

$$\begin{aligned}\delta(\{p, q\}, 1) &= \delta(p, 1) \cup \delta(q, 1) = \{p\} \cup \{t\} \\ &= \{p, t\}\end{aligned}$$

**Step 3 :** Two new subsets {p, q, r, s} and {p, t} are generated. Their successors are calculated.

$$\begin{aligned}\delta(\{p, q, r, s\}, 0) &= \delta(p, 0) \cup \delta(q, 0) \cup \delta(r, 0) \cup \delta(s, 0) \\ &= \{p, q\} \cup \{r, s\} \cup \{p, r\} \cup \emptyset \\ &= \{p, q, r, s\}\end{aligned}$$

$$\begin{aligned}\delta(\{p, q, r, s\}, 1) &= \delta(p, 1) \cup \delta(q, 1) \cup \delta(r, 1) \cup \delta(s, 1) \\ &= \{q\} \cup \{t\} \cup \{r\} \cup \emptyset \\ &= \{p, t\}\end{aligned}$$

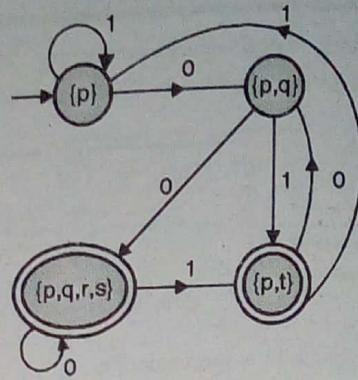
$$\begin{aligned}\delta(\{p, t\}, 0) &= \delta(p, 0) \cup \delta(t, 0) \\ &= \{p, q\} \cup \emptyset = \{p, q\}\end{aligned}$$

$$\begin{aligned}\delta(\{p, t\}, 1) &= \delta(p, 1) \cup \delta(t, 1) \\ &= \{p\} \cup \emptyset = \{p\}\end{aligned}$$

No, new subset is generated. Every subset containing either s or t is marked as a final state.

**Informal Description:** Strings over {0, 1} with second digit from the end is 0.

	0	1
$\rightarrow \{p\}$	{p, q}	{p}
{p, q}	{p, q, r, s}	{p, t}
{p, q, r, s}* {p, t}* (a)	{p, q, r, s}	{p, t}
	{p, q}	{p}



(b) State diagram

Fig. 2.35 : Final DFA for Q. 27

**Q. 28** Construct a NFA that accepts a set of all strings over {a, b} ending in aba. Use this NFA to construct DFA accepting the same set of strings.

May 2014

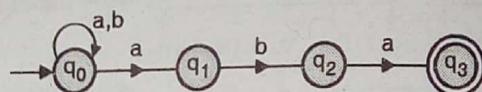
**Ans. :**

Fig. 2.36 (a) : Non-deterministic finite automata

Non-determinism should be utilized to full extent while designing an NFA. A string of length n, ending in aba can be recognized by the NFA given in Fig. 2.36(a). First n-3 characters can be absorbed by the state  $q_0$  by making a guess. On guessing the last three characters as aba, the machine can make a transition from  $q_0$  to  $q_3$ .

NFA to DFA conversion :

**Step 1 :**  $\{q_0\}$  is taken as first subset

$$a\text{-successor of } \{q_0\} = \delta(q_0, a) = \{q_0, q_1\}$$

$$b\text{-successor of } \{q_0\} = \delta(q_0, b) = \{q_0\}$$

**Step 2 :** A new subset  $\{q_0, q_1\}$  is generated. Successors of  $\{q_0, q_1\}$  are calculated.

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

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

**Step 3 :** A new subset  $\{q_0, q_2\}$  is generated. Successors of  $\{q_0, q_2\}$  are calculated.

$$\begin{aligned}\delta(\{q_0, q_2\}, a) &= \delta(q_0, a) \cup \delta(q_2, a) = \{q_0, q_1\} \cup \{q_3\} \\ &= \{q_0, q_1, q_3\}\end{aligned}$$

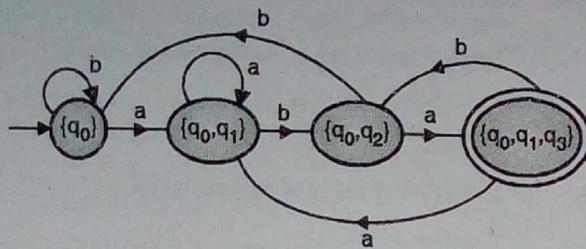
$$\delta(\{q_0, q_2\}, b) = \delta(q_0, b) \cup \delta(q_2, b) = \{q_0\} \cup \emptyset = \{q_0\}$$

**Step 4 :** A new subset  $\{q_0, q_1, q_3\}$  is generated. Successors of  $\{q_0, q_1, q_3\}$  are calculated.

$$\begin{aligned}\delta(\{q_0, q_1, q_3\}, a) &= \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_3, a) \\ &= \{q_0, q_1\} \cup \emptyset \cup \emptyset = \{q_0, q_1\}\end{aligned}$$

$$\begin{aligned}\delta(\{q_0, q_1, q_3\}, b) &= \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_3, b) \\ &= \{q_0\} \cup \{q_2\} \cup \emptyset = \{q_0, q_2\}\end{aligned}$$

No, new subset is generated. Every subset containing  $q_3$  is marked as a final state.



(b) State diagram of the DFA

	a	b
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}^*$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

(c) State table of the DFA

Fig. 2.36

**Q. 29** Give Mealy and Moore machine for the following : From input  $\Sigma^*$ , where  $\Sigma = \{0, 1, 2\}$  print the residue modulo 5 of the input treated as ternary (base 3).

May 2006, Dec. 2015

Ans. :

## (a) Mealy machine

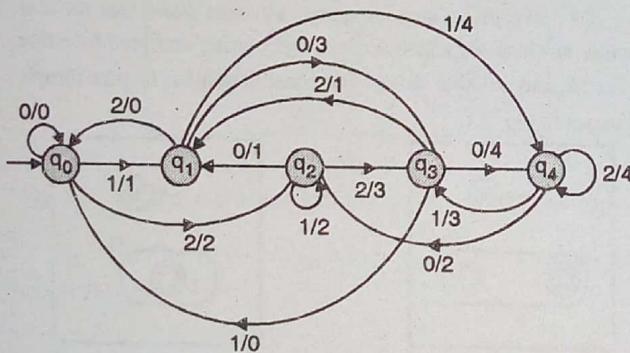


Fig. 2.37(a) : Mealy machine

Meaning of various states is :

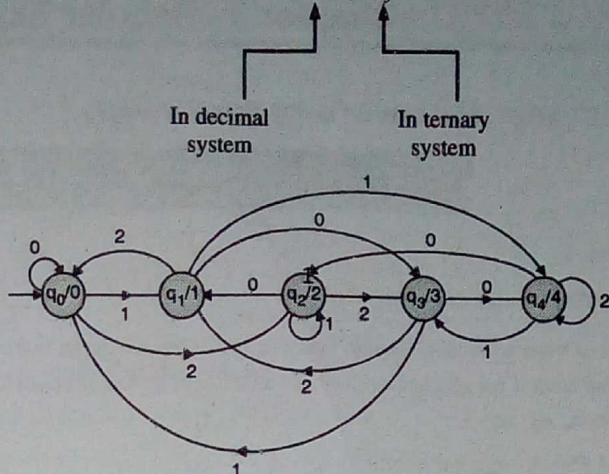
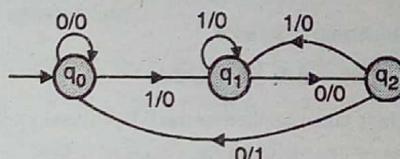
 $q_0$  – Running remainder is 0 $q_1$  – Running remainder is 1 $q_2$  – Running remainder is 2. $q_3$  – Running remainder is 3 =  $(10)_3$  $q_4$  – Running remainder is 4 =  $(11)_3$ 

Fig. 2.37(b) : Moore machine

**Q. 30** Design a mealy machine for a binary input sequence such that if the sequence ends with 100 the output is 1 otherwise output is 0.

Dec. 2006, May 2008, Dec. 2008

Ans. :



(a) State diagram

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

(b) State table

Fig. 2.38

Meaning of various states :

 $q_0$  – start state $q_1$  – previous symbol is 1 $q_2$  – preceding two symbols are 10A transition from  $q_2$  to  $q_0$  will make the preceding three symbol as 100 and hence the output 1.

## Chapter 3 : Regular Expressions and Languages

**Q. 1 Write short note on Myhill-Nerode theorem.**

Dec. 2005, May 2006, Dec. 2006, May 2007  
May 2008, Dec. 2008, Dec. 2012, May 2013

**Ans. :**

### Myhill-Nerode theorem

Given a language L, two strings x and y are said to be in the same class if for all possible strings z either both xz and yz are in L or both are not.

The Myhill-Nerode theorem says :

1. A language L divides the set of all possible strings into mutually exclusive classes.
2. If L is regular, the number of classes created by L is finite.
3. If the number of classes L creates is finite, then L is regular.

In finite automata, each state can be thought of as creating a class of strings. Two strings are said to be in the same class if they both trace a path from starting state  $q_0$  to some state  $q_i$  (say).

Number of strings is infinite.

Number of states in an FA is finite.

Many strings when applied to the FA will end up in the same state. Each state of FA can stand for a class of strings.

**Q. 2 Show that**

$$(1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1) = 0^*1 \\ (0 + 10^*1)^*$$

May 2006

**Ans. :**

$$\begin{aligned} \text{L.H.S.} &= (1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1) \\ &= (1 + 00^*1)[\epsilon + (0 + 10^*1)^*(0 + 10^*1)] \\ &= (1 + 00^*1)(0 + 10^*1)^* \\ &= [(\epsilon + 00^*)1](0 + 10^*1)^* = 0^*1(0 + 10^*1)^* \\ &= \text{R.H.S.} \end{aligned}$$

**Q. 3 Prove  $L = \{(ab)^n a^k : n > k, k \geq 0\}$  is not regular.**

May 2006

**Ans. :**

**Step 1 :** Let us assume that L is regular and L is accepted by an FA with n states.

**Step 2 :** Let us choose a string

$$\omega = (ab)^{n+1} a^n$$

$$|\omega| = 2(n+1) + n = 3n + 2 \geq n$$

Let us write  $\omega$  as xyz, with

$$|y| > 0 \quad \text{and} \quad |xy| \leq n$$

The string xy will contain a maximum of n symbols from  $(ab)^n$ .

**Step 3 :** In the string  $xy^i z$  with  $i = 0$ , at least one 'a' or atleast one 'b' will be erased from  $(ab)^{n+1}$  of  $(ab)^n a^n$ . This will lead to one of the following situations :

1. Number of a's in  $(ab)^n$  is equal to number of a's in  $a^k$  of  $(ab)^n a^k$ .

2.  $xy^0 z$  will not be of the form  $(ab)^n a^n$ .

Therefore,  $xy^0 z \notin L$ .

Hence, this is proved by contradiction.

**Q. 4 Write short notes on closure properties of regular language.** Dec. 2006, May 2013, Dec. 2014

**Ans. :**

### Closure properties of regular language

If an operation on regular languages generates a regular language then we say that the class of regular languages is closed under the above operation. Some of the important closure properties for regular languages are given below.

- |                           |                 |
|---------------------------|-----------------|
| 1. Union                  | 2. Difference   |
| 3. Concatenation          | 4. Intersection |
| 5. Complementation        | 6. Kleene star  |
| 7. Transpose or reversal. |                 |

### 1. Regular Language is Closed under Union

Let  $M_1 = (S, \Sigma, \delta_1, s_0, F)$  and

$M_2 = (Q, \Sigma, \delta_2, q_0, G)$  be two given automata.

To prove the closure property; we must show that there is another machine  $M_3$  which accepts every string accepted by either  $M_1$  or  $M_2$  and no other string. The construction  $M_3$  is quite simple as shown in Fig. 3.1.

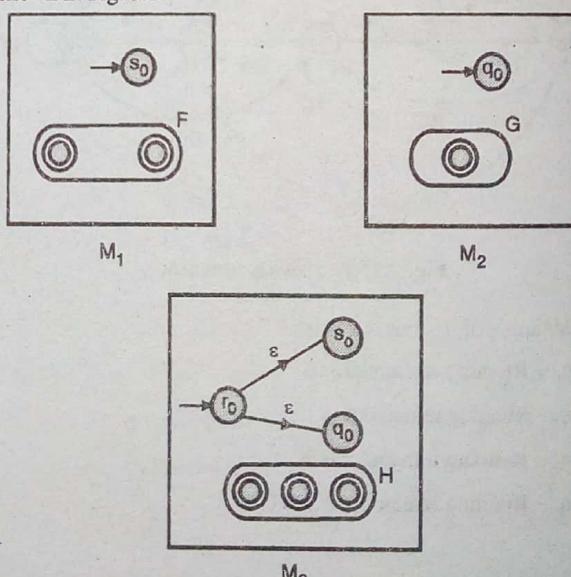


Fig. 3.1 :  $M_3$  is constructed such the  $L(M_3) = L(M_1) \cup L(M_2)$

Machine  $M_3$  is constructed to accept  $L(M_1) \cup L(M_2)$ .

$M_3 = (R, \Sigma, \delta_3, r_0, H)$  where  $r_0$  is a new start state. Two  $\epsilon$ -moves, one from  $r_0$  to  $s_0$  and another from  $r_0$  to  $q_0$  are added.

$$R = S \cup Q \cup \{r_0\}$$

$$H = F \cup G$$

$$\delta_3 = \delta_1 \cup \delta_2 \cup \{(r_0, \epsilon, s_0), (r_0, \epsilon, q_0)\}$$

Machine  $M_3$  can non-deterministically choose either  $M_1$  or  $M_2$ . Therefore,

$$L(M_3) = L(M_1) \cup L(M_2)$$

## 2. Regular Language is Closed under Concatenation

Let  $M_1 = (S, \Sigma, \delta_1, s_0, F)$

and  $M_2 = (Q, \Sigma, \delta_2, q_0, G)$  be two given automata.

To prove that closure property under concatenation, we must show that there is another machine  $M_3$  such that  $L(M_3) = L(M_1) \cdot L(M_2)$ . The construction of  $M_3$  is shown in Fig. 3.2.

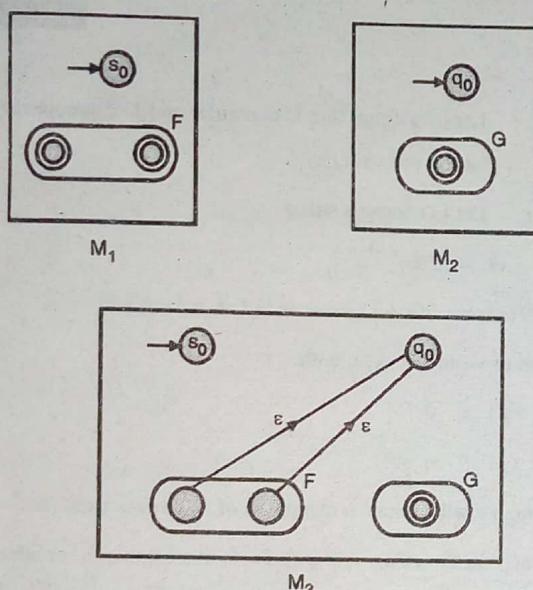


Fig. 3.2 :  $M_3$  is constructed such that  $L(M_3) = L(M_1) \cdot L(M_2)$

$M_3$  is constructed by adding  $\epsilon$ -move from every final state of  $M_1$  to start state of  $M_2$ .

Machine  $M_3$  is given by :

$$M_3 = (R, \Sigma, \delta_3, s_0, G) \text{ where}$$

$$\delta_3 = \delta_1 \cup \delta_2 \cup \{\epsilon\text{-move from every final state of } M_1 \text{ to start state of } M_2\}$$

Machine  $M_3$  recognizes  $L(M_1) \cdot L(M_2)$  by going non-deterministically from the final state of  $M_1$  to start state of  $M_2$ .

## 3. Regular Language is Closed under Kleene Star

Let  $M_1 = (Q, \Sigma, \delta, q_0, F)$  be the given automata. We can construct a non-deterministic finite automata  $M_2$  such that  $L(M_2) = L(M_1)^*$ . The construction of  $M_2$  from  $M_1$  is shown in Fig. 3.3.

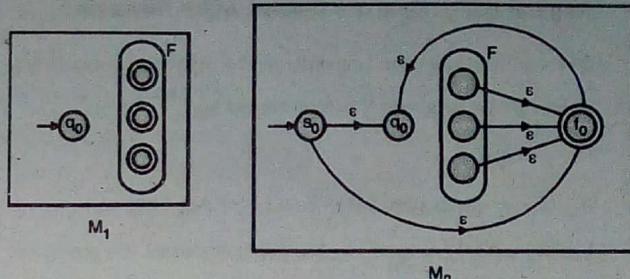


Fig. 3.3 :  $M_2$  is constructed such that  $L(M_2) = L(M_1)^*$

$M_2$  is constructed as given below :

- A new start state  $s_0$  is added with an  $\epsilon$ -move from  $s_0$  to  $q_0$ .
- A new final state  $f_0$  is added with  $\epsilon$ -moves from every state of  $F$  to  $f_0$ . An  $\epsilon$ -move is added from  $s_0$  to  $f_0$  as  $\epsilon$  is a member of  $L(M_1)^*$ .

$$\text{Machine } M_2 = (Q \cup \{s_0, f_0\}, \Sigma, \delta, s_0, \{f_0\})$$

Machine can accept a string  $\in L(M_1)$  and resume back from the start state  $q_0$  through the  $\epsilon$ -move from  $f_0$  to  $q_0$ . Thus accepting  $L(M_1)^*$ .

## 4. Regular Language is Closed under Complementation

Let  $M = (Q, \Sigma, \delta, q_0, F)$  be the given automata. To prove the closure property under complementation, we must show that there is another machine  $\bar{M}$  which accepts  $L(\bar{M})$  where

$$\begin{array}{ccc} L(\bar{M}) & = & L(\bar{M}) \\ | & & | \\ \text{Given} & & \text{Machine after} \\ \text{machine} & & \text{complementation} \end{array}$$

If  $M$  is a deterministic finite automata then  $\bar{M}$  can be constructed by interchanging final and non final states of  $M$ .

$$\therefore \bar{M} = (Q, \Sigma, \delta, q_0, Q - F)$$

## 5. Regular Language is Closed under Intersection

If  $L_1$  and  $L_2$  are two regular languages, then

$$\begin{aligned} L_1 \cap L_2 &= ((L_1 \cap L_2)')' = (\bar{L}_1 \cup \bar{L}_2)' \\ &= \Sigma^* - [(\Sigma^* - L_1) \cup (\Sigma^* - L_2)] \end{aligned}$$

Closeness under intersection follows directly from closeness under union and complementation.

## 6. Regular Languages are Closed under Difference

Let  $L_1$  and  $L_2$  are two regular languages. The difference  $L_1 - L_2$  is the set of strings that are in language  $L_1$  but not in  $L_2$ . Construction of a composite automata for  $L(M_1) - L(M_2)$  is explained in Chapter 2. Thus regular languages are closed under difference.

### 7. Regular Languages are Closed under Reversal

Reversal of a language  $L$  is obtained by reversing every string in  $L$ . Reversal of a language  $L$  is represented by  $L^R$ .

For example,

if  $L = \{aab, abb, aaa\}$ , then  $L^R = \{baa, bba, aaa\}$

Let  $M_1 = (Q, \Sigma, \delta, q_0, F)$  be the given automata. To prove the closure property under reversal, we must show that there is another machine  $M_2$  which accepts  $L(M_1)^R$ .

$$\text{or } L(M_2) = L(M_1)^R$$

$M_2$  can be constructed from  $M_1$  by :

1. By reversing every transition in  $M_1$ .
2. Start state of  $M_1$  is made the only final state.
3. A new start state  $s_0$  is added with  $\epsilon$ -move to every final state of  $M_1$ .

### Q. 5 Design a NFA to accept $(a + b)^* aba$ convert it to a reduced DFA.

May 2007

Ans. :

$(a + b)^* aba$

RE to NFA

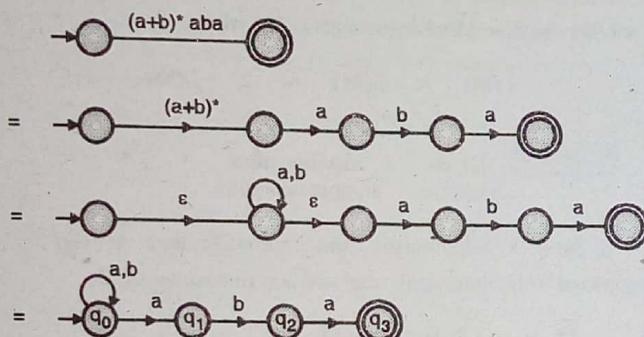


Fig. 3.4 : RE to NFA

NFA to DFA

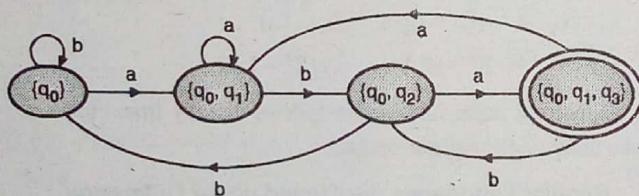


Fig. 3.5 : NFA to DFA

### Q. 6 Write RE for the following languages

- (i) The set of all strings over  $\{0, 1\}$  without length two.
- (ii)  $L = \{a^n b^m \mid (n+m) \text{ is even}\}$

- (iii)  $L = \{\omega \in (a, b)^* \mid (\text{number of } a's \text{ in } \omega) \bmod 3 = 0\}$
- (iv)  $L = \{a^n b^m \mid n \geq 4, m \leq 3\}$

May 2006, Dec. 2007, May 2008

Ans. :

- (i) The set of all strings over  $\{0, 1\}$  without length two.  
 $\epsilon + (0+1) + (0+1)(0+1)(0+1)^*$
- (ii)  $L = \{a^n b^m \mid (n+m) \text{ is even}\}$   
 $((aa)^*ab + bb)(bb)^*$
- (iii)  $L = \{\omega \in (a, b)^* \mid (\text{number of } a's \text{ in } \omega) \bmod 3 = 0\}$   
 $(b + ab^*ab^*)^*$
- (iv)  $L = \{a^n b^m \mid n \geq 4, m \leq 3\}$   
 $aaaa^*[e + b + bb + bbb]$

### Q. 7 Prove $L = \{(ab)^n a^k \mid n > k, k \geq 0\}$ is not regular.

May 2008

Ans. :

Step 1 : Let us assume that  $L$  is regular and  $L$  is accepted by an FA with  $n$  states.

Step 2 : Let us choose a string

$$\omega = (ab)^{n+1} a^n$$

$$|\omega| = 2(n+1) + n = 3n + 2 \geq n$$

Let us write  $\omega$  as  $xyz$ , with

$$|y| > 0$$

$$\text{and } |xy| \leq n$$

The string  $xy$  will contain a maximum of  $n$  symbols from  $(ab)^n$ .

Step 3 : In the string  $xy^i z$  with  $i = 0$ , at least one 'a' or atleast one 'b' will be erased from  $(ab)^{n+1}$  of  $(ab)^{n+1} a^n$ . This will lead to one of the following situations :

1. Number of a's in  $(ab)^n$  is equal to number of a's in  $a^k$  of  $(ab)^n a^k$ .
2.  $xy^0 z$  will not be of the form  $(ab)^n a^k$ .

Therefore,  $xy^0 z \in L$ .

Hence, this is proved by contradiction.

### Q. 8 Construct a NFA for the RE $(01^* + 1)$ and convert it to DFA.

Dec. 2008

Ans. :

$$(01^* + 1)$$

RE to NFA

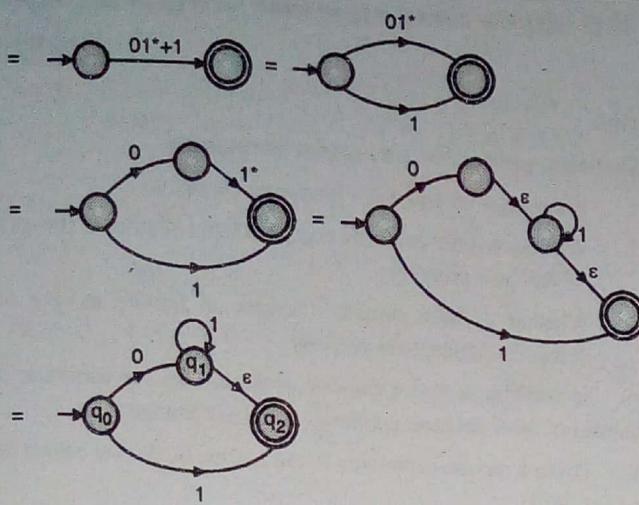


Fig. 3.6(a) : RE to NFA

NFA to DFA

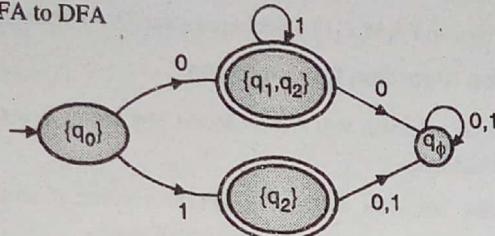


Fig. 3.6(b) : NFA to DFA

**Q. 9 Construct an NFA with  $\epsilon$ -moves for the RE  $10(0 + 01 + 0110)^*$**

[May 2009]

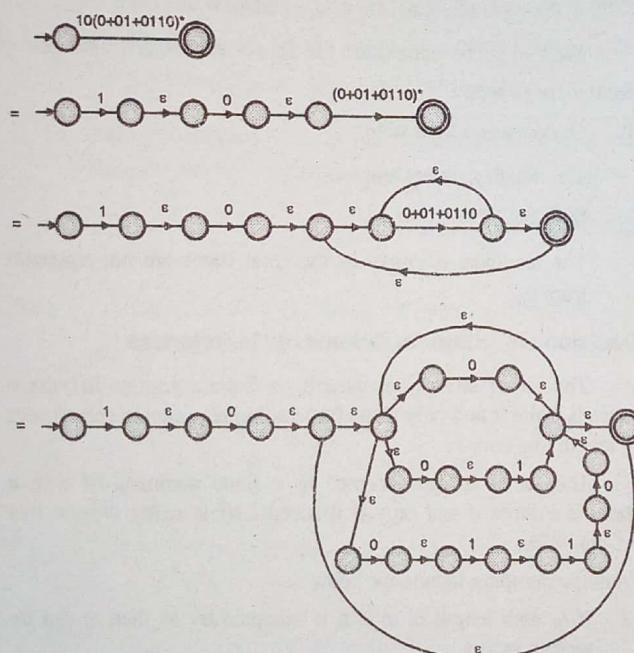
**Ans. :**

Fig. 3.7

**Q. 10 State the pumping lemma for regular language.**

**Ans. :**

### Pumping lemma for regular language

Pumping lemma gives a necessary condition for an input string to belong to a regular set.

Pumping lemma does not give sufficient condition for a language to be regular.

Pumping lemma should not be used to establish that a given language is regular.

Pumping lemma should be used to establish that a given language is not regular.

The pumping lemma uses the pigeonhole principle which states that if  $n$  pigeons are placed into less than  $n$  holes, some holes have to have more than one pigeon in it. Similarly, a string of length  $\geq n$  when recognized by a FA with  $n$  states will see some states repeating.

### Definition of Pumping Lemma

Let  $L$  be a regular language and  $M = (Q, \Sigma, \delta, q_0, F)$  be a finite automata with  $n$ -states. Language  $L$  is accepted by  $m$ . Let  $\omega \in L$  and  $| \omega | \geq n$ , then  $\omega$  can be written as  $xyz$ , where

- (i)  $| y | > 0$
- (ii)  $| xy | \leq n$
- (iii)  $xy^i z \in L$  for all  $i \geq 0$  here  $y^i$  denotes that  $y$  is repeated or pumped  $i$  times.

### Interpretation of Pumping Lemma

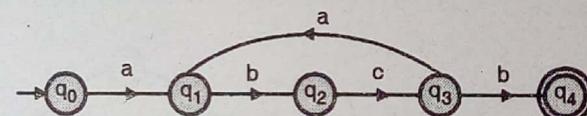


Fig. 3.8 : FA considered for interpretation of pumping lemma

Let us consider the FA of Fig. 3.8

No. of states = 5 ( $q_0$  to  $q_4$ )

Let us take a string  $\omega$  with  $| \omega | \geq 5$ , recognized by the FA.

$\omega = abcabcb$

To recognize the string  $\omega = abcabcb$ , the machine will transit through various states as shown in Fig. 3.6.2.

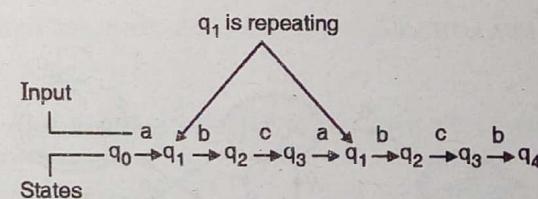


Fig. 3.9 : Transitions of FA on input abcabcb

As the input abcabcb takes the machine through the loop  $q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_1$ , this loop can repeat any number of times. In terms of abcabcb, we can say that if abcabcb is accepted by FA

then every string in  $a(bca)^*bcb$  will be accepted by the FA of Fig. 3.8. The portion bca is input during the loop.

$$q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4$$

Thus, if abcabcb is accepted by the FA then abcabcb can be written as xyz, with

$$x = a$$

$$y = bca$$

$$z = bcb$$

Length of abcabcb is  $\geq n$

$xy^i z$  for every  $i \geq 0$  or  $a(bca)^i bcb$  for every  $i \geq 0$  will be accepted by the FA of Fig. 3.8.

### Q. 11 Construct NFA from $(0+1)^*(00+11)$ and convert into minimized DFA.

Dec. 2009

**Ans. :**

$$(0+1)^*(00+11)$$

RE to NFA

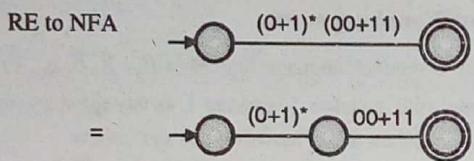


Fig. 3.10 : RE to NFA

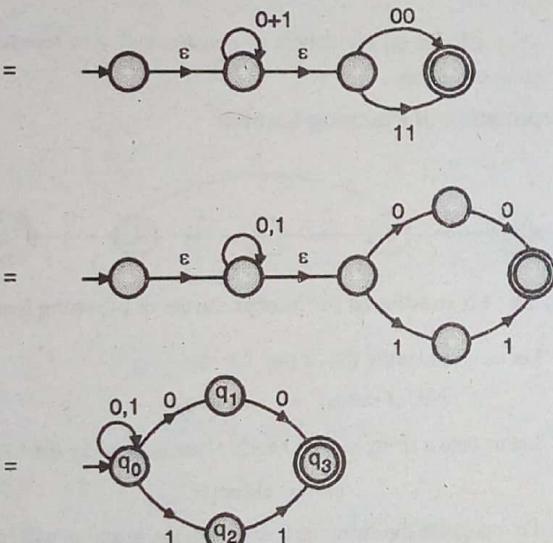


Fig. 3.10(a) : RE to NFA

NFA to DFA

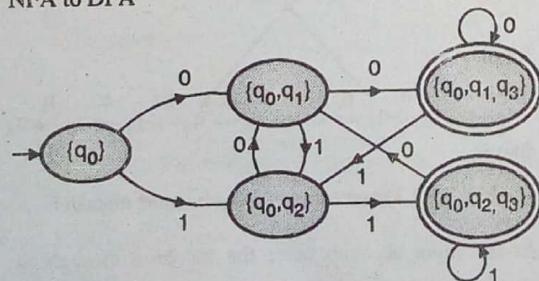


Fig. 3.10(b) : NFA to DFA

### Q. 12 Explain decision properties for regular languages.

Dec. 2009

**Ans. :**

#### Decision properties for regular languages

1. Is a regular set empty? – Emptiness property.
2. Whether a finite automata accepts a finite number of strings? – Finiteness property.
3. Whether a finite automata accepts an infinite number of strings? – Infiniteness property.

In addition to above decision problems, we can formulate a number of other decision problems. Some of them are :

1. Given a regular expression R and a string  $\omega$ , does  $\omega$  belong to  $L(R)$ ?
2. Given two FAs  $M_1$  and  $M_2$ , is  $L(M_1) = L(M_2)$ ?
3. Given two FAs  $M_1$  and  $M_2$ , is  $L(M_1)$  subset of  $L(M_2)$ ?
4. Given an FA M, is M a minimum state FA accepting  $L(M)$ ?

#### Decision Algorithm for emptiness :

Finite automata will fail to accept any string if it does not have a final state.

Finite automata will fail to accept a string if none of its accepting states is reachable from the initial state.

We can determine the emptiness of language accepted by an FA by calculating  $Q_k$ , the set of states that can be reached from  $q_0$  by using strings of length k or less.

$$Q_k = \begin{cases} \{q_0\} & \text{if } k = 0 \\ \{Q_{k-1} \cup \{\delta(q, a) \mid q \in Q_{k-1} \text{ and } a \in \Sigma\}} & \text{if } k > 0 \end{cases}$$

We can go on computing the  $Q_k$  for each  $k \geq 0$  until one of the two cases arise :

1.  $Q_k$  contains a final state.  
The language is not empty.
2.  $Q_k = Q_{k-1}$   
The language is empty as the final states are not reachable from  $q_0$ .

#### Decision algorithm for finiteness / infiniteness :

The set of strings accepted by a finite automata M with n states is finite if and only if the finite automata accepts only strings of length less than n.

The set of strings accepted by a finite automata M with n states is infinite if and only if it accepts some string  $\omega$  such that  $n \leq |\omega| < 2n$ .

From the pumping lemma we know :

1. If  $\omega$  with length of  $\omega \geq n$  is accepted by M then  $\omega$  can be written as xyz.
2. For every,  $i xy^i z$  will be accepted by M.

We can always design an algorithm to generate all strings over  $\Sigma$  with length between n and  $2n$ .

If any of these strings is accepted by M then  $L(M)$  is infinite else  $L(M)$  is finite.

**Q. 13 Using pumping lemma for regular sets, prove that the language  $L = \{\omega\omega^R \mid \omega \in \{0, 1\}^*\}$  is not regular.**

May 2010

**Ans. :**

**Step 1 :** Let us assume that L is regular and L is accepted by a FA with n states.

**Step 2 :** Let us choose a string

$$\omega = \underbrace{a^n b}_{\omega} \underbrace{ba^n}_{\omega^R} \leftarrow \text{from } \omega\omega^R$$

$$|\omega| = 2n + 2 \geq n$$

Let us write  $\omega$  as xyz with

$$|y| > 0 \quad \text{and} \quad |xy| \leq n$$

Since  $|xy| \leq n$ , x must be of the form  $a^s$ .

Since  $|xy| \leq n$ , y must be of the form  $a^r \mid r > 0$ .

Now,

$$\omega = a^n bba^n = \underbrace{a^s}_{x} \underbrace{a^r}_{y} \underbrace{a^{n-s-r} bba^n}_{z}$$

**Step 3 :** Let us check whether  $xy^i z$  for  $i = 2$  belongs to L.

$$xy^2 z = a^s a^{2r} a^{n-s-r} bba^n = a^{n+r} bba^n$$

Since  $r > 0$ ,  $a^{n+r} bba^n$  is not of the form  $\omega\omega^R$  as the strings starts with  $(n+r)$  a's but ends in  $(n)$  a's.

Therefore,  $xy^2 z \notin L$ . Hence by contradiction, we can say that the given language is not regular.

**Q. 14 Using pumping lemma for regular sets. Prove that the language  $L = \{\omega\omega \mid \omega \in \{0, 1\}^*\}$  is not regular.**

Dec. 2006, Dec. 2010

**Ans. :**

**Step 1 :** Let us assume that the given language is regular and L is accepted by a FA with a n states.

**Step 2 :** Let us choose a string

$$\omega = \underbrace{a^n b}_{\omega} \underbrace{a^n b}_{\omega} \leftarrow \text{from } \omega\omega$$

$$|\omega| = 2n + 2 \geq n$$

Let us write  $\omega$  as xyz with

$$|y| > 0$$

$$\text{and } |xy| \leq n$$

Since  $|xy| \leq n$ , x must be of the form  $a^s$ .

Since  $|xy| \leq n$ , y must be of the form  $a^r \mid r > 0$ .

Now,  $\omega = a^n b a^n = \underbrace{a^s}_{x} \underbrace{a^r}_{y} \underbrace{a^{n-s-r} b a^n}_{z}$

**Step 3 :** Let us check whether  $xy^i z$  for  $i = 2$  belongs to L.

$$xy^2 z = a^s a^{2r} a^{n-s-r} b a^n = a^{n+r} b a^n$$

Since  $r > 0$ ,  $a^{n+r} b a^n$  is not of the form  $\omega\omega^R$  as the number of a's in the first half is  $n+r$  and in the second half is n.

Therefore,  $xy^2 z \notin L$ . Hence by contradiction, the given language is not regular.

**Q. 15 Show that the language  $L = \{a^n b a^n \mid n > 0\}$  is not regular.**

Dec. 2009, Dec. 2011

**Ans. :**

**Step 1 :** Let us assume that L is regular and L is accepted by an FA with n states.

**Step 2 :** Let us choose a string

$$\omega = a^n b a^n$$

$$|\omega| = 2n + 1 \geq n$$

Let us write  $\omega$  as xyz, with

$$|y| > 0$$

$$\text{and } |xy| \leq n$$

Since,  $|xy| \leq n$ , y must be of the form  $a^r \mid r > 0$

Since,  $|xy| \leq n$ , x must be of the form  $a^s$ .

Now,  $a^n b a^n$  can be written as :

$$a^s a^{n-s-r} b a^n$$

**Step 3 :** Let us check whether  $xy^i z$  for  $i = 0$  belongs to L.

$$xy^0 z = a^s (a^r)^0 a^{n-s-r} b a^n = a^{n-r} b a^n$$

Since,  $r > 0$  the string  $a^{n-r} b a^n \notin L$ .

Hence by contradiction we can say that the given language is not regular.

**Q. 16 Write short note on application areas of R.E.**

Dec. 2012

**Ans. :**

#### Application areas of Regular Expression

##### 1. R.E. in Unix

The UNIX regular expression lets us specify a group of characters using a pair of square brackets [ ]. The rules for character classes are :

1. [ab] Stand for a + b
2. [0 - 9] Stand for a digit from 0 to 9
3. [A - Z] Stands for an upper-case letter

4. **[a - z]** Stands for a lower-case letter  
 5. **[0 - 9A-Za - z]** Stands for a letter or a digit.

The **grep** utility in UNIX, scans a file for the occurrence of a pattern and displays those lines in which the given pattern is found.

For example :

\$ grep president emp.txt

It will list those lines from the file emp.txt which has the pattern "president". The pattern in grep command can be specified using regular expression.

6. \* matches zero or more occurrences of previous character.
7. ● matches a single character.
8. [^ pqr] Matches a single character which is not a p, q or r.
9. ^ pat Matches pattern pat at the beginning of a line
10. pat \$ Matches pattern at end of line.

### Example

- (a) The regular expression [aA] g [ar] [ar] wal stands for either "Agarwal" or 'agrawal'.
- (b) g\* stands for zero or more occurrences of g.
- (c) \$grep "A . \* thakur" emp.txt will look for a pattern starting with A. and ending with thakur in the file emp.txt.

### 2. Lexical Analysis

Lexical analysis is an important phase of a compiler. The lexical analyser scans the source program and converts it into a stream of tokens. A token is a string of consecutive symbol defining an entity.

For example a C statement  $x = y + z$  has the following tokens :

- x – An identifier
- = – Assignment operator
- y – An identifier
- + – Arithmetic operator +
- z – An identifier

Keywords, identifiers and operators are common examples of tokens.

The UNIX utility **lex** can be used for writing of a lexical analysis program. Input to **lex** is a set of regular expressions for each type of token and output of **lex** is a C program for lexical analysis.

**Q. 17 Design a DFA corresponding to the regular expression.  $(a + b)^* aba (a + b)^*$**  May 2013

**Ans. :**

The language associated with the R.E.  $(a + b)^* aba (a + b)^*$  = strings with "aba" as substring.

DFA for strings with aba as substring.

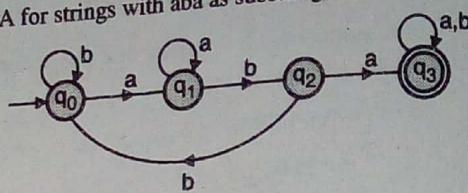
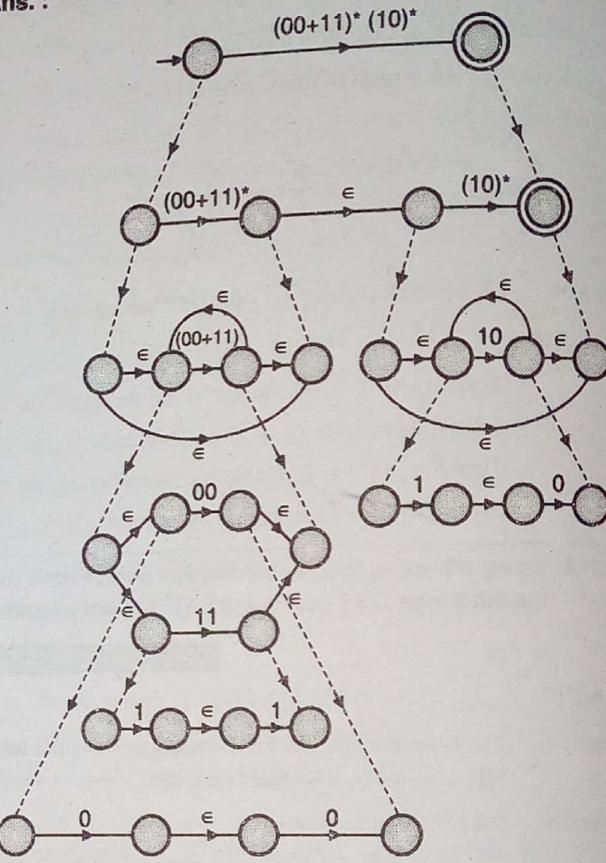


Fig. 3.11

**Q. 18 Construct an NFA with epsilon transition for the following RE.  $(00 + 11)^* (10)^*$**  May 2014

**Ans. :**



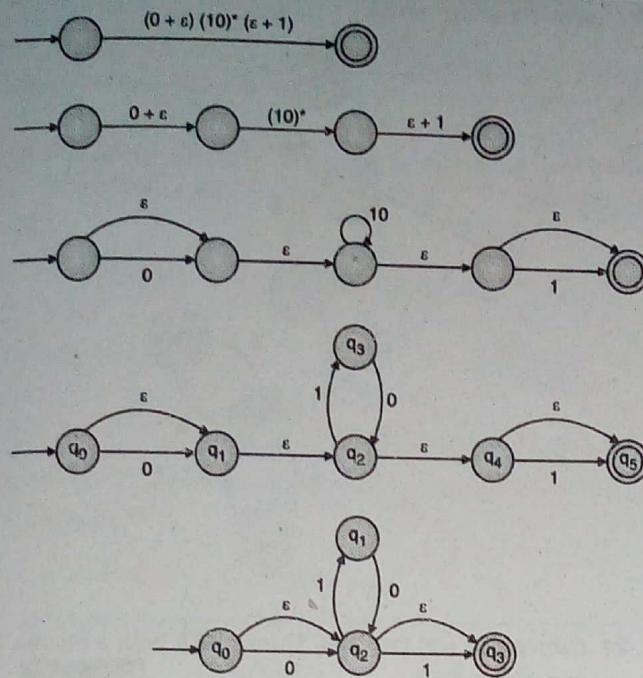


Fig. 3.13

(Note : States have been removed.)

**Step 2 :**  $\epsilon$ -NFA to DFA

$\epsilon$ -closure of states

$$\begin{aligned} q_0 &\rightarrow \{q_0, q_2, q_3\}, & q_1 &\rightarrow \{q_1\} \\ q_2 &\rightarrow \{q_2, q_3\}, & q_3 &\rightarrow \{q_3\} \end{aligned}$$

The DFA using the direct method is given below.

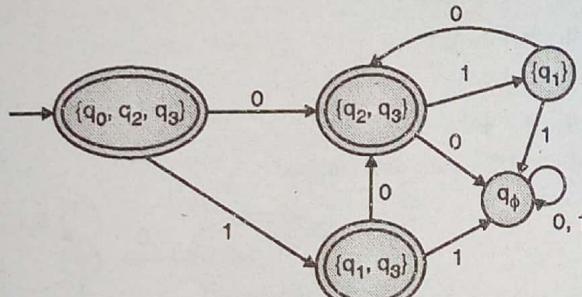


Fig. 3.14

**Q. 20** Using pumping lemma for regular sets, prove that the language,  $L = \{0^n \mid n \text{ is a prime}\}$  is not regular.

Dec. 2007, Dec. 2009, Dec. 2015, May 2016

**Ans. :**

**Step 1 :** Let us assume that the given language is regular and L is accepted by a FA with n states.

**Step 2 :** Let us choose a string  $w = a^p$ , where p is a prime and  $p > n$ .

$$|w| = |a^p| = p \geq n$$

Let us write w as xyz with

$$|y| > 0$$

$$\text{and } |xy| \leq n$$

We can assume that  $y = a^m$  for  $m > 0$ .

**Step 3 :** Length of  $xy^iz$  can be written as given below :

$$|xy^iz| = |xyz| + |y^{i-1}| = p + (i-1)m$$

$$\text{as } |y| = |a^m| = m$$

Let us check whether  $P(i-1)m$  is a prime for every i.

$$\text{For } i = p+1, p+(i-1)m = P + P_m = P(1+m).$$

$P(1+m)$  is not a prime as it has two factors p and  $(1+m)$  and

$$|p| > 1,$$

$$|1+m| > 1$$

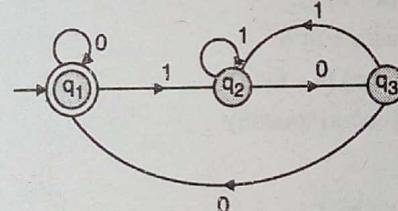
So  $xy^{p+1}z \notin L$ . Hence by contradiction the given language is not regular.

**Q. 21** Draw a state diagram and construct a regular expression corresponding to the following state transition table. Dec. 2016

State	0	1
$\rightarrow^* q_1$	$q_1$	$q_2$
$q_2$	$q_3$	$q_2$
$q_3$	$q_1$	$q_2$

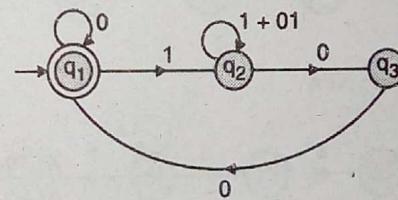
**Ans. :**

**State diagram**

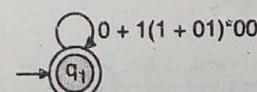


**R.E. form state diagram**

**Step 1 :** Removing loop between  $q_2$  and  $q_3$  we get



**Step 2 :** Removing the main loop, we get



**Q. 22** Show that the language  $L = \{a^n b^n\}$  is not regular.

Dec. 2006, May 2010, Dec. 2010, Dec. 2012, May 2013

May 2014, Dec. 2016, May 2017, Dec. 2017

**Ans. :**

**Step 1 :** Let us assume that L is regular and L is accepted by a FA with n states.

**Step 2 :** Let us choose a string

$$\omega = a^n b^n$$

$$|\omega| = 2n \geq n$$

Let us write  $\omega$  as xyz, with

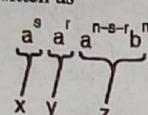
$$|y| > 0$$

$$\text{and } |xy| \leq n$$

Since,  $|xy| \leq n$ , y must be of the form  $a^r$  ( $r > 0$ )

Since,  $|xy| \leq n$ , x must be the form  $a^s$ .

Now,  $a^n b^n$  can be written as



**Step 3 :** Let us check whether  $xy^iz$  for  $i = 2$  belongs to L.

$$\begin{aligned} xy^2z &= a^s(a^r)^2 a^{n-s-r} b^n \\ &= a^s a^{2r} a^{n-s-r} b^n \\ &= a^{s+2r+n-s-r} b^n \\ &= a^{n+r} b^n \end{aligned}$$

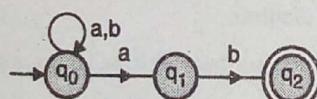
Since  $r > 0$ , number of a's in  $a^{n+r} b^n$  is greater than number of b's. Therefore,  $xy^2z \notin L$ . Hence by contradiction we can say that the given language is not regular.

### Q. 23 Construct NFA for given regular expressions :

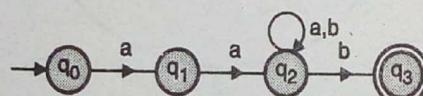
- (i)  $(a + b)^*ab$
- (ii)  $aa(a + b)^*b$
- (iii)  $(aba)(a + b)^*$
- (iv)  $(ab/ba)^*|(aa/bb)^*$

**Ans. :**

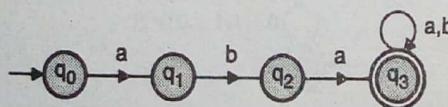
(i)  $(a + b)^*ab : \text{NFA}$



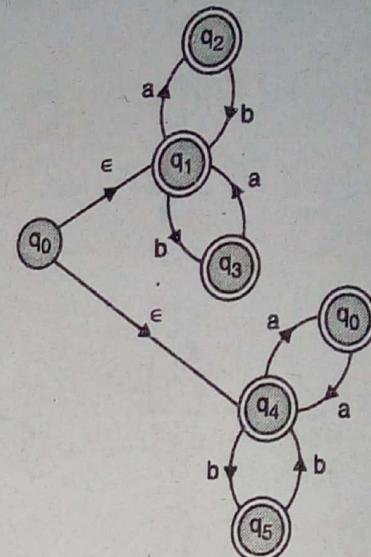
(ii)  $aa(a + b)^*b : \text{NFA}$



(iii)  $(aba)(a + b)^* : \text{NFA}$



(iv)  $(ab/ba)^*|(aa/bb)^* : \text{NFA}$

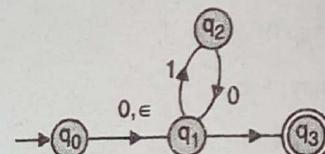


**Q. 24 Convert  $(0 + \epsilon)(10)^*(\epsilon + 1)$  into NFA with  $\epsilon$ -moves and obtain DFA.**

Dec/2017

**Ans. :**

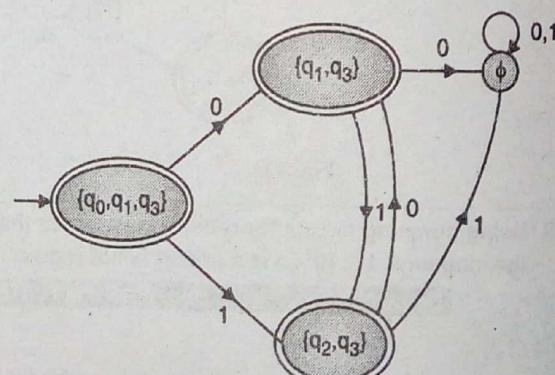
**Step 1 :** NFA for the given expression :



**Step 2 :**  $\epsilon$ -closure of states :

- $q_0 \rightarrow \{q_0, q_1, q_3\}$
- $q_1 \rightarrow \{q_1, q_3\}$
- $q_2 \rightarrow \{q_2\}$
- $q_3 \rightarrow \{q_3\}$

**Step 3 :** DFA using direct method :



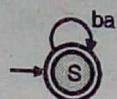
## Chapter 4 : Regular Grammar (RG)

**Q. 1 Construct right linear grammar and left linear grammar for the language  $(ba)^*$ .**

Dec. 2006

**Ans. :**

Transition system for  $(ba)^*$  is given by :



We can write left linear grammar and the right linear grammar from the transition systems.

**Right linear grammar :**

$$S \rightarrow baS \mid \epsilon$$

**Left linear grammar :**

$$S \rightarrow Sba \mid \epsilon$$

**Q. 2 Final the equivalent DFA accepting the regular language defined by the right linear grammar given as :**

$$S \rightarrow aA \mid bB, A \rightarrow aA \mid bc \mid aB \rightarrow aB \mid bC \rightarrow bB$$

May 2009

**Ans. :**

A new final state F is being introduced to handle productions like,

$$A \rightarrow a, \quad B \rightarrow b$$

**Step 1 :** Adding transitions corresponding to every production, we get the FA shown in Fig. 4.1(a).

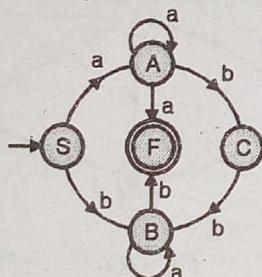


Fig. 4.1(a)

**Step 2 :** Drawing an equivalent DFA, we get :

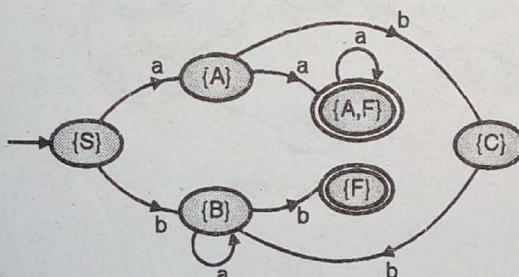


Fig. 4.1(b)

**Step 3 :** A dead state is added to handle  $\phi$ -transition. The resulting DFA is shown in Fig. 4.1(c).

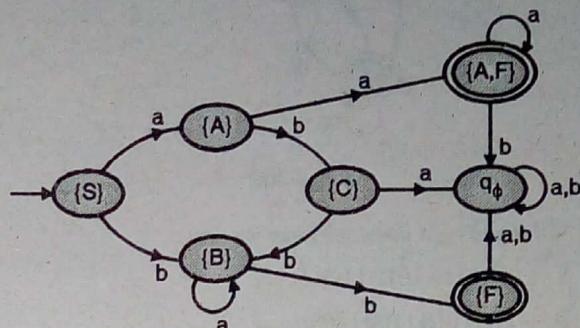


Fig. 4.1(c)

**Q. 3 Construct left linear and right linear grammar for the regular expression.**

$$((01 + 10)^*11^*00)^*$$

May 2009

**Ans. :**

The given expression can be represented using a transition system as shown below :

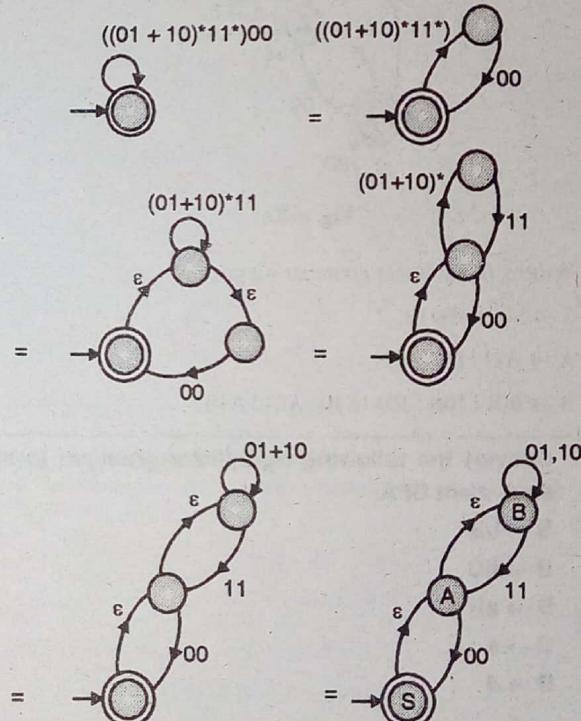


Fig. 4.2(a)

Removing  $\epsilon$ -transitions, we get :

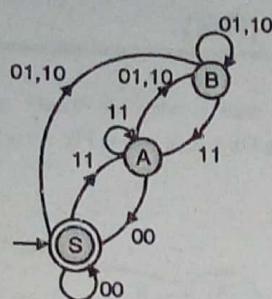


Fig. 4.2(b)

Writing of right linear grammar we get,

$$\begin{aligned} S &\rightarrow 00S \mid 11A \mid 01B \mid 10B \mid \epsilon \\ A &\rightarrow 11A \mid 01B \mid 10B \mid 00S \\ B &\rightarrow 01B \mid 10B \mid 11A \end{aligned}$$

For writing of left linear grammar, we interchange the start state and the final state and change direction of all transitions. The resulting transition system is given by :

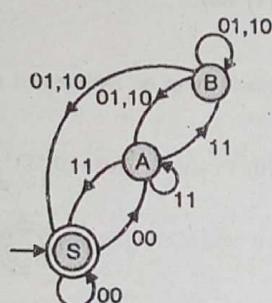


Fig. 4.2(c)

Writing of left linear grammar we get,

$$\begin{aligned} S &\rightarrow S00 \mid A00 \mid \epsilon \\ A &\rightarrow A11 \mid B11 \mid S11 \\ B &\rightarrow B01 \mid 10B \mid S01 \mid S10 \mid A01 \mid A10 \end{aligned}$$

#### Q. 4 Convert the following right-linear grammar to an equivalent DFA.

$$\begin{aligned} S &\rightarrow bB \\ B &\rightarrow bC \\ B &\rightarrow aB \\ C &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

**Ans. :**

Re-writing the production we get

$$\begin{aligned} S &\rightarrow bB \\ B &\rightarrow bC \mid b \\ B &\rightarrow aB \\ C &\rightarrow a \end{aligned}$$

**Step 1 :** Adding transitions corresponding to every production, we get

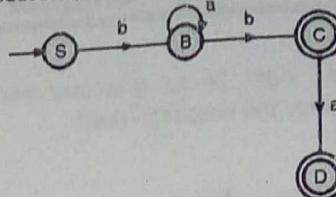


Fig. 4.3(a)

**Step 2 :** Adding a state E to handle  $\phi$ -transitions, we get the final DFA.

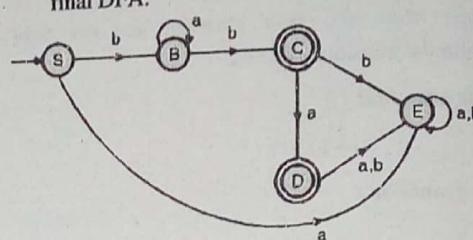


Fig. 4.3(b) : Final DFA

#### Q. 5 Convert following RG to DFA

$$\begin{aligned} S &\rightarrow 0A \mid 1B, \quad A \rightarrow 0C \mid 1A \mid 0, \\ B &\rightarrow 1B \mid 1A \mid 1, \quad C \rightarrow 0 \mid 0A. \end{aligned}$$

**Ans. :**

A new final state F is being introduced to handle productions like,  
 $A \rightarrow 0, B \rightarrow 1, C \rightarrow 0$ .

**Step 1 :** Adding transitions corresponding to every production, we get the FA shown in Fig. 4.4(a).

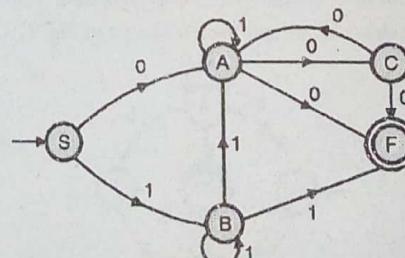


Fig. 4.4(a)

**Step 2 :** Drawing an equivalent DFA, we get

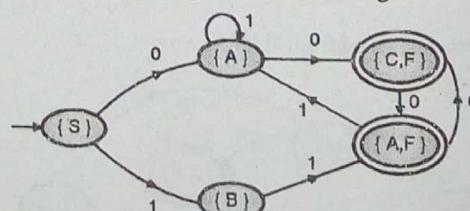


Fig. 4.4(b)

**Step 3 :** States  $\{S\}$ ,  $\{A\}$ ,  $\{B\}$ ,  $\{C, F\}$ , and  $\{A, F\}$  are renamed as  $q_0, q_1, q_2, q_3, q_4$  and a dead state  $q_\phi$  is introduced to handle  $\phi$  – transitions. The resulting DFA is shown in Fig. 4.4(c) :

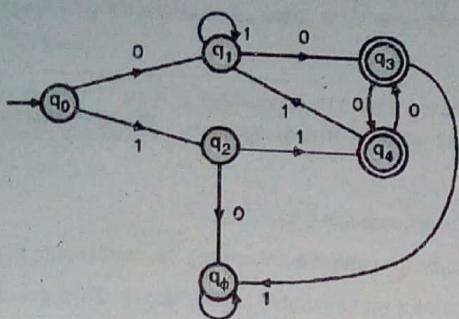


Fig. 4.4(c) : Final DFA

**Q. 6** Write an equivalent left linear grammar from the given right linear grammar.

$$\begin{aligned} S &\rightarrow 0A \mid 1B \\ A &\rightarrow 0C \mid 1A \mid 0 \\ B &\rightarrow 1B \mid 1A \mid 1 \\ C &\rightarrow 0 \mid 0A \end{aligned}$$

**Ans. :**

**Step 1 :** Transition system for the given right linear grammar is as shown in Fig. 4.5(a).

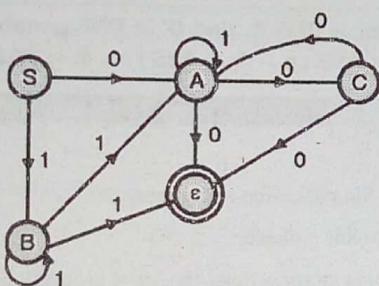


Fig. 4.5(a) : Transition graph

## Chapter 5 : Context Free Grammars (CFG)

**Q. 1** Write an unambiguous CFG for arithmetic expressions with operators : +, \*, /, ^, unary minus and operand a, b, c, d, e, f. Also, if should be possible to generate brackets with your grammar. Derive  $(a + b)^d / e + (-f)$  from your grammar.

Dec 2005

**Ans. :**

An unambiguous grammar is given below.

$E \rightarrow E + T \mid T$  [+ has lowest priority with L  $\rightarrow$  R associativity]

**Step 2 :** Interchanging the start state with the final state and reversing direction of transitions, we get

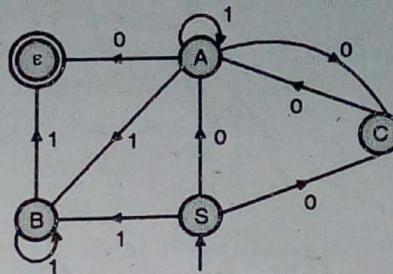


Fig. 4.5(b)

**Step 3 :** Writing of left linear grammar from the transition system, we get :

$$\begin{aligned} S &\rightarrow C0 \mid A0 \mid B1 \\ A &\rightarrow A1 \mid C0 \mid B1 \mid 0 \\ B &\rightarrow B1 \mid 1 \\ C &\rightarrow A0. \end{aligned}$$

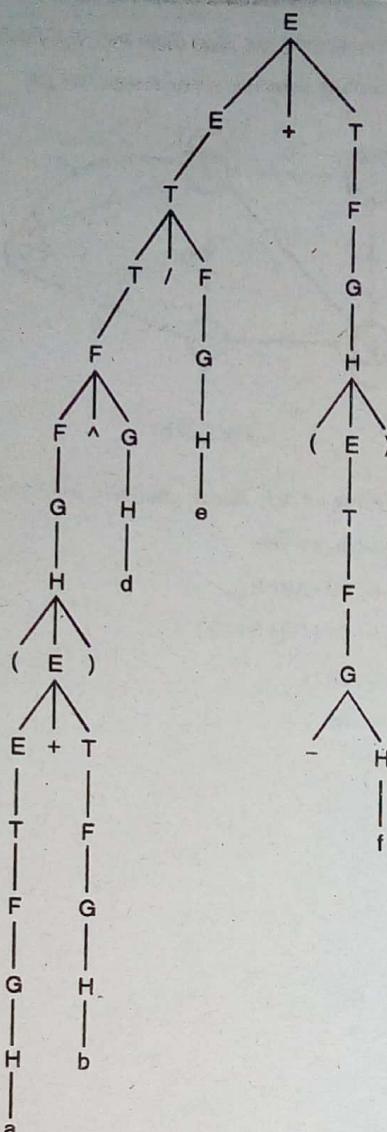
$T \rightarrow T * F \mid T / F \mid F$  [\* and / has higher priority over + with L  $\rightarrow$  R associativity]

$F \rightarrow F^G \mid G$  [^ has higher priority over \* and / with L  $\rightarrow$  R associativity]

$G \rightarrow -H \mid H$  [unary – has the highest priority]

$H \rightarrow a \mid b \mid c \mid d \mid e \mid f \mid (E)$  [to handle brackets and identifiers]

Derivation tree for  $(a + b)^d / e + (-f)$

Fig. 5.1 : Derivation tree for  $(a + b)^d / e + (-f)$ **Q. 2 Convert the following CFG to GNF :**

$S \rightarrow aSa \mid bSb \mid c$

Dec. 2005

**Ans. :**

The grammar can be brought to GNF through simple substitutions  $C_a \rightarrow a$  and  $C_b \rightarrow b$ .

$S \rightarrow aSC_a \mid bSC_b \mid C$

$C_a \rightarrow a$

$C_b \rightarrow b$

**Q. 3 Write short note on GNF.****Ans. :****Greibach Normal Form (GNF)**

A context free grammar  $G = (V, T, P, S)$  is said to be in GNF if every production is of the form :

$A \rightarrow a\alpha,$

Where,  $a \in T$  is a terminal and  $\alpha$  is a string of zero or more variables.

The language  $L(G)$  should be without  $e$ .

Right hand side of each production should start with a terminal followed by a string of non-terminals of length zero or more.

**Q. 4 Prove that the language  $L = \{a^p \mid p \text{ is a prime}\}$  is not context free language.** [May 2006, May 2012]

**Ans. :**

1. Let us assume that  $L$  is a CFL.
2. Let  $n$  be the natural number for  $L$ , as per the pumping lemma.
3. Let  $p$  be a prime number greater than  $n$ . Then  $z = a^p \in L$ . We can write  $z = uvxyz$ .
4. By pumping lemma  $uv^0xy^0z = uxz \in L$ . Therefore,  $|uxz|$  is a prime number.

Let us assume that  $|uxz| = q$ .

Now, let us consider a string  $uv^qxy^qz$ ,

The length of  $uv^qxy^qz$  is given by :

$|uv^qxy^qz| = q + q(|v| + |y|)$ , which is not a prime with  $q$  is a factor.

Thus,  $uv^qxy^qz \notin L$ . This is a contradiction.

Therefore,  $L$  is not a context free language.

**Q. 5 Given a CFG  $G$ , find  $G'$  in CNF generating  $L(G)$  - e  
 $S \rightarrow ASB \mid \epsilon \quad A \rightarrow AaS \mid a \quad B \rightarrow SbS \mid A \mid bb$**

[May 2006, May 2009, May 2010, Dec. 2011]

**Ans. :****Step 1 : Simplification of grammar**

Symbol  $S$  is nullable.

After removing  $\epsilon$ -productions, the set of productions is given by

$S \rightarrow ASB \mid AB$

$A \rightarrow AaS \mid Aa \mid a$

$B \rightarrow SbS \mid Sb \mid bS \mid b \mid A \mid bb$

Unit production  $B \rightarrow A$  is removed, the resulting set of productions is given by

$S \rightarrow ASB \mid AB$

$A \rightarrow AaS \mid Aa \mid a$

$B \rightarrow SbS \mid Sb \mid bS \mid b \mid AaS \mid Aa \mid a \mid bb$

**Step 2 : Every symbol in  $\alpha$ , in productions of the form  $A \rightarrow \alpha$  where  $|\alpha| \geq 2$  should be a variable.**

This can be done by adding two productions :

$C_a \rightarrow a$

and  $C_b \rightarrow b$

The set of productions after the above changes is

$$\begin{aligned} S &\rightarrow ASB \mid AB \\ A &\rightarrow AC_aS \mid AC_a \mid a \\ B &\rightarrow SC_bS \mid SC_b \mid C_bS \mid b \mid AC_aS \mid AC_a \mid a \mid C_bC_b \\ C_a &\rightarrow a \\ C_b &\rightarrow b \end{aligned}$$

### Step 3 : Finding an equivalent CNF

Original production	Equivalent productions in CNF
$S \rightarrow ASB$	$S \rightarrow AC_1$ $C_1 \rightarrow SB$
$S \rightarrow AB$	$S \rightarrow AB$
$A \rightarrow AC_aS$	$A \rightarrow AC_2$ $C_2 \rightarrow C_aS$
$A \rightarrow AC_a$	$A \rightarrow AC_a$
$A \rightarrow a$	$A \rightarrow a$
$B \rightarrow SC_bS$	$B \rightarrow SC_3$ $C_3 \rightarrow C_bS$
$B \rightarrow SC_b \mid C_bS \mid b$	$S \rightarrow SC_b \mid C_bS \mid b$
$B \rightarrow AC_aS$	$B \rightarrow AC_2$
$B \rightarrow AC_a \mid a \mid C_bC_b$	$B \rightarrow AC_a \mid a \mid C_bC_b$
$C_a \rightarrow a$	$C_a \rightarrow a$
$C_b \rightarrow b$	$C_b \rightarrow b$

### Q. 6 Convert the following grammar into GNF

$$S \rightarrow XY1I0 \quad X \rightarrow 00XIY \quad Y \rightarrow 1X1$$

May 2006, May 2012

### Ans. :

Simplification of grammar

The unit production  $X \rightarrow y$  is removed, the equivalent set of productions is given by :

$$\begin{aligned} S &\rightarrow XY1I0 \\ X &\rightarrow 00XIY \\ Y &\rightarrow 1X1 \end{aligned}$$

The symbol X is non-generating.

The set of productions after elimination of X is given by :

$$S \rightarrow 0, \text{ it is in GNF}$$

### Q. 7 Find CFG for generating

- String containing alternate sequence of 0's and 1's;  $\Sigma = \{0, 1\}$
- The string containing no consecutive 'b's but 'a's can be consecutive.
- The set of all string over alphabet {a, b} with exactly twice as many a's as b's.
- Language having number of a's greater than number of b's.

Dec. 2006, May 2009, Dec. 2009

Ans. :

- String containing alternate sequence of 0's and 1's;  $\Sigma = \{0, 1\}$

Since, any binary number will satisfy the condition of alternate sequence of 0's and 1's, the language  $L = (0 + 1)^*$

The set of productions are :

$$S \rightarrow 0S \mid 1S \mid \epsilon$$

$$\therefore \text{CFG } G = (\{S\}, \{0, 1\}, \{S \rightarrow 0S \mid 1S \mid \epsilon\}, S)$$

- The string containing no consecutive b's but a's can be consecutive.

The set of productions for the given language L are :

$$P = \{$$

$$S \rightarrow aS \mid bX \mid b \mid \epsilon$$

$$X \rightarrow aSa$$

}

These production can easily be written from the FA for the above language. The FA is shown in Fig. Ex. 5.2.33.

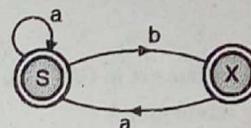


Fig. 5.2

$$\text{Set of variables } V = \{S, X\}$$

$$\text{Set of terminals } T = \{a, b\}$$

$$\text{Start symbol } = S$$

- The set of all strings over alphabet {a, b} with exactly twice as many a's as b's.

$$\text{The CFG } G = (V, T, P, S)$$

$$\text{Where } V = \{S\}$$

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

$$P = \{S \rightarrow aSaSbS \mid aSbSaS \mid bSaSaS \mid \epsilon\}$$

$$S = \text{Start symbol}$$

- Language having number of a's greater than number of b's.

The set of productions for the grammar are given by :

$$P = \{$$

$$S \rightarrow SaS \mid aSS \mid SSa \mid a \mid aX \mid Xa$$

$$X \rightarrow aB \mid bA$$

$$A \rightarrow aX \mid bAA \mid a$$

$$B \rightarrow bX \mid aBB \mid b$$

}

The variable X generates a string having equal number of a's and b's. Group of excess a's over b's are generated by S-productions.

Where

Set of variables  $V = \{S, X, A, B\}$

Set of terminals  $T = \{a, b\}$

Start symbol = S

**Q. 8 Convert the given grammar to GNF.**

$$S \rightarrow SSlaSblaB$$

Dec. 2006

**Ans. :**

- Step 1 :** Other than the first symbol on the RHS of every production, every symbol must be a variable.

We can make the substitution X for b.

The resulting set of productions after the above substitution is :

$$S \rightarrow SSlaSXlaX$$

$$X \rightarrow b$$

- Step 2 :** Removing left recursion from s-production, we get :

$$S \rightarrow aSX_{1}laXS_{1}laSXlaX$$

$$S_{1} \rightarrow SS_{1}IS$$

$$X \rightarrow b$$

- Step 3 :**  $S_1$ -productions are not in GNF. They can be brought to GNF by substituting S.

$$S \rightarrow aSX_{1}laXS_{1}laSXlaX$$

$$S_{1} \rightarrow aSX_{1}S_{1}laXS_{1}S_{1}laSX_{1}laXS_{1}laSX_{1}laXS_{1}laSXlaX$$

$$X \rightarrow b$$

**Q. 9 Prove that  $L = \{0^i 1^j 2^k 3^l \mid i \geq 1 \text{ and } j \geq 1\}$  is not context free.**

Dec. 2007

**Ans. :**

- Let us assume that L is CFL
- Let us pick up a word  $\omega = 0^n 1^n 2^n 3^n$ , where the constant n is given as per the pumping lemma.
- $\omega$  is rewritten as  $uvxyz$  where  $|vxy| \leq n$  and  $v \cdot y \neq \epsilon$  i.e. both v and y are not null.
- From pumping lemma, if  $uvxyz \in L$  then  $uv^i xy^i z$  is in L(G) for each  $i = 0, 1, 2, \dots$

There are two case :

**Case I :** vy contains three symbols. These three symbols could be 0,1,2 or 1,2,3.

The exact ordering of 0,1,2,3 will be broken in  $uv^2 xy^2 z$  and hence  $uv^2 xy^2 z \notin L(G)$

**Case II :** If vy does not contain three symbols then  $uv^2 xy^2 z$  will have either unequal number of 0's and 2's or unequal number of 1's and 3's. Hence,  $uv^2 xy^2 z \notin L(G)$ .

Thus, proved by contradiction.

**Q. 10 Prove that  $L = \{a^i b^i c^i \mid i \geq 1\}$  is not a CFL.**

May 2008

**Ans. :**

- Let us assume that L is CFL.
- Let us pick up a word  $w = a^n b^n c^n$  where the constant n is given as per the pumping lemma.
- w is rewritten as  $uvxyz$ .  
Where  $|vxy| \leq n$  and  $v \cdot y \neq \epsilon$  i.e., both v and y are not null.
- From pumping lemma, if  $uvxyz \in L$  then  $uv^i xy^i z$  is in L (G) for each  $i = 0, 1, 2, \dots$

There are two cases :

**Case I :** vy contains all three symbols a, b and c.

If vy contains all three symbols a, b and c then either v or y contains two symbols. The exact ordering of a, b and c will be broken in  $uv^2 xy^2 z$  and hence  $uv^2 xy^2 z \notin L(G)$

**Case II :** If vy does not contain three symbols a, b and c then  $uv^2 xy^2 z$  will have unequal number of a's, b's and c's and hence  $uv^2 xy^2 z \notin L(G)$ .

Hence, it is proved by contradiction.

**Q. 11 Convert the following grammar to CNF  $S \rightarrow AACD$**

$$A \rightarrow aAb \mid \epsilon \quad C \rightarrow aC \mid a \quad A \rightarrow aDa \mid bDb \mid \epsilon$$

May 2008

**Ans. :**

First of all, the grammar must be simplified.

- Step 1 :** Removing null productions.

$$\text{Nullable set} = \{A\}$$

Null productions are removed with the resulting set of production as :

$$S \rightarrow AACD \mid ACD \mid CD$$

$$A \rightarrow aAbab$$

$$C \rightarrow aCla$$

$$A \rightarrow aDaDbDb$$

- Step 2 :** Removing non-generating symbol

Symbol S and D are non-generating.

Since, the starting symbol itself is non-generating, it is an invalid grammar.

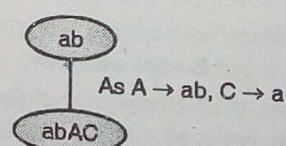


Fig. 5.3

- Q. 12 Given a CFG G, find G' in CNF generating L(G) - ε**
- $S \rightarrow ASB \mid \epsilon$   
 $A \rightarrow AaS \mid a$   
 $B \rightarrow SbS \mid A \mid bb$

May 2006, May 2009, May 2010, Dec. 2011.

**Ans. :**

**Step 1 :** Simplification of grammar

Symbol S is nullable.

After removing ε-productions, the set of productions is given by

$$S \rightarrow ASB \mid AB$$

$$A \rightarrow AaS \mid Aa \mid a$$

$$B \rightarrow SbS \mid Sb \mid bS \mid b \mid A \mid bb$$

Unit production  $B \rightarrow A$  is removed, the resulting set of productions is given by

$$S \rightarrow ASB \mid AB$$

$$A \rightarrow AaS \mid Aa \mid a$$

$$B \rightarrow SbS \mid Sb \mid bS \mid b \mid AaS \mid Aa \mid bb$$

**Step 2 :** Every symbol in  $\alpha$ , in productions of the form  $A \rightarrow \alpha$  where  $|\alpha| \geq 2$  should be a variable.

This can be done by adding two productions :

$$C_a \rightarrow a$$

and  $C_b \rightarrow b$

The set of productions after the above changes is

$$S \rightarrow ASB \mid AB$$

$$A \rightarrow AC_aS \mid AC_a \mid a$$

$$B \rightarrow SC_bS \mid SC_b \mid C_bS \mid b \mid AC_aS \mid AC_a \mid a \mid C_bC_b$$

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

**Step 3 :** Finding an equivalent CNF

Original production	Equivalent productions in CNF
$S \rightarrow ASB$	$S \rightarrow AC_1$ $C_1 \rightarrow SB$
$S \rightarrow AB$	$S \rightarrow AB$
$A \rightarrow AC_aS$	$A \rightarrow AC_2$ $C_2 \rightarrow C_aS$
$A \rightarrow AC_a$	$A \rightarrow AC_a$
$A \rightarrow a$	$A \rightarrow a$
$B \rightarrow SC_bS$	$B \rightarrow SC_3$ $C_3 \rightarrow C_bS$
$B \rightarrow SC_b \mid C_bS \mid b$	$S \rightarrow SC_b \mid C_bS \mid b$
$B \rightarrow AC_aS$	$B \rightarrow AC_2$
$B \rightarrow AC_a \mid a \mid C_bC_b$	$B \rightarrow AC_a \mid a \mid C_bC_b$
$C_a \rightarrow a$	$C_a \rightarrow a$
$C_b \rightarrow b$	$C_b \rightarrow b$

- Q. 13 Let  $G = (V, T, P, S)$  be the CFG having following set of productions. Derive the string "aabbaa" using leftmost derivation and rightmost derivation.**

$$S \rightarrow aAS \mid a, A \rightarrow SbA \mid SS \mid ba$$

May 2009

**Ans. :**

**(i) Leftmost derivation :**

Leftmost derivation of aabbaa is being shown with the help of the parse tree.

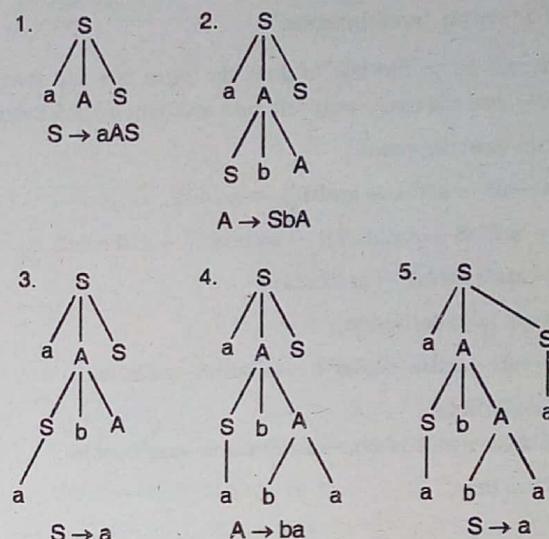


Fig. 5.4(a)

$$S \rightarrow aAS \rightarrow aSbAS \rightarrow aabAS \rightarrow aabbAS \rightarrow aabbaa$$

**(ii) Rightmost derivation :**

Rightmost derivation of aabbaa is being shown with the help of the parse tree.

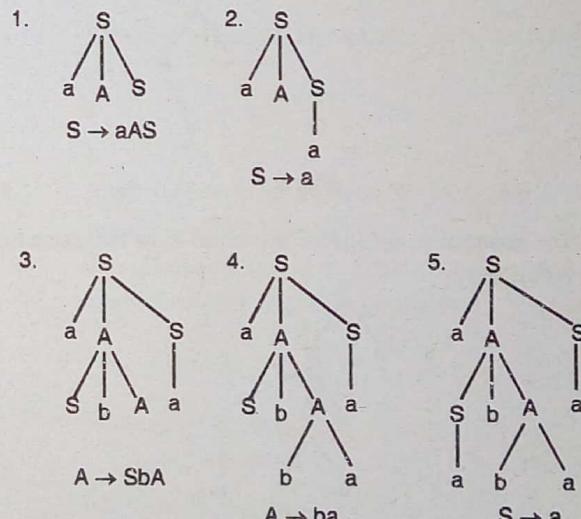


Fig. 5.4(b)

$$S \rightarrow aAS \rightarrow aAa \rightarrow aSbAa \rightarrow aSbbAa \rightarrow aabbaa$$

- Q. 14** Let G be the grammar  $S \rightarrow aB \mid bA \mid A \rightarrow a \mid aS \mid bAA \mid b \mid bS \mid aBB$ . Find  
 (i) Left most derivation  
 (ii) Right most derivation  
 (iii) Parse Tree  
 (iv) Is the grammar unambiguous?

For given strings (A) aaabbabbba (B) bbaaabababa  
 (C) 00110101      Dec. 2009, Dec. 2012, May 2013

**Ans. :**

**(A) For string "aaabbabbba"**

It will be worthwhile to draw the parse tree and from the parse tree, one can easily write left most and right most derivation.

- (i) Left most derivation :

$$\begin{aligned} S &\rightarrow aB \rightarrow aaBB \rightarrow aaaBBB \rightarrow aaabBB \\ &\rightarrow aaabbB \rightarrow aaabbaBB \rightarrow aaabbabB \rightarrow aaabbabbS \\ &\rightarrow aaabbabbA \rightarrow aaabbabbba \end{aligned}$$

- (ii) Right most derivation :

$$\begin{aligned} S &\rightarrow aB \rightarrow aaBB \rightarrow aaBaBB \rightarrow aaBaBbS \rightarrow aaBaBbbA \\ &\rightarrow aaBaBbba \\ &\rightarrow aaBabbba \rightarrow aaaBBabbba \rightarrow aaaBbabbbba \rightarrow aaabbabbba \end{aligned}$$

- (iii) Parse tree :

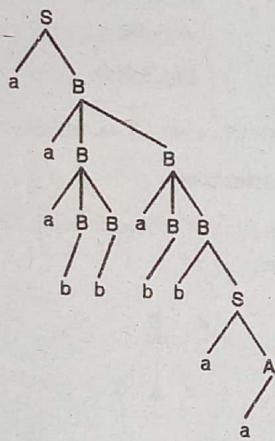
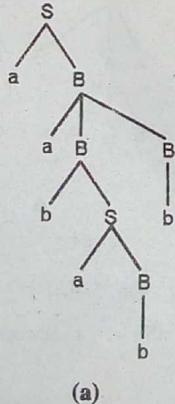


Fig. 5.5

- (iv) The grammar is ambiguous as we can draw two parse trees for aabbabb :



(a)

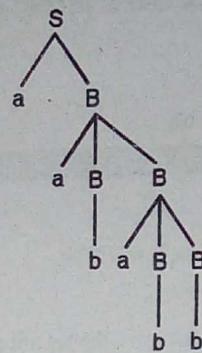


Fig. 5.5

**(B) For string "bbaaabababa"**

- (i) Leftmost derivation

$$\begin{aligned} S &\rightarrow bA \rightarrow bbAA \rightarrow bbaA \rightarrow bbaaS \\ &\rightarrow bbaaaB \rightarrow bbaaabs \rightarrow bbaaabA \\ &\rightarrow bbaaabbas \rightarrow bbaaabbabA \rightarrow bbaaabababa \end{aligned}$$

- (ii) Rightmost derivation

$$\begin{aligned} S &\rightarrow bA \rightarrow bbAA \rightarrow bbAaS \rightarrow bbAaaB \\ &\rightarrow bbAaS \rightarrow bbAaabbA \rightarrow bbAaabbAS \\ &\rightarrow bbAaabbAS \rightarrow bbAaabbabA \rightarrow bbaaabababa \end{aligned}$$

- (iii) Parse tree for bbaaabababa

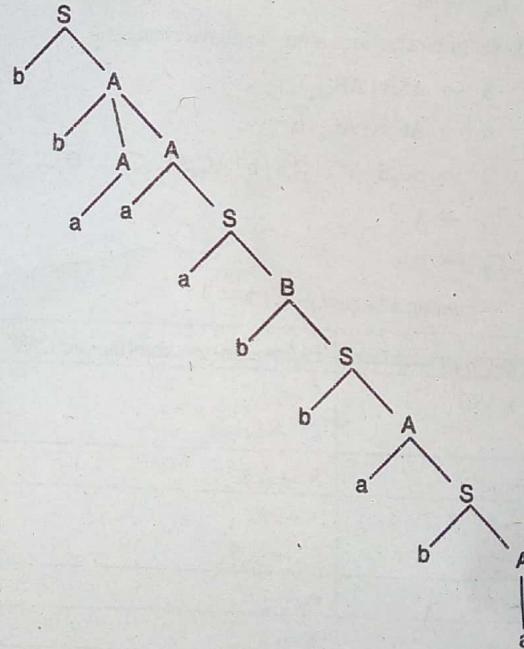


Fig. 5.5(c)

**(C) For the string 00110101**

- (i) Leftmost derivation

$$\begin{aligned} S &\rightarrow 0BB \rightarrow 00BB \rightarrow 001B \rightarrow 0011S \\ &\rightarrow 00110 \quad B \rightarrow 001101S \rightarrow 0011010B \\ &\rightarrow 00110101 \end{aligned}$$

(ii) Rightmost derivation

$$\begin{aligned} S &\rightarrow 0B \rightarrow 00BB \rightarrow 00B1S \rightarrow 00B10B \\ &\rightarrow 00B101S \rightarrow 00B1010B \rightarrow 00B10101 \\ &\rightarrow 001110101 \end{aligned}$$

(iii) Parse tree

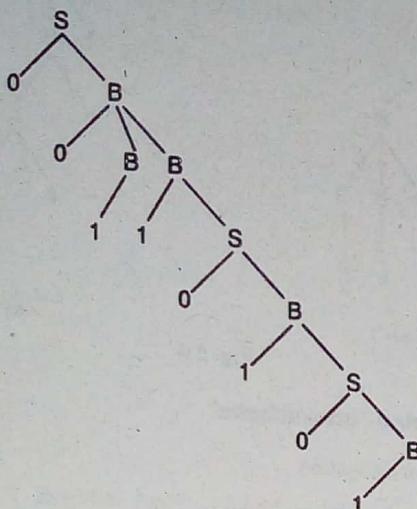


Fig. 5.5(d)

**Q. 15 Obtain a grammar to generate the language  $L = \{0^n 1^{2n} \mid n \geq 0\}$ .**

May 2010

**Ans. :**

Productions for the required language are as follows.

$$P = \{S \rightarrow 0S11 \mid \epsilon\}$$

CFG for the above language is ( $\{S\}$ ,  $\{0, 1\}$ , P, S)

**Q. 16 Reduce the following grammar to GNF.  $S \rightarrow AB$ ,  $A \rightarrow BSB \mid BB \mid b$ ,  $B \rightarrow aAb \mid a$**

May 2011

**Ans. :**

**Step 1 :** Making every symbol other than the first symbol (in derived string  $\alpha$  in  $A \rightarrow \alpha$ ) as a variable :

Variables  $C_b$  is substituted for b with resulting set of productions give as :

$$S \rightarrow AB$$

$$A \rightarrow BSB \mid BB \mid b$$

$$B \rightarrow aAC_b \mid a, C_b \rightarrow b$$

**Step 2 :** The variables S, A, B and  $C_b$  are renamed as  $A_1$ ,  $A_2$ ,  $A_3$  and  $A_4$  respectively. The resulting set of productions is given below.

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 A_3 \mid A_3 A_3 \mid b$$

$$A_3 \rightarrow aA_1 A_4 \mid a$$

$$A_4 \rightarrow b$$

**Step 3 : Convert to CFG**

**Given production**

**Equivalent Production in GNF**

$$A_4 \rightarrow b \rightarrow A_4 \rightarrow b$$

$$A_3 \rightarrow aA_1 A_4 \mid a \rightarrow A_3 \rightarrow aA_1 A_4 \mid a$$

$$A_2 \rightarrow A_3 A_1 A_3$$

↓

$$\xrightarrow{\text{Substituting } A_3} A_2 \rightarrow aA_1 A_4 A_1 A_3 \mid aA_1 A_3$$

$$A_2 \rightarrow A_3 A_3$$

↓

$$\xrightarrow{\text{Substituting } A_3} A_2 \rightarrow aA_1 A_4 A_3 \mid aA_3$$

$$A_2 \rightarrow b$$

$$A_1 \rightarrow A_2 A_3$$

↓

$$\xrightarrow{\text{Substituting } A_2} A_1 \rightarrow aA_1 A_4 A_1 A_3 A_3 \mid aA_1 A_3 A_3 \\ \mid aA_1 A_4 A_1 A_3 A_3 \mid aA_3 A_3 \mid bA_3$$

∴ The final set of productions is :

$$A_1 \rightarrow aA_1 A_4 A_1 A_3 A_3 \mid aA_1 A_3 A_3 \mid aA_1 A_4 A_3 A_3 \mid aA_3 A_3 \mid bA_3$$

$$A_2 \rightarrow aA_1 A_4 A_1 A_3 \mid aA_1 A_3 \mid aA_1 A_4 A_3 \mid aA_3 \mid b$$

$$A_3 \rightarrow aA_1 A_4 \mid a$$

$$A_4 \rightarrow b$$

**Q. 17 Reduce the following grammars to GNF**

$$B \rightarrow aAb \mid a, S \rightarrow AA \mid 1, A \rightarrow SS \mid 1 \quad \text{May 2011}$$

**Ans. :**

**Step 1 :** Renaming of variables by substituting  $A_1$  for S and  $A_2$  for A.

$$A_1 \rightarrow A_2 A_2 \mid 1$$

$$A_2 \rightarrow A_1 A_1 \mid 1$$

**Step 2 :** Every production of the form  $A_i \rightarrow A_j \alpha$  with  $i > j$  must be modified to make  $i \leq j$ .

$A_2$  - production,  $A_2 \rightarrow A_1 A_1$  should be modified. We must substitute  $A_2 A_2 \mid 1$  for the first  $A_1$ .

$$[A_2 \rightarrow A_1 A_1] \Rightarrow \begin{cases} A_2 \rightarrow A_2 A_2 A_1 \\ A_2 \rightarrow 1 A_1 \end{cases}$$

The resulting set of productions is :

$$A_1 \rightarrow A_2 A_2 \mid 1$$

$$A_2 \rightarrow A_2 A_2 A_1 \mid 1 A_1 \mid 1$$

**Step 3 :** Removing left recursion :

The  $A_2$  - production contains left recursion. Left recursion can be removed through

$$A_2 \rightarrow 1 A_1 B_2 \mid 1 B_2$$

$$B_2 \rightarrow A_2 A_1 B_2 \mid A_2 A_1$$

The resulting set of productions is :

$$A_1 \rightarrow A_2 A_2 \mid 1$$

$$A_2 \rightarrow 1 A_1 B_2 \mid 1 B_2 \mid 1 A_1 \mid 1$$

$$B_2 \rightarrow A_2 A_1 B_2 \mid A_2 A_1$$

**Step 4 :**  $A_2$  - productions are in GNF.

$A_1$  and  $B_2$  productions can be converted to GNF with the help of  $A_2$  - productions.

$$A_2 \rightarrow 1 A_1 B_2 \mid 1 B_2 \mid 1 A_1 \mid 1$$

$$A_1 \rightarrow 1 A_1 B_2 A_2 \mid 1 B_2 A_2 \mid 1 A_1 A_2 \mid 1 A_2 \mid 1$$

$$B_2 \rightarrow 1 A_1 B_2 A_1 B_2 \mid 1 B_2 A_1 B_2 \mid 1 A_1 A_1 B_2$$

$$\mid 1 A_1 B_2 \mid 1 A_1 B_2 A_1 \mid 1 B_2 A_1 \mid 1 A_1 A_1 \mid 1 A_1$$

**Q. 18 Let G be the grammar  $S \rightarrow aB \mid bAA \rightarrow a \mid 1 aS \mid bAA \mid B \rightarrow b \mid bS \mid aBB$  Find**

- (i) Left most derivation
- (ii) Right most derivation
- (iii) Parse Tree
- (iv) Is the grammar unambiguous ?

For given strings (A) aaabbabbba (B) bbaaabbbaba  
(C) 00110101      Dec. 2009, Dec. 2012, May 2013

**Ans. :**

**(A) For string "aaabbabbba"**

It will be worthwhile to draw the parse tree and from the parse tree, one can easily write left most and right most derivation.

(i) Left most derivation :

$$\begin{aligned} S &\rightarrow aB \rightarrow aaBB \rightarrow aaaBBB \rightarrow aaabBB \\ &\rightarrow aaabbB \rightarrow aaabbaBB \rightarrow aaabbabB \rightarrow aaabbabbS \\ &\rightarrow aaabbabbA \rightarrow aaabbabbba \end{aligned}$$

(ii) Right most derivation :

$$\begin{aligned} S &\rightarrow aB \rightarrow aaBB \rightarrow aaBaBB \rightarrow aaBaBbS \rightarrow aaBaBbbA \\ &\rightarrow aaBaBbba \\ &\rightarrow aaBabbba \rightarrow aaaBBabbba \rightarrow aaaBbabbba \rightarrow aaabbabbba \end{aligned}$$

(iii) Parse tree :

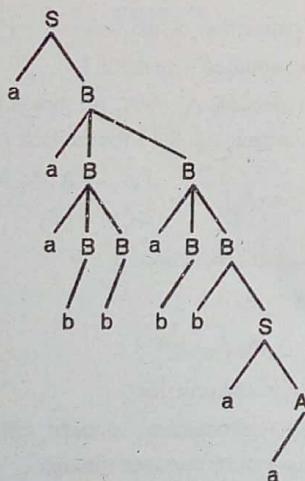
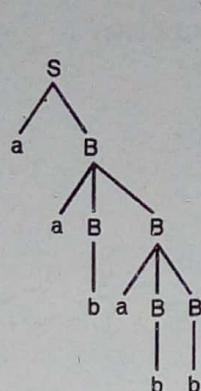
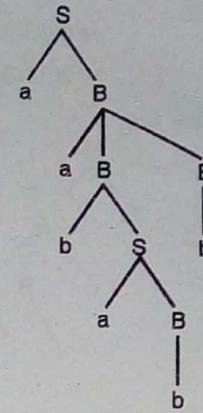


Fig. 5.6

(iv) The grammar is ambiguous as we can draw two parse trees for  $aabbabb$ :



(a)



(b)

Fig. 5.6

**(B) For string "bbaaabbbaba"**

(i) Leftmost derivation

$$\begin{aligned} S &\rightarrow bA \rightarrow bbAA \rightarrow bbaA \rightarrow bbaaS \\ &\rightarrow bbaaaB \rightarrow bbaaabs \rightarrow bbaaabba \\ &\rightarrow bbaaabbas \rightarrow bbaabbabA \rightarrow bhaabbaba \end{aligned}$$

(ii) Rightmost derivation

$$\begin{aligned} S &\rightarrow bA \rightarrow bbAA \rightarrow bbAaS \rightarrow bbAaaB \\ &\rightarrow bbAaabS \rightarrow bbAaabbA \rightarrow bbAaabbaS \\ &\rightarrow bbAaabbabA \rightarrow bbAaabbaba \rightarrow bbaabbaba \end{aligned}$$

(iii) Parse tree for bbaaabbbaba

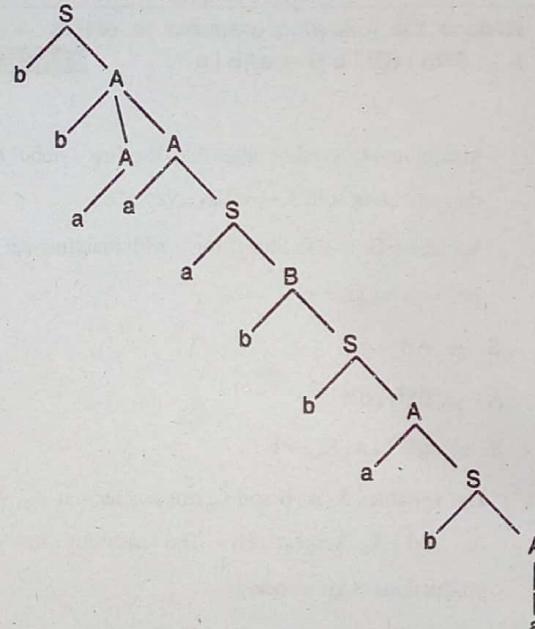


Fig. 5.6(c)

**(C) For the string 00110101**

(i) Leftmost derivation

$$S \rightarrow 0BB \rightarrow 00BB \rightarrow 001B \rightarrow 0011S$$

$\rightarrow 00110 \quad B \rightarrow 001101S \rightarrow 0011010B$   
 $\rightarrow 00110101$

## (ii) Rightmost derivation

$S \rightarrow 0B \rightarrow 00BB \rightarrow 00B1S \rightarrow 00B10B$   
 $\rightarrow 00B101S \rightarrow 00B1010B \rightarrow 00B10101$   
 $\rightarrow 00110101$

## (iii) Parse tree

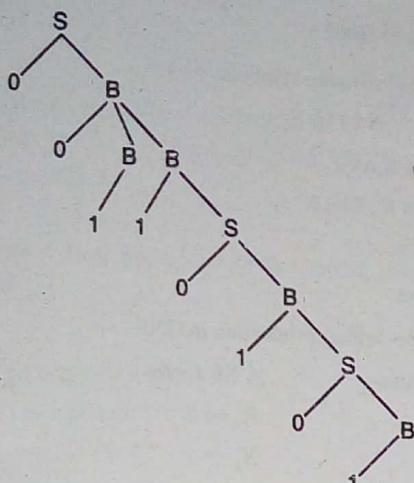


Fig. 5.6(d)

## Q. 19 Consider the following grammar :

$S \rightarrow iCtS \mid iCtSeS \mid a \quad C \rightarrow b$  For the String 'ibtibtaea' find the following : (i) Leftmost derivation (ii) Rightmost derivation (iii) Parse Tree (iv) Check if the above grammar is Ambiguous

May 2014

## Ans. :

## (i) Leftmost derivation :

$S \rightarrow iCtS$  [using  $S \rightarrow iCtS$ ]  
 $\rightarrow ibtS$  [using  $C \rightarrow b$ ]  
 $\rightarrow ibtiCtSeS$   
 $\quad$  [using  $S \rightarrow iCtSeS$ ]  
 $\rightarrow ibtibtSeS$  [using  $C \rightarrow b$ ]  
 $\rightarrow ibtibtaeS$  [using  $S \rightarrow a$ ]  
 $\rightarrow ibtibtaea$  [using  $S \rightarrow a$ ]

## (ii) Rightmost derivation :

$S \rightarrow iCtS$  [using  $S \rightarrow iCtS$ ]  
 $\rightarrow iCtCtSeS$   
 $\quad$  [using  $S \rightarrow iCtSeS$ ]  
 $\rightarrow iCtiCtSea$  [using  $S \rightarrow a$ ]  
 $\rightarrow iCteCtaea$  [using  $S \rightarrow a$ ]  
 $\rightarrow iCtebttaea$  [using  $C \rightarrow b$ ]  
 $\rightarrow ibtebttaea$  [using  $C \rightarrow b$ ]

## (iii) Parse Tree :

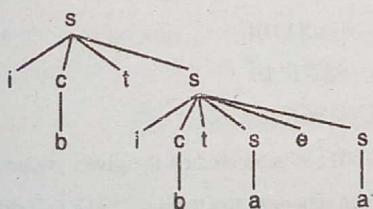


Fig. 5.7

Q. 20 Convert the following Grammar to CNF form :  
 $S \rightarrow ABA \mid A \rightarrow aA \mid bA \mid \epsilon \mid B \rightarrow bB \mid aA \mid \epsilon$ 

May 2014

## Ans. :

1. The non-terminals {S, A, B} are nullable. Null productions are removed. The resulting grammar is :

 $S \rightarrow ABA \mid BA \mid AB \mid AA \mid A \mid B$  $A \rightarrow aA \mid bA \mid a \mid b$  $B \rightarrow bB \mid aA \mid b \mid a$ 

2. Removing unit productions, we get

 $S \rightarrow ABA \mid BA \mid AB \mid AA \mid aA \mid bA \mid aB \mid bB \mid aB$  $A \rightarrow aA \mid bA \mid a \mid b$  $B \rightarrow bB \mid aA \mid b \mid a$ 

3. Every symbol in  $\alpha$ , in production of the form  $A \rightarrow \alpha$  where  $|\alpha| \geq 2$  should be a variable.

This can be done by adding two productions.

 $C_a \rightarrow a$  $C_b \rightarrow b$ 

The set of productions after the above changes is :

 $S \rightarrow ABA \mid BA \mid AB \mid AA \mid C_a A \mid C_b A \mid aB \mid C_b B \mid C_a A$  $A \rightarrow C_a A \mid C_b A \mid a \mid b$  $B \rightarrow C_b B \mid C_a A \mid b \mid a$  $C_a \rightarrow a, C_b \rightarrow b$ 

4. Finding an equivalent CNF.

Original production	Equivalent productions in CNF
$S \rightarrow ABA$	$S \rightarrow A C_1, C_1 \rightarrow BA$
$S \rightarrow BA \mid AB \mid AA \mid C_a A \mid C_b A \mid aB \mid C_b B \mid C_a A$	$S \rightarrow BA \mid AB \mid AA \mid C_a A \mid C_b A \mid a \mid b \mid C_b B \mid C_a A$
$A \rightarrow C_a A \mid C_b A \mid a \mid b$	$A \rightarrow C_a A \mid C_b A \mid a \mid b$
$B \rightarrow C_b B \mid C_a A \mid b \mid a$	$B \rightarrow C_b B \mid C_a A \mid b \mid a$
$C_a \rightarrow a$	$C_a \rightarrow a$
$C_b \rightarrow b$	$C_b \rightarrow b$

## Q. 21 Obtain leftmost derivation, rightmost derivation and derivation tree for the string "cccbaccba".

The grammar is  $S \rightarrow SS \mid a \mid SSb \mid c$ 

Dec. 2014

Ans. :

Derivation tree :

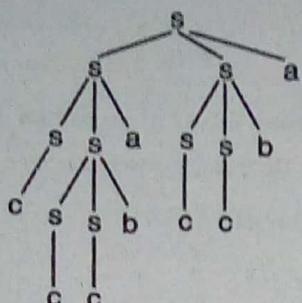


Fig. 5.8

Left most derivation	Right most derivation
$S \rightarrow SSa$	$S \rightarrow SSa$
$\rightarrow SSaSa$	$\rightarrow SSSba$
$\rightarrow cSaSa$	$\rightarrow SScba$
$\rightarrow cSSbaSa$	$\rightarrow Sccba$
$\rightarrow ccSbaSa$	$\rightarrow SSaccba$
$\rightarrow cccbaSa$	$\rightarrow SSSbaccba$
$\rightarrow cccbaSSba$	$\rightarrow SScbaccba$
$\rightarrow cccbacSba$	$\rightarrow Scbaccba$
$\rightarrow cccbaccba$	$\rightarrow cccbaccba$

Q. 22 Convert following grammar to CNF and GNF.

$$\begin{aligned} S &\rightarrow ASB \mid a \mid bb \\ A &\rightarrow aSA \mid a \\ B &\rightarrow SbS \mid bb \end{aligned}$$

Ans. :

$$\begin{aligned} S &\rightarrow ASB \mid a \mid bb \\ A &\rightarrow aSA \mid a \\ B &\rightarrow SbS \mid bb \end{aligned}$$

Converting to CNF :

Re-writing the grammar, we get,

$$\begin{aligned} S &\rightarrow ASB \mid a \mid V_1V_1 \\ A &\rightarrow V_2SA \mid a \\ B &\rightarrow SV_1S \mid V_1V_1 \\ V_1 &\rightarrow b \\ V_2 &\rightarrow a \end{aligned}$$

Now, re-writing each production in its equivalent CNF form, we get,

Productions	CNF forms
$S \rightarrow ASB$	$S \rightarrow AV_3, V_3 \rightarrow SB$
$S \rightarrow a$	$S \rightarrow a$

$$\begin{array}{ll} S \rightarrow V_1V_1 & S \rightarrow V_1V_1 \\ A \rightarrow V_2SA \mid a & S \rightarrow V_2V_4, V_4 \rightarrow SA \\ & A \rightarrow a \\ B \rightarrow SV_1S \mid V_1V_1 & B \rightarrow SV_5, V_5 \rightarrow V_1S \\ & B \rightarrow V_1V_1 \\ V_1 \rightarrow b & V_1 \rightarrow b \\ V_2 \rightarrow a & V_2 \rightarrow a \end{array}$$

Converting to GNF :

Step 1 : Substituting symbols, we get,

$$\begin{aligned} S &\rightarrow ASB \mid a \mid bX_1 \\ A &\rightarrow aSA \mid a \\ B &\rightarrow SX_2S \mid bX_1 \\ X_1 &\rightarrow b \\ X_2 &\rightarrow a \end{aligned}$$

Step 2 : Re-writing production in GNF :

Productions	CNF forms
(1) $X_1 \rightarrow b$	$X_1 \rightarrow b$
(2) $X_2 \rightarrow a$	$X_2 \rightarrow a$
(3) $A \rightarrow aSA \mid a$	$A \rightarrow aSA \mid a$
(4) $S \rightarrow ASB \mid a \mid bX_1$	$S \rightarrow aSASB \mid aSB [substituting A]$ $S \rightarrow a \mid bX_1$
(5) $B \rightarrow SX_2S \mid bX_1$	$S \rightarrow aSASBX_2S \mid aSBX_2S \mid aX_2S \mid bX_1X_2S$ [substituting for S] $S \rightarrow bX$

Q. 23 Consider the following grammar  $G = (V, T P, S)$ ,  $V = (S, X)$ ,  $T = \{0, 1\}$  and productions P are

$$S \rightarrow 0 \mid 0X1 \mid 01S1$$

$$X \rightarrow 0XX1 \mid 1S$$

S is start symbol. Show that above grammar is ambiguous.

Dec. 2015

Ans. :

A grammar is said to be ambiguous grammar if the language generated by the grammar contains some strings that has 2 parse trees.

Ex. : Let us consider the given grammar

$$S \rightarrow 0 \mid 0X1 \mid 01S1$$

$$X \rightarrow 0XX1 \mid 1S$$

where, S is the start symbol.

A string 010011 is generated by the given grammar.

The grammar generates the string 010011 in 2 different ways. The 2 derivations are shown in Fig. 1(a)-Q. 61 and Fig. 1(b)-Q. 61. As the same string has 2 different parse trees. The given grammar is ambiguous grammar.



$$B_2 \rightarrow aA_1B_2A_1 | bB_2A_1 | aA_1A_1 | bA_1$$

The final set of productions is :

$$A_2 \rightarrow aA_1B_2 | bB_2 | aA_1 | b$$

$$A_1 \rightarrow aA_1B_2A_2 | bB_2A_2 | aA_1A_2 | bA_2 | a$$

A set of productions P

$$B_2 \rightarrow aA_1B_2A_1B_2 | bB_2A_1B_2 | aA_1A_1B_2 | bA_1B_2 | aA_1B_2A_1 | bB_2A_1 | aA_1A_1 | bA_1$$

where, Set of variables V =  $(A_1, A_2, B_2)$

Set of terminals T =  $(a, b)$

Start symbol =  $A_1$

Set of productions P = Given above.

**Q. 26 Consider the following grammar :**

$$S \rightarrow ICtS | ICtSeS | a$$

$$C \rightarrow b$$

For the string 'ibtibtaea' find the following :

- (i) Leftmost derivation
- (ii) Rightmost derivation
- (iii) Parse tree
- (iv) Check if above grammar is ambiguous.

Dec. 2017

Ans. :

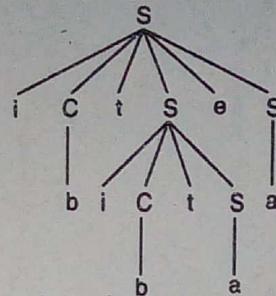
(i) Leftmost derivation

$$\begin{aligned} S &\Rightarrow ICtSeS \xrightarrow{C \rightarrow b} ibtSeS \xrightarrow{S \rightarrow iCtS} \\ &\quad ibtICtSeS \xrightarrow{C \rightarrow b} ibtibtSeS \xrightarrow{S \rightarrow a} ibtibtaeS \\ &\Rightarrow ibtibtaea \end{aligned}$$

(ii) Rightmost derivation

$$\begin{aligned} S &\Rightarrow ICtSeS \xrightarrow{S \rightarrow a} ICtSeA \xrightarrow{S \rightarrow iCtS} \\ &\quad iCtiCtSeA \xrightarrow{S \rightarrow a} iCtiCtaea \\ &\xrightarrow{C \rightarrow b} iCtibtaea \xrightarrow{C \rightarrow b} ibtibtaea \end{aligned}$$

(iii) Parse tree



(iv) It is an ambiguous grammar due to laughing if problem.

**Q. 27 Reduce following grammar to GNF.**

$$S \rightarrow AB$$

$$A \rightarrow BSBIBBib$$

$$B \rightarrow alaAb$$

$$(i) \quad S \rightarrow 01S101$$

$$S \rightarrow 10S110$$

$$S \rightarrow 001\epsilon$$

Ans. :

Removing  $\epsilon$ -production, we get,

$$S \rightarrow 01S101|10S110|00$$

It can be converted into GNF in an easy way by introducing two production

$$X \rightarrow 1 \text{ and } Y \rightarrow 0$$

∴ Productions in GNF are

$$S \rightarrow 0XS10X1YS11Y|0Y$$

$$X \rightarrow 1$$

$$Y \rightarrow 0$$

## Chapter 6 : Pushdown Automata (PDA)

**Q. 1 Distinguish between NPDA and DPDA.** Dec. 2005

Ans. :

Distinguish between NPDA and DPDA

A NPDA provides non-determinism to PDA.

In a DPDA there is only one move in every situation. Where as, in case of NPDA there could be multiple moves under a situation. DPDA is less powerful than NPDA.

Every context free language can not be recognized by a DPDA but it can be recognized by NPDA. The class of language a DPDA can accept lies in between a regular language and CFL. A palindrome can be accepted by NPDA but it can not be accepted by a DPDA

Q. 2 Design a PDA to accept  $(bdb)^n c^n$ .

Ans. :

To solve this problem, we can take a stack symbol  $x$ . For every 'bdb', one  $x$  will be pushed on top of the stack. After reading  $(bdb)^n$ , the stack should contain  $n$  number of  $x$ 's. These  $x$ 's will be matched with  $c$ 's. The transitions for the PDA accepting through an empty stack are given in Fig. 6.1.

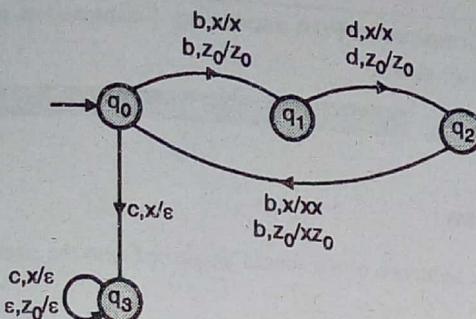


Fig. 6.1

A cycle through  $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_0$  traces a group of  $bdb$ .

The PDA  $M = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi\}$

Where,

$$Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{b, d, c\}, \Gamma = \{x, z_0\}$$

$q_0$  is the initial state,  $z_0$  is initial stack symbol.

The transition function  $\delta$  is given by,

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

$$\delta(q_0, b, x) = (q_1, x)$$

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

$$\delta(q_1, d, x) = (q_2, x)$$

$$\delta(q_2, b, z_0) = (q_0, xz_0)$$

$$\delta(q_2, b, x) = (q_0, xx)$$

$$\delta(q_0, c, x) = (q_3, \epsilon)$$

$$\delta(q_3, c, x) = (q_3, \epsilon)$$

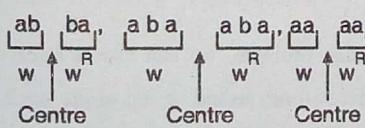
$$\delta(q_3, \epsilon, z_0) = (q_3, \epsilon) \text{ Accept through empty stack.}$$

Q. 3 Design a PDA for detection of even palindrome over  $\{a, b\}$ .

Dec. 2005, May 2006, May 2007, May 2016

Ans. :

An even palindrome will be of the form  $ww^R$



If the length of  $w$  is  $n$  then a palindrome of even length is :

First  $n$  characters are equal to the last  $n$  characters in the reverse order.

The character immediately before the middle position will be identical to the character immediately after the middle position.

## Algorithm :

There is no way of finding the middle position by a PDA; therefore the middle position is fixed non-deterministically.

1. First  $n$  characters are pushed onto the stack.  $n$  is non-deterministic.
2. The  $n$  characters on the stack are matched with the last  $n$  characters of the input string.
3.  $n$  is decided non-deterministically. Every character out of first  $n$  characters, whose previous character is same as itself should be considered for two cases :

- (a) It is first character of the second half.

- Pop the current stack symbol using the transitions :

$$\delta(q_0, a, a) \Rightarrow (q_1, \epsilon)$$

$$\delta(q_0, b, b) \Rightarrow (q_1, \epsilon)$$

Must be identical

- (b) It belongs to first half.

- Push the current input

$$\delta(q_0, a, \epsilon) \Rightarrow (q_0, a)$$

$$\delta(q_0, b, \epsilon) \Rightarrow (q_0, b)$$

4.  $n$  is decided non-deterministically. Every character out of first  $n$  characters, whose previous character is not same as itself should be pushed onto the stack.

- Push the current symbol using the transitions :

$$\delta(q_0, a, b) \Rightarrow (q_0, ab)$$

$$\delta(q_0, b, a) \Rightarrow (q_0, ba)$$

The transition table for the PDA is given below :

$$\delta(q_0, a, z_0) \Rightarrow \{(q_0, az_0)\}$$

$$\delta(q_0, b, z_0) \Rightarrow \{(q_0, bz_0)\}$$

$$\delta(q_0, a, a) \Rightarrow \{(q_0, aa) (q_1, \epsilon)\}$$

$$\delta(q_0, a, b) \Rightarrow \{(q_0, ab)\}$$

$$\delta(q_0, b, a) \Rightarrow \{(q_0, ba)\}$$

$$\delta(q_0, b, b) \Rightarrow \{(q_0, bb), (q_1, \epsilon)\}$$

$$\delta(q_1, a, a) \Rightarrow \{(q_1, \epsilon)\}$$

$$\delta(q_1, b, b) \Rightarrow \{(q_1, \epsilon)\}$$

$$\delta(q_1, \epsilon, z_0) \Rightarrow \{(q_1, \epsilon)\} \text{ [Accept through an empty stack]}$$

Where,

$$\text{the set of states } Q = \{q_0, q_1\}$$

$$\text{the set of input symbols } \Sigma = \{a, b\}$$

$$\text{the set of stack symbols } \Gamma = \{a, b, z_0\}$$

$$\text{Starting state} = q_0$$

$$\text{Initial stack symbol} = z_0$$

**Q. 4 Construct a PDA equivalent to the following CFG.**

$S \rightarrow 0BB$

$\rightarrow 0S \mid 1S \mid 0$

**Test if  $010^4$  is in the language****[May 2006, May 2011, May 2012]****Ans. :**

The equivalent PDA, M is given by

$M = (\{q\}, \{0, 1\}, \{0, 1, S, B\}, \delta, q, S, \phi)$ ,

where  $\delta$  is given by

$\delta(q, \epsilon, S) \Rightarrow \{(q, 0BB)\}$	For each production
$\delta(q, \epsilon, B) \Rightarrow \{(q, 0S), (q, 1S), (q, 0)\}$	in the given grammar
$\delta(q, 0, 0) \Rightarrow \{(q, \epsilon)\}$	
$\delta(q, 1, 1) \Rightarrow \{(q, \epsilon)\}$	For each terminal

**Acceptance of  $010^4$  by M :**

$\delta(q, 010000, S) \quad \delta(q, \epsilon, S) = (q, 0BB)$	$\longrightarrow (q, 010000, 0BB)$
$\delta(q, 0, 0) = (q, \epsilon)$	$\longrightarrow (q, 10000, BB)$
$\delta(q, \epsilon, B) = (q, 1S)$	$\longrightarrow (q, 10000, 1SB)$
$\delta(q, 1, 1) = (q, \epsilon)$	$\longrightarrow (q, 0000, SB)$
$\delta(q, \epsilon, S) = (q, 0BB)$	$\longrightarrow (q, 0000, 0BBBB)$
$\delta(q, 0, 0) = (q, \epsilon)$	$\longrightarrow (q, 000, BBB)$
$\delta(q, \epsilon, B) = (q, 0)$	$\longrightarrow (q, 000, 0BB)$
$\delta(q, 0, 0) = (q, \epsilon)$	$\longrightarrow (q, 00, BB)$
$\delta(q, \epsilon, B) = (q, 0)$	$\longrightarrow (q, 00, 0B)$
$\delta(q, 0, 0) = (q, \epsilon)$	$\longrightarrow (q, 0, B)$
$\delta(q, \epsilon, B) = (q, 0)$	

$\longrightarrow (q, 0, 0)$

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

$\longrightarrow (q, \epsilon, \epsilon)$

Thus the string  $010^4$  is accepted by M using an empty stack.

$\therefore 010^4 \in L$

**Q. 5 Construct a PDA accepting { anbman | m, n ≥ 1 } by null store.****[Dec. 2006, Dec. 2010, May 2012, May 2013]****Ans. :****Algorithm :**

1. The sequence of 'a's should be pushed onto the stack in state  $q_0$
2. On first 'b', the machine moves to  $q_1$  and remains there for 'b's. 'b' will have no effect on the stack.
3. For every 'a', an 'a' is erased from the stack.

The PDA accepting through empty stack is given by

$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \phi)$

Where the transition function  $\delta$  is :

1.  $\delta(q_0, a, z_0) = (q_0, az_0)$  [First 'a' is pushed]
2.  $\delta(q_0, a, a) = (q_0, aa)$  [Subsequent 'a's are pushed]
3.  $\delta(q_0, b, a) = (q_1, a)$  [Input symbols 'b's are skipped]
4.  $\delta(q_1, b, a) = (q_1, a)$
5.  $\delta(q_1, a, a) = (q_2, \epsilon)$  [An 'a' is erased on first 'a' of last 'a's]
6.  $\delta(q_2, a, a) = (q_2, \epsilon)$  [An 'a' is erased on subsequent 'a's of last 'a's]
7.  $\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$  [Accepting through empty stack]

**Q. 6 Design a PDA to accept  $(ab)^n(cd)^n$ .****[May 2007]****Ans. :**

To solve this problem, we can take a stack symbol x. For every 'ab', one x will be pushed on top of the stack. After reading  $(ab)^n$ , the stack should contain n number of x's. These x's will be matched with  $(cd)^n$ . For every 'cd' one x will be popped.

The transitions for the PDA accepting through an empty stack are given in Fig. 6.2.

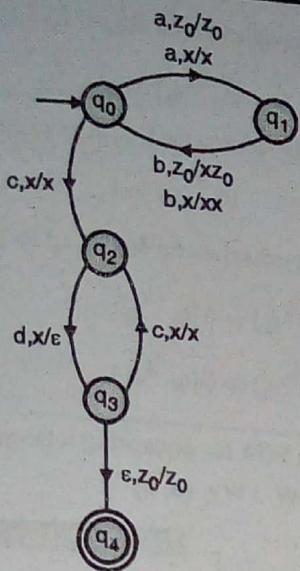


Fig. 6.2

PDA accepts through the final state  $q_4$ .

The PDA  $M = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$

Where,

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

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

$$\Gamma = \{x, z_0\}$$

The transition function  $\delta$  is given by,

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

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

$$\delta(q_1, b, z_0) = (q_0, x z_0)$$

$$\delta(q_1, b, x) = (q_0, xx)$$

$$\delta(q_0, c, x) = (q_2, x)$$

$$\delta(q_2, d, x) = (q_3, \epsilon)$$

$$\delta(q_3, c, x) = (q_2, x)$$

$$\delta(q_2, \epsilon, z_0) = (q_4, z_0)$$

$q_0$  is initial state,

$z_0$  is initial stack symbol.

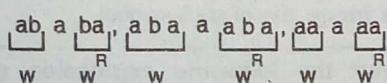
Set of final states  $F = \{q_4\}$

Q. 7 Design a PDA for detection of odd palindrome over  $\{a, b\}$ . Dec. 2007

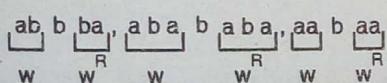
Ans. :

An odd palindrome will be of the form :

1.  $waw^R$



2.  $wbw^R$



If the length of  $w$  is  $n$  then a palindrome of odd length is :

First  $n$  characters are equal to the last  $n$  characters in reverse order with middle character as 'a' or 'b'.

#### Algorithm :

There is no way of finding the middle position of a string by a PDA, therefore the middle position is fixed non-deterministically.

1. First  $n$  characters are pushed onto the stack, where  $n$  is non-deterministic.
2. The  $n$  characters on the stack are matched with the last  $n$  characters of the input string.
3.  $n$  is decided non-deterministically. Every character out of first  $n$  characters should be considered for two cases :
  - (a) It is not the middle character – push the current character using the transition :

$$\delta(q_0, a, \epsilon) \Rightarrow (q_0, a)$$

$$\delta(q_0, b, \epsilon) \Rightarrow (q_0, b)$$

- (b) It is a middle character – go for matching of second half with the first half.

$$\delta(q_0, a, \epsilon) \Rightarrow (q_1, \epsilon)$$

$$\delta(q_0, b, \epsilon) \Rightarrow (q_1, \epsilon)$$

The status of the stack and the state of the machine is shown in the Fig. 6.3. Input applied is ababa.

Left child  $\rightarrow$  current input is taken as the middle character

Right child  $\rightarrow$  current input is not a middle character.

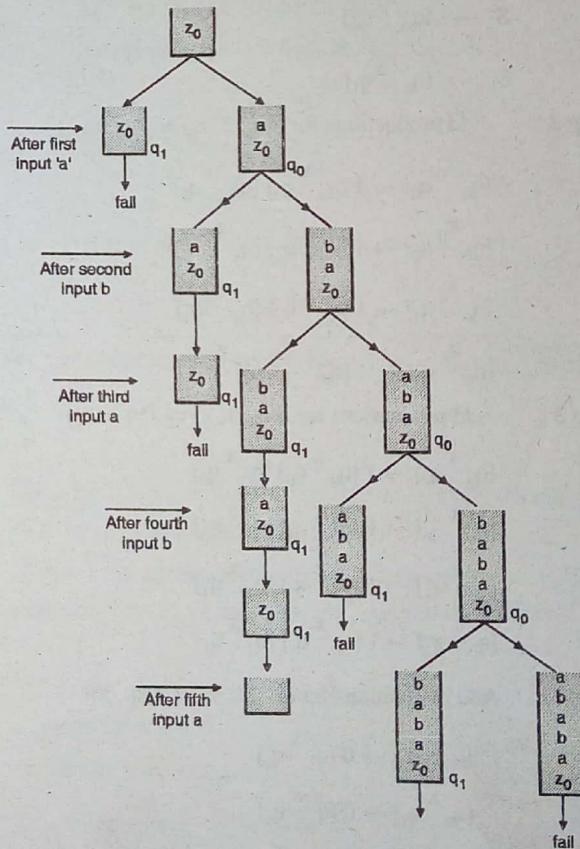


Fig. 6.3 : Processing of string by the PDA. String is taken as "ababa"

The transition table for the PDA is given below,

$$\delta(q_0, a, \epsilon) \Rightarrow \{(q_1, \epsilon), (q_0, a)\}$$

$\epsilon$  - indicates that irrespective of the current stack symbol, perform the transition.

$$\delta(q_0, b, \epsilon) \Rightarrow \{(q_1, \epsilon), (q_0, b)\}$$

$$\delta(q_1, a, a) \Rightarrow \{(q_1, \epsilon)\}$$

$$\delta(q_1, b, b) \Rightarrow \{(q_1, \epsilon)\}$$

$$\delta(q_1, \epsilon, z_0) \Rightarrow \{(q_1, \epsilon)\} \text{ [Accept through an empty stack]}$$

Where, The set of states  $Q = \{q_0, q_1\}$

The set input alphabet  $\Sigma = \{a, b\}$

The set of stack symbols  $\Gamma = \{a, b, z_0\}$

Starting state  $= q_0$

Initial stack symbol  $= z_0$

**Q. 8 Give the CFG generating the language accepted by the following PDA :  $M = (\{q_0, q_1\}, \{0, 1\}, \{z_0, x\}, \delta, q_0, z_0, \phi)$  when  $\delta$  is given below :**

$\delta(q_0, 1, z_0) = \{(q_0, xz_0)\}$   $\delta(q_0, 1, x) = \{(q_0, xx)\}$   
 $\delta(q_0, 0, x) = \{(q_1, x)\}$   $\delta(q_0, \epsilon, z_0) = \{(q_0, \epsilon)\}$   
 $\delta(q_1, 1, x) = \{(q_1, \epsilon)\}$   $\delta(q_1, 0, z_0) = \{(q_0, z_0)\}$

Déc. 2007.

**Ans. :**

**Step 1 : Add productions for the start symbol**

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

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

**Step 2 : Add productions for  $\delta(q_0, 1, z_0) = \{(q_0, xz_0)\}$**

$$[q_0 \ z_0 \ q_0] \xrightarrow{1} [q_0 \ x \ q_0] [q_0 \ z_0 \ q_0]$$

$$[q_0 \ z_0 \ q_0] \xrightarrow{1} [q_0 \ x \ q_1] [q_1 \ z_0 \ q_0]$$

$$[q_0 \ z_0 \ q_1] \xrightarrow{1} [q_0 \ x \ q_0] [q_0 \ z_0 \ q_1]$$

$$[q_0 \ z_0 \ q_1] \xrightarrow{1} [q_0 \ x \ q_1] [q_1 \ z_0 \ q_1]$$

**Step 3 : Add productions for  $\delta(q_0, 1, x) \Rightarrow \{(q_0, xx)\}$**

$$[q_0 \ x \ q_0] \xrightarrow{1} [q_0 \ x \ q_0] [q_0 \ x \ q_0]$$

$$[q_0 \ x \ q_0] \xrightarrow{1} [q_0 \ x \ q_1] [q_1 \ x \ q_0]$$

$$[q_0 \ x \ q_1] \xrightarrow{1} [q_0 \ x \ q_0] [q_0 \ x \ q_1]$$

$$[q_0 \ x \ q_1] \xrightarrow{1} [q_0 \ x \ q_1] [q_1 \ x \ q_1]$$

**Step 4 : Add productions for  $\delta(q_0, 0, x) \Rightarrow \{(q_1, x)\}$**

$$[q_0 \ x \ q_0] \xrightarrow{0} [q_1 \ x \ q_0]$$

$$[q_0 \ x \ q_0] \xrightarrow{0} [q_1 \ x \ q_1]$$

**Step 5 : Add productions for  $\delta(q_0, \epsilon, z_0) = \{(q_1, \epsilon)\}$**

$$[q_0 \ z_0 \ q_1] \xrightarrow{\epsilon} \epsilon$$

**Step 6 : Add production for  $\delta(q_1, 1, x) \Rightarrow \{(q_1, \epsilon)\}$**

$$[q_1 \ x \ q_1] \xrightarrow{1} \epsilon$$

**Step 7 : Add productions for  $\delta(q_1, 0, z_0) \Rightarrow \{(q_0, z_0)\}$**

$$[q_1 \ z_0 \ q_0] \xrightarrow{0} [q_0 \ z_0 \ q_0]$$

$$[q_1 \ z_0 \ q_0] \xrightarrow{0} [q_0 \ z_0 \ q_1]$$

### Q. 9 Design a PDA for accepting a language

$$L = \{ WcW^T \mid W \in \{a, b\}^* \}$$

May 2008, May 2010, May 2011

**Ans. :**

$W^T$  stands for reverse of  $W$ . A string of the form  $WcW^T$  is an odd length palindrome with the middle character as  $c$ .

**Algorithm :**

If the length of the string is  $2n + 1$ , then the first  $n$  symbols should be matched with the last  $n$  symbols in the reverse order. A stack can be used to reverse the first  $n$  input symbols.

Status of the stack and state of the machine is shown in Fig. 6.4. Input applied is  $abbcbba$ .

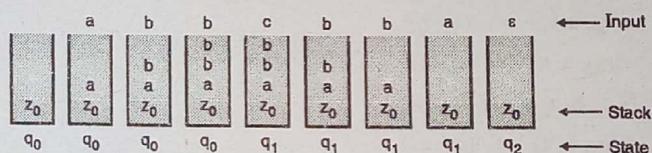


Fig. 6.4 : A PDA on input abbcbba

The PDA accepting through final state is given by

$$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_2\})$$

Where the transition function  $\delta$  is given below :

1.  $\delta(q_0, a, \epsilon) = (q_0, a)$  First  $n$  symbols are pushed onto the stack
2.  $\delta(q_0, b, \epsilon) = (q_0, b)$
3.  $\delta(q_0, c, \epsilon) = (q_1, \epsilon)$  [State changes on c]
4.  $\delta(q_1, a, a) = (q_1, \epsilon)$
5.  $\delta(q_1, b, b) = (q_1, \epsilon)$  Last  $n$  symbols are matched with first  $n$  symbols in reverse order
6.  $\delta(q_1, \epsilon, z_0) = (q_2, z_0)$  [Accepted through final state]

A transition of the form  $\delta(q_0, a, \epsilon) = (q_0, a)$  implies that always push  $a$ , irrespective of stack symbol.

**Q. 10 Convert the following expression grammar to PDA I → a | b | Ia | Ib | I0 | I1 E → I | E \* | E | E \* | E | E**

Dec. 2008

**Ans. :**

The equivalent PDA,  $M$  is given by,

$M = \{(\{q\}, \{0, 1, a, b, *, +, (, )\}, \{0, 1, a, b, *, +, (, ), I, E\}, \delta, q, E, \phi)$   
where,  $\delta$  is given by,

$$\begin{aligned}\delta(q, \epsilon, E) &= \{(q, I), (q, E * E), (q, E + E), (q, (E))\} \\ \delta(q, \epsilon, I) &= \{(q, a), (q, b), (q, Ia), (q, I0), (q, II)\} \\ \delta(q, 0, 0) &= \{(q, \epsilon)\} \\ \delta(q, 1, 1) &= \{(q, \epsilon)\} \\ \delta(q, a, a) &= \{(q, \epsilon)\} \\ \delta(q, b, b) &= \{(q, \epsilon)\} \\ \delta(q, +, +) &= \{(q, \epsilon)\} \\ \delta(q, *, *) &= \{(q, \epsilon)\} \\ \delta(q, (, )) &= \{(q, \epsilon)\} \\ \delta(q, ), )) &= \{(q, \epsilon)\}\end{aligned}$$

**Q. 11** Design a PDA for CFL that checks the well formedness of parenthesis i.e. the language  $L$  of all "balanced" string of two types of parenthesis say "( )" and "[ ]". Trace the sequence of moves made corresponding to input string (([ ]) [ ]). May 2009, May 2014, Dec 2017

**Ans. :**

The transition function of the PDA is given below :

- |  |   |
|--|---|
| 1. $\delta(q_0, (, z_0) = (q_0, (z_0)$       | Push the opening bracket '(                   |
| 2. $\delta(q_0, (, () = (q_0, (($            |   |
| 3. $\delta(q_0, (, [ ) = (q_0, ([ )$         |   |
| 4. $\delta(q_0, [, z_0) = (q_0, [z_0)$       | Push the opening bracket '['                  |
| 5. $\delta(q_0, [, () = (q_0, [($            |   |
| 6. $\delta(q_0, [, [ ) = (q_0, [[ )$         |   |
| 7. $\delta(q_0, ), )) = (q_0, \epsilon)$     | POP an opening bracket for a closing bracket. |
| 8. $\delta(q_0, ], ]) = (q_0, \epsilon)$     |   |
| 9. $\delta(q_0, \epsilon, z_0) = (q_f, z_0)$ | Accept through a final state.                 |

Simulation of PDA for the input string (([]))

$$\begin{aligned}(q_0, (([])), z_0) &\xrightarrow{\text{Rule 1}} (q_0, ([])), ((z_0)) \\ &\xrightarrow{\text{Rule 2}} (q_0, ([])), (((z_0))) \\ &\xrightarrow{\text{Rule 5}} (q_0, ([])), (((z_0))) \\ &\xrightarrow{\text{Rule 8}} (q_0, (\epsilon)), (((z_0)))\end{aligned}$$

$$\begin{aligned}&\xrightarrow{\text{Rule 7}} (q_0, ([])), (z_0) \\ &\xrightarrow{\text{Rule 5}} (q_0, ([])), ((z_0)) \\ &\xrightarrow{\text{Rule 8}} (q_0, (\epsilon)), (z_0) \\ &\xrightarrow{\text{Rule 7}} (q_0, \epsilon, z_0) \\ &\xrightarrow{\text{Rule 9}} (q_f, \epsilon, z_0)\end{aligned}$$

**Q. 12** Consider the PDA with the following moves :  
 $\delta(q_0, a, z_0) = \{(q_0, az_0)\}$   $\delta(q_0, a, a) = \{(q_0, aa)\}$   $\delta(q_0, b, a) = \{(q_1, \epsilon)\}$   $\delta(q_1, b, a) = \{(q_1, \epsilon)\}$   $\delta(q_1, \epsilon, z_0) = \{(q_1, \epsilon)\}$  Obtain CFG equivalent to PDA.

May 2009

**Ans. :**

**Step 1 :** Add productions for the start symbol.

$$S \rightarrow [q_0]^{z_0} q_0$$

$$S \rightarrow [q_0]^{z_0} q_1$$

**Step 2 :** Add productions for  $(q_0, a, a) = \{(q_0, aa)\}$

$$[q_0]^{a} q_0 \rightarrow a [q_0]^{a} q_0 [q_0]^{a} q_0$$

$$[q_0]^{a} q_0 \rightarrow a [q_0]^{a} q_1 [q_1]^{a} q_0$$

$$[q_0]^{a} q_1 \rightarrow a [q_0]^{a} q_0 [q_0]^{a} q_1$$

$$[q_0]^{a} q_1 \rightarrow a [q_0]^{a} q_1 [q_1]^{a} q_1$$

**Step 3 :** Add productions for  $\delta(q_0, b, a) = \{(q_1, \epsilon)\}$

$$[q_0]^{a} q_1 \rightarrow b$$

**Step 4 :** Add productions for  $\delta(q_1, b, a) = \{(q_1, \epsilon)\}$

$$[q_1]^{a} q_1 \rightarrow b$$

**Step 5 :** Add productions for  $\delta(q_1, \epsilon, z_0) \rightarrow \{(q_1, \epsilon)\}$

$$[q_1]^{z_0} q_1 \rightarrow \epsilon$$

**Q. 13** Write short note on DPDA. Dec. 2009

**Ans. :**

**DPDA**

In a DPDA there is only one move in every situation. A DPDA is less powerful than NPDA.

Every context free language cannot be accepted by a DPDA. For example, a string of the form  $ww^R$  can not be processed by a DPDA.

The class of a language a DPDA can accept lies in between a regular language and CFL.

A DPDA is defined as :

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F), \text{ where}$$

$\delta(q, a, x)$  has one move for any  $q \in Q, X \in \Gamma$  and  $a \in \Sigma$ .

**Q. 14 Design a PDA for detection of palindromes over {a, b}.**

Dec. 2012

**Ans. :**

A palindrome will be of the form :

1.  $ww^R$  ] even palindrome
2.  $waw^R$  ]
3.  $wbw^R$  ] odd palindrome

If the length of w is n then a palindrome is :

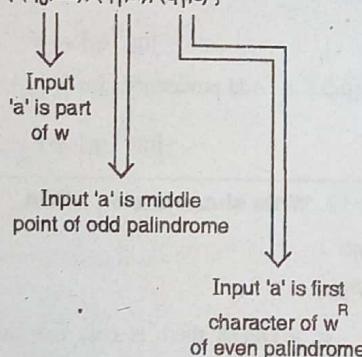
First n characters are equal to the last n characters in the reverse order with the middle character as :

- (1) a [ For odd palindrome ]
- (2) b [ For odd palindrome ]
- (3)  $\epsilon$  [ For even palindrome ]

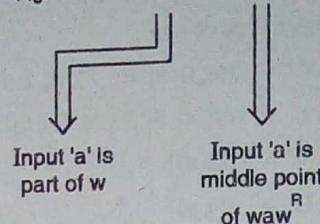
The transition table for the PDA is given below :

$$\begin{aligned}\delta(q_0, a, z_0) &\Rightarrow \{(q_1, z_0), (q_0, az_0)\} \\ \delta(q_0, b, z_0) &\Rightarrow \{(q_1, z_0), (q_0, bz_0)\} \\ \delta(q_0, a, a) &\Rightarrow \{(q_0, aa), (q_1, a), (q_1, \epsilon)\} \\ \delta(q_0, a, b) &\Rightarrow \{(q_0, ab), (q_1, b)\} \\ \delta(q_0, b, a) &\Rightarrow \{(q_0, ba), (q_1, a)\} \\ \delta(q_0, b, b) &\Rightarrow \{(q_0, bb), (q_1, b), (q_1, \epsilon)\} \\ \delta(q_1, a, a) &\Rightarrow \{(q_1, \epsilon)\} \\ \delta(q_1, b, b) &\Rightarrow \{(q_1, \epsilon)\} \\ \delta(q_1, \epsilon, z_0) &\Rightarrow \{(q_1, \epsilon)\}\end{aligned}$$

[Accept through an empty stack].

**Details of important transitions :**The transaction,  $\delta(q_0, a, a) \Rightarrow \{(q_0, aa), (q_1, a), (q_1, \epsilon)\}$ The transition rule for  $\delta(q_0, a, a)$ , must consider the three cases :

1. Input 'a' is part of w of the palindrome.
2. Input 'a' is middle character of  $w^R$ .
3. Input 'a' is the first character of  $w^R$ .

The transaction,  $\delta(q_0, a, b) \Rightarrow \{(q_0, ab), (q_1, b)\}$ 

**Q. 15 Write application of PDA.**

Dec. 2012

**Ans. :****Applications of PDA**

PDA is a machine for CFL.

A string belonging to a CFL can be recognized by a PDA.

PDA is extensively used for parsing.

PDA is an abstract machine; it can also be used for giving proofs of lemma on CFL.

**Q. 16 Design a PDA to accept language**

{ $a^{n-1} b^{2n+1} \mid n \geq 1$ }

Dec. 2014

**Ans. :**

For every 'a' in the input, 2 b's are pushed onto the stack.

For the first 'b' in the input, 2 b's are pushed onto the stack.

For every 'b' in the input, 1 'b' is popped out from the stack.

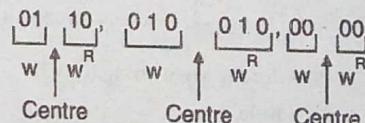
Finally the stack should become empty.

**Transitions**

$$\begin{aligned}\delta(q_0, a, z_0) &= (q_0, bbz_0) \\ \delta(q_0, a, b) &= (q_0, bbb) \\ \delta(q_0, b, z_0) &= (q_1, bbz_0) \\ \delta(q_0, b, b) &= (q_1, bbb) \\ \delta(q_1, b, b) &= (q_1, \epsilon) \\ \delta(q_1, \epsilon, z_0) &= (q_1, \epsilon)\end{aligned}$$

[Accept using empty stack]

**Q. 17 Design PDA to check even palindrome over  $\Sigma = \{0, 1\}$**

**Ans. :**An even palindrome will be of the form  $ww^R$ 

If the length of w is n then a palindrome of even length is :

First n characters are equal to the last n characters in the reverse order.

The character immediately before the middle position will be identical to the character immediately after the middle position.

**Algorithm :**

There is no way of finding the middle position by a PDA; therefore the middle position is fixed non-deterministically.

1. First  $n$  characters are pushed onto the stack.  $n$  is non-deterministic.
2. The  $n$  characters on the stack are matched with the last  $n$  characters of the input string.
3.  $n$  is decided non-deterministically. Every character out of first  $n$  characters, whose previous character is same as itself should be considered for two cases :

(a) It is first character of the second half.

- Pop the current stack symbol using the transitions :  
 $\delta(q_0, 0, 0) \Rightarrow (q_1, \epsilon)$   
 $\delta(q_0, 1, 1) \Rightarrow (q_1, \epsilon)$   
Must be identical

(b) It belongs to first half.

- Push the current input

$$\begin{aligned}\delta(q_0, 0, \epsilon) &\Rightarrow (q_0, 0) \\ \delta(q_0, 1, \epsilon) &\Rightarrow (q_0, 1)\end{aligned}$$

4.  $n$  is decided non-deterministically. Every character out of first  $n$  characters, whose previous character is not same as itself should be pushed onto the stack.

- Push the current symbol using the transitions :

$$\begin{aligned}\delta(q_0, 0, 1) &\Rightarrow (q_0, 01) \\ \delta(q_0, 1, 0) &\Rightarrow (q_0, 10)\end{aligned}$$

The transition table for the PDA is given below :

$$\begin{aligned}\delta(q_0, 0, z_0) &\Rightarrow \{(q_0, 0z_0)\} \\ \delta(q_0, 1, z_0) &\Rightarrow \{(q_0, 1z_0)\} \\ \delta(q_0, 0, 0) &\Rightarrow \{(q_0, 00)(q_1, \epsilon)\} \\ \delta(q_0, 0, 1) &\Rightarrow \{(q_0, 01)\} \\ \delta(q_0, 1, 0) &\Rightarrow \{(q_0, 10)\} \\ \delta(q_0, 1, 1) &\Rightarrow \{(q_0, 11), (q_1, \epsilon)\} \\ \delta(q_1, 0, 0) &\Rightarrow \{(q_1, \epsilon)\} \\ \delta(q_1, 1, 1) &\Rightarrow \{(q_1, \epsilon)\} \\ \delta(q_1, \epsilon, z_0) &\Rightarrow \{(q_1, \epsilon)\}\end{aligned}$$

[Accept through an empty stack]

Where,

$$\text{the set of states } Q = \{q_0, q_1\}$$

$$\text{the set of input symbols } \Sigma = \{0, 1\}$$

$$\text{the set of stack symbols } \Gamma = \{0, 1, z_0\}$$

$$\text{Starting state } = q_0$$

$$\text{Initial stack symbol } = z_0$$

- Q. 18 Design DPDA to accept language  $L = \{x \in \{a, b\}^* \mid N_a(x) > N_b(x), N_a(x) > N_b(x) \text{ means number of } a's \text{ are greater than number of } b's \text{ in string } x\}$**

Dec. 2015

**Ans. :**

The PDA is being designed to accept the string using final state. The stack is being used to store excess of  $a$ 's over  $b$ 's or excess of  $b$ 's over  $a$ 's out of input seen so far.

**Transitions**

1.  $\delta(q_0, a, z_0) = (q_0, a z_0)$  [Extra 'a' is pushed]
2.  $\delta(q_0, b, z_0) = (q_0, b z_0)$  [Extra 'b' is pushed]
3.  $\delta(q_0, a, a) = (q_0, aa)$  [Excess a's are pushed]
4.  $\delta(q_0, a, b) = (q_0, \epsilon)$  [Excess b's decreased by 1]
5.  $\delta(q_0, b, b) = (q_0, bb)$  [Excess b's are pushed]
6.  $\delta(q_0, b, a) = (q_0, \epsilon)$  [Excess a's decreased by 1]
7.  $\delta(q_0, \epsilon, a) = (q_1, \epsilon)$  [Input ends with excess a's on the stack]

The PDA is given by :

$$M = (\{q_0, q_1\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_1\})$$

- Q. 19 Construct PDA accepting the language  $L = \{a^{2n} b^n \mid n > 0\}$ .**

May 2016

**Ans. :****Algorithm :**

1. For every pair of leading  $a$ 's, one  $X$  is inserted in the stack.
2.  $X$ 's on the stack are matched with trailing  $b$ 's.

The PDA is given by

$$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{X, Z_0\}, \delta, q_0, Z_0, \emptyset)$$

where the transition function  $\delta$  is

1.  $\delta(q_0, a, Z_0) = (q_1, Z_0)$
2.  $\delta(q_1, a, Z_0) = (q_2, XZ_0)$
3.  $\delta(q_2, a, X) = (q_1, X)$
4.  $\delta(q_1, a, X) = (q_2, XX)$
5.  $\delta(q_2, b, X) = (q_3, \epsilon)$
6.  $\delta(q_3, b, X) = (q_3, \epsilon)$
7.  $\delta(q_3, \epsilon, Z_0) = (q_3, \epsilon)$

Accept through empty stack.

- Q. 20 Design a PDA corresponding to the grammar :**

$$S \rightarrow aSA \mid \epsilon$$

$$A \rightarrow bB$$

$$B \rightarrow b$$

Dec. 2016

**Ans. :**

The equivalent PDA,  $M$  is given by :

$$M = (\{q\}, \{a, b\}, \{a, b, S, A, B\}, \delta, q, S, \phi)$$

where  $\delta$  is given by :

$$\delta(q, \epsilon, S) \Rightarrow \{(q, aSA), (q, b)\}$$

$$\delta(q, \epsilon, A) \Rightarrow \{(q, bB)\}$$

$$\delta(q, \epsilon, B) = \{(q, b)\}$$

$$\delta(q, a, a) = \{(q, \epsilon)\}$$

$$\delta(q, b, b) = \{(q, \epsilon)\}$$

### Q. 21 Design a PDA to accept language

$$\{a^{n-1} b^{2n+1} \mid n \geq 1\}$$

Dec. 2017

**Ans. :**

$$1. \delta(q_0, a, Z_0) \Rightarrow (q_1, aaZ_0)$$

$$2. \delta(q_1, a, a) \Rightarrow (q_1, aa)$$

$$3. \delta(q_1, b, a) \Rightarrow (q_2, a)$$

$$4. \delta(q_2, b, a) \Rightarrow (q_1, \epsilon)$$

$$5. \delta(q_2, \epsilon, Z_0) \Rightarrow (q_2, \epsilon)$$

Accept through empty stack.

## Chapter 7 : Turing Machine (TM)

### Q. 1 Write short note on : Universal TM.

Dec. 2005, May 2007, Dec. 2007, May 2008, Dec. 2008,

May 2009, May 2010, Dec. 2011, May 2012,

Dec. 2012, Dec. 2015

**Ans. :**

### Universal TM

A general-purpose computer can be programmed to solve different types of problems. A TM can also behave like a general-purpose computer. A general purpose computer solves a problem as given below :

1. A program is written in a high level language and its machine-code is obtained with the help of a compiler.
2. Machine code is loaded in main memory.
3. Input to the program can also be loaded in memory.
4. Program stored in memory is executed line by line. Execution involves reading a line of code pointed by IP (instruction pointer), decoding the code and executing it.

We can follow a similar approach for a TM. Such, a TM is known as **Universal Turing Machine**. Universal Turing Machine (UTM) can solve all sorts of solvable problems.

A Turing machine  $M$  is designed to solve a particular problem  $p$ , can be specified as :

1. The initial state  $q_0$  of the TM  $M$ .
2. The transition function  $\delta$  of  $M$  can be specified as given :

If the current state of  $M$  is  $q_i$  and the symbol under the head is  $a_i$  then the machine moves to state  $q_j$  while changing  $a_i$  to  $a_j$ . The move of tape head may be :

1. To-left,
2. To-Right or
3. Neutral

Such a move of TM can be represented by tuple

$$\{(q_i, a_i, q_j, a_j, m_f) : q_i, q_j \in Q ; a_i, a_j \in \Gamma ; m_f \in \{\text{To-left, To-right, Neutral}\}\}$$

UTM should be able to simulate every turing machine.

Simulation of a Turing will involve :

1. Encoding behaviour of a particular TM as a program.
2. Execution of the above program by UTM.

A move of the form  $(q_i, a_i, q_j, a_j, m_f)$  can be represented as  $10^i 10^{i+1} 10^j 10^K$ ,

Where  $K = 1$ , if move is to the left

$K = 2$ , if move is to the right

$K = 3$ , if move is 'no-move'

State  $q_0$  is represented by 0,

State  $q_1$  is represented by 00,

State  $q_n$  is represented by  $0^{n+1}$ .

First symbol can be represented by 0,

Second symbol can be represented by 00 and so on.

Two elements of a tuple representing a move are separated by 1.

Two moves are separated by 11.

### Execution by UTM :

We can assume the UTM as a 3-tape turing machine.

1. Input is written on the first tape.
2. Moves of the TM in encoded form is written on the second tape.
3. The current state of TM is written on the third tape.

The control unit of UTM by counting number of 0's between 1's can find out the current symbol under the head. It can find the current state from the tape 3. Now, it can locate the appropriate move based on current input and the current state from the tape 2. Now, the control unit can extract the following information from the tape 2 :

1. Next state
2. Next symbol to be written
3. Move of the head.

Based on this information, the control unit can take the appropriate action.

**Q. 2 Design a TM which recognizes palindromes over alphabet {a,b}**

May 2006, May 2009, May 2014, Dec. 2017

**Ans. :**

A palindrome can have one of the following forms :

1.  $\omega\omega^R$
2.  $\omega\omega^R$
3.  $\omega b \omega^R$

Where  $\omega$  is a string over {a,b} with  $|\omega| \geq 0$

**Algorithm :**

1. Algorithm requires  $n$  cycles, where  $|\omega| = n$ .
2. In each cycle, first character is matched with the last character and both are erased.

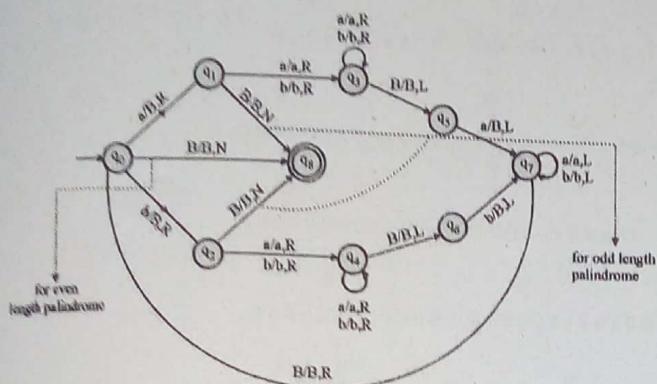


Fig. 7.1(a) : Transition diagram

If the leftmost character is 'a' the machine takes a path through  $q_0 \rightarrow q_1 \rightarrow q_3 \rightarrow q_5 \rightarrow q_7$ , looking for last character as 'a'.

If the leftmost character is 'b', the machine takes a path through

$q_0 \rightarrow q_2 \rightarrow q_4 \rightarrow q_6 \rightarrow q_7$ , looking for last character as 'b'.

The Turing machine M is given by :

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

where,  $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$

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

$\Gamma = \{a, b, B\}$

The transition function  $\delta$  is given in Fig. 7.1(a)

$q_0$  = initial state

B = blank symbol

F =  $\{q_8\}$ , halting state

Working of TM for input abbabba is shown in Fig. 7.1(a) :

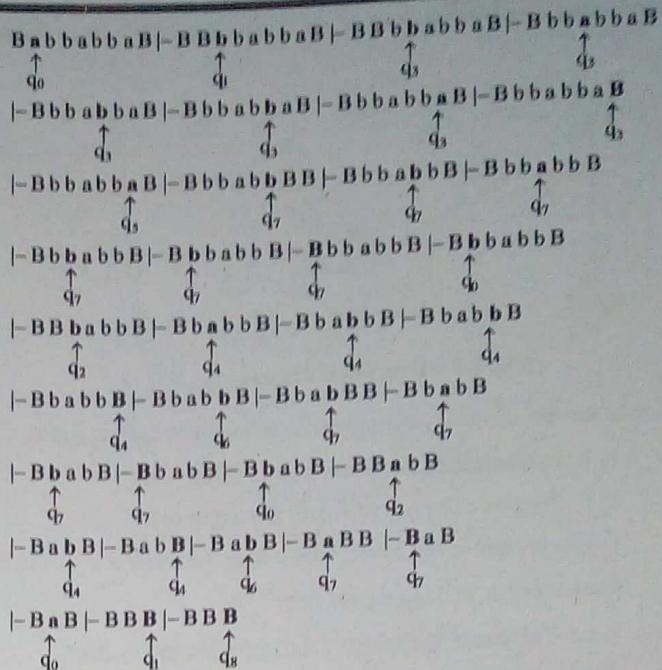


Fig. 7.1(a)

**Q. 3 Design a TM to compute multiplication of two unary numbers.**

May 2007

**Ans. :**

Multiplication algorithm is being explained with the help of an example.

$3 \times 5$  will require three cycles.

Initial	$B000 \# 00000 \# B$
	$\underbrace{3}_{\text{Product will be stored here}} \quad \underbrace{5}_{\text{}}$

Cycle 1	$BX00 \# 00000 \# 00000B$	$1 \times 5 = 5$
		$\underbrace{1 \times 5}_{= 5}$

Cycle 2	$BXX0 \# 00000 \# 00000000B$	$2 \times 5 = 10$
		$\underbrace{2 \times 5}_{= 10}$

Cycle 3	$BXXX \# 00000 \# 0000000000B$	$3 \times 5 = 15$
		$\underbrace{3 \times 5}_{= 15}$

To calculate  $3 \times 5$ , three times, 5 zero's are appended.

Unary representation of 3 is 000.

Unary representation of 5 is 00000.

3, 5 and the result, are separated by #.

Inside each major cycles (three cycles for 3), there will be a number of minor cycles (5 minor cycles for 5) to append 0's one at a time.

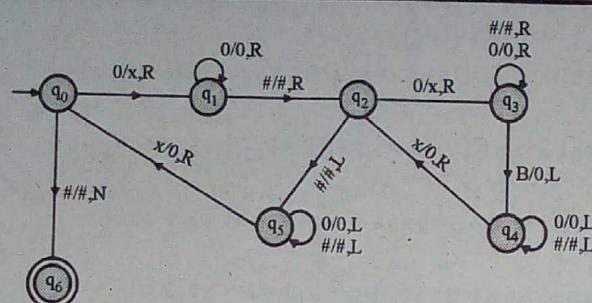


Fig. 7.2 : Transition diagram for TM

Let us assume that the two numbers to be multiplied are  $x_1$  and  $x_2$ .

$x_1$  is represented by  $\omega_1$ , where  $\omega_1$  is a string of 0's.

$x_2$  is represented by  $\omega_2$ , where  $\omega_2$  is a string of 0's.

$x_1 * x_2$  is represented by  $\omega_3$ , where  $\omega_3$  is a string of 0's.

# separates  $\omega_1$  and  $\omega_2$ ,  $\omega_2$  and  $\omega_3$ .

In the TM shown in Fig. Ex. 7.3.6, there are two cycles.

The cycle  $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_5 \rightarrow q_0$  appends  $\omega_2$  to  $\omega_3$  for every zero in  $\omega_1$ , with the help of cycle  $q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_2$ .

Working of TM for  $2 \times 2$  is shown in Fig. 7.2(a) :

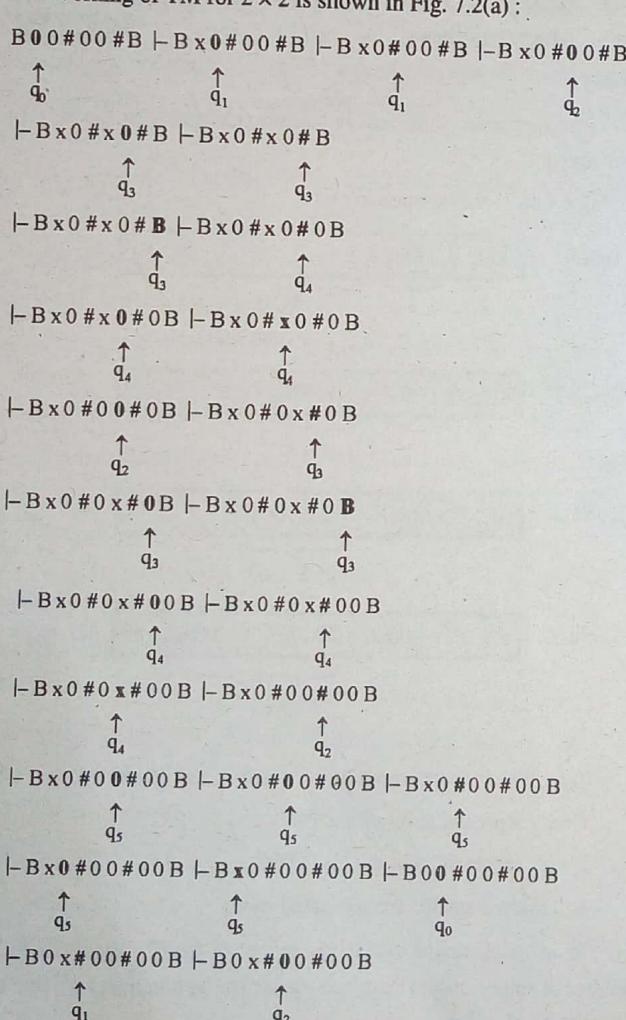


Fig. 7.2Contd...

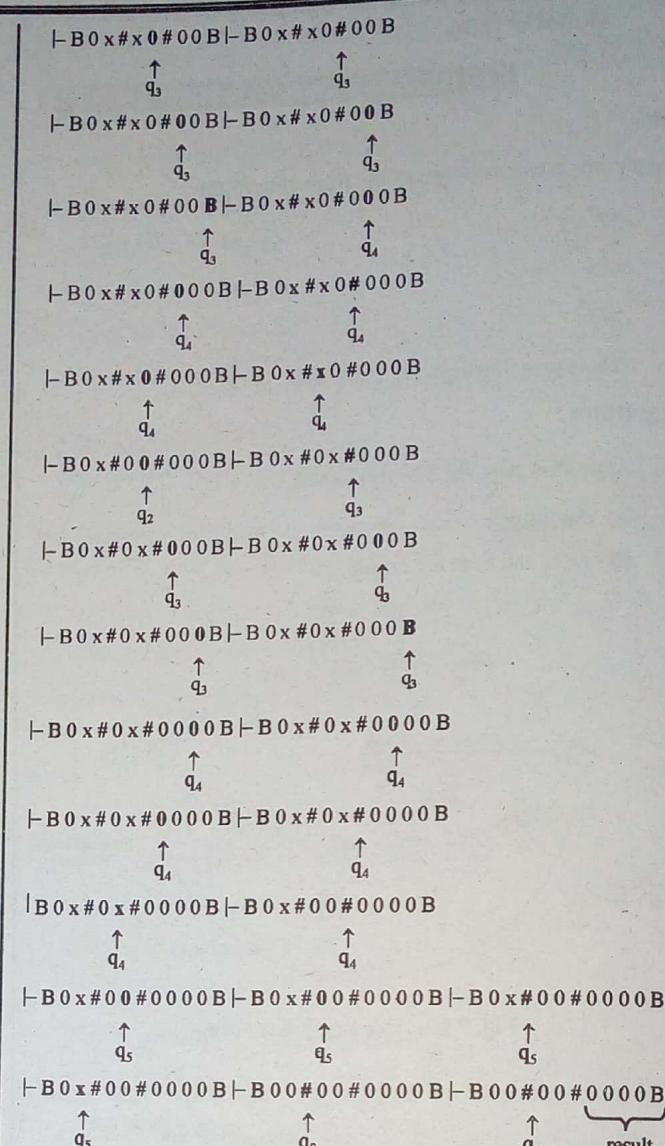


Fig. 7.2(a)

**Q. 4 Design a TM to find the value of  $\log_2(n)$ , where n is any binary number.**

Dec. 2007

**Ans. :**

$\log_2(n)$  of any number n lying between  $2^n$  and  $2^{n+1}$  is given by n.

i.e. if  $2^n \leq n < 2^{n+1}$ , then  $\log_2(n) = n$

Let us consider the case of a number

$$n = 36$$

$$2^5 \leq 36 < 2^6$$

Therefore,  $\log_2(36) = 5$

36 can be written as 100100.

Any number n satisfying the condition  $2^5 \leq n < 2^6$  can be written as 1XXXXX (where X stands for either 1 or 0).  $\log_2(1XXXXX)$  can be calculated by erasing the most significant bit 1 and renaming other bits as '0'. Unary representation of 5 is 00000.

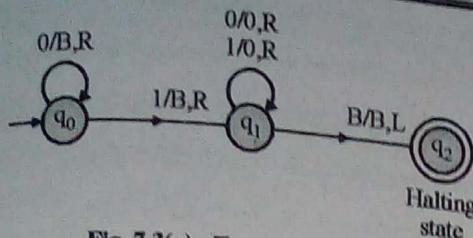


Fig. 7.3(a) : Transition diagram

	0	1	B	
$\rightarrow q_0$	$(q_0, B, R)$	$(q_1, B, R)$	-	
$q_1$	$(q_1, 0, R)$	$(q_1, 0, R)$	$(q_2, B, L)$	
$q_2^*$	$q_2$	$q_2$	$q_2$	← Halting state

Fig. 7.3(b) : Transition table

Working of TM for  $(36)_{10}$  is shown in Fig. 7.3(c) :  
 $(36)_{10} = (0100100)_2$

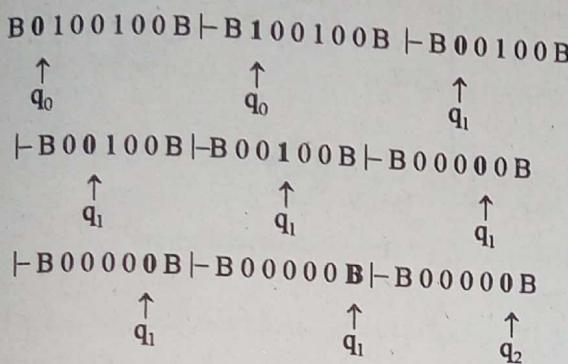


Fig. 7.3(c)

### Q. 5 Design a Turing machine to compute $n!$ .

Dec. 2008

**Ans. :**

It is assumed that  $n$  is represented in unary system.

Factorial of  $n$  can be calculated through repeated application of :

1. Multiplication
  2. Copy
- Operations.

Algorithm is being explained with the help of example.

Algorithm for  $|3|$ .

Initial configuration  $0 \# 0 0 0 \# B B \dots$

#### Cycle 1 :

1. Multiplication  $0 \# 0 0 0 \# 0 0 0 B \dots$

Product

2. Copy  $n - 1$ , i.e. 2  $0 \# 0 0 0 \# 0 0 0 \# 0 0$

$n \quad n \times 1 \quad n - 1$

#### Cycle 2 :

1. Multiplication  $0 \# 0 0 0 \# 0 0 0 \# 0 0 \# 0 0 0 0 0$

$n \quad n \times 1 \quad n - 1 \quad n \times (n - 1)$

2. Copy  $n - 2$ , i.e. 1  $0 \# 0 0 0 \# 0 0 0 \# 0 0 \# 0 0 0 0 0 \# 0$

$n \quad n \times 1 \quad n - 1 \quad n \times (n - 1) \quad n - 2$

#### Cycle 3 :

1.  $0 \# 0 0 0 \# 0 0 0 \# 0 0 \# 0 0 0 0 0 \# 0 \# 0 0 0 0 0 \#$

$n \quad 1 \times n \quad n - 1 \quad n \times (n - 1) \quad n - 2 \quad n \times (n - 1)(n - 2)$

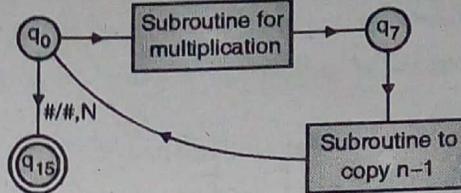


Fig. 7.4(a)

#### Subroutine for multiplication :

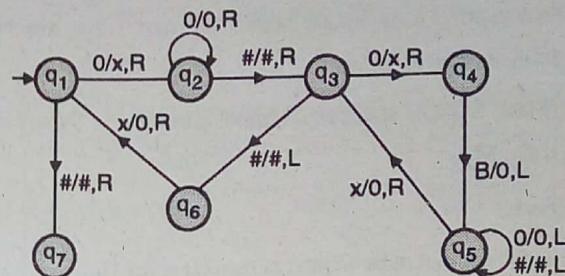


Fig. 7.4(b)

#### Subroutine to copy $n - 1$ :

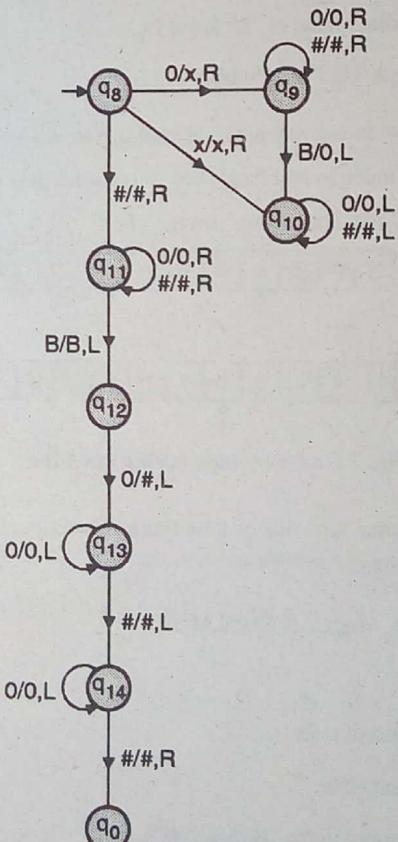


Fig. 7.4(c)

**Q. 6 Write note on 'Multiple Turing machine'.**

Dec. 2007

**Ans. :**

### Multiple Turing machine

#### 1. A Turing Machine with Multiple Heads

A turing machine with single tape can have multiple heads. Let us consider a turing machine with two heads  $H_1$  and  $H_2$ . Each head is capable of performing read/write /move operation independently.

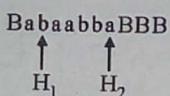


Fig. 7.5 : A Turing machine with two heads

The transition behavior of 2-head one tape Turing machine can be defined as given below :

$$\delta \text{ (State, Symbol under } H_1, \text{ Symbol under } H_2) = (\text{New state}, (S_1, M_1), (S_2, M_2))$$

Where,

$S_1$  is the symbol to be written in the cell under  $H_1$ .

$M_1$  is the movement (L, R, N) of  $H_1$ .

$S_2$  is the symbol to be written in the cell under  $H_2$ .

$M_2$  is the movement (L, R, N) of  $H_2$ .

#### 2. Multi-Tape Turing Machine

Multi-Tape turing machine has multiple tuples with each tape having its own independent head. Let us consider the case of a two tape turing machine. It is shown in Fig. 7.6.

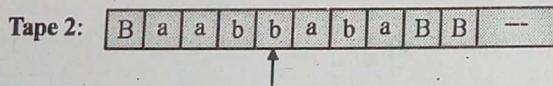
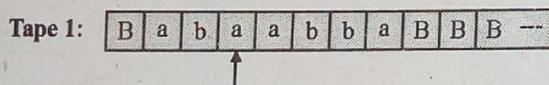


Fig. 7.6 : A two-tape turing machine

The transition behavior of a two-tape Turing machine can be defined as :

$$\delta(q_1, a_1, a_2) = (q_2, (S_1, M_1), (S_2, M_2))$$

Where,

$q_1$  is the current state,

$q_2$  is the next state,

$a_1$  is the symbol under the head on tape 1,

$a_2$  is the symbol under the head on tape 2,  
 $S_1$  is the symbol written in the current cell on tape 1,  
 $S_2$  is the symbol written in the current cell on tape 2,  
 $M_1$  is the movement (L, R, N) of head on tape 1,  
 $M_2$  is the movement (L, R, N) of head on tape 2.

**Q. 7 Design a TM which recognizes words of the form  $a^n b^n c^n \mid n \geq 1$ .**

May 2006, May 2008, Dec. 2011, Dec. 2016

**Ans. :**

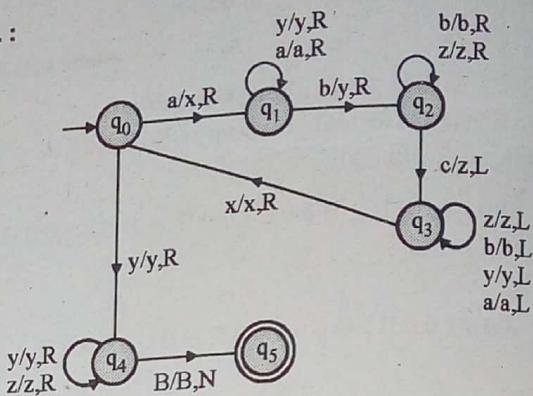


Fig. 7.7(a) : Transition diagram

	a	b	c	x	y	z	B
$\rightarrow q_0$	$(q_1, x, R)$	-	-	-	$(q_4, y, R)$	-	-
$q_1$	$(q_1, a, R)$	$(q_2, y, R)$	-	-	$(q_1, y, R)$	-	-
$q_2$	-	$(q_2, b, R)$	$(q_3, z, R)$	-	-	$(q_2, z, R)$	-
$q_3$	$(q_3, a, L)$	$(q_3, b, L)$	-	$(q_0, x, R)$	$(q_3, y, L)$	$(q_3, z, L)$	-
$q_4$	-	-	-	-	$(q_4, y, R)$	$(q_4, z, R)$	$(q_5, B, N)$
$q_5^*$	$q_5$						

Hating state

Fig. 7.7(b) : Transition table

The Turing machine  $M$  is given by :

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

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

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

$\Gamma = \{a, b, c, x, y, z, B\}$

$\delta =$  The transition is given Fig. 7.7(a, b)

$q_0 =$  Initial state

$B =$  Blank symbol

$F = \{q_5\}$ , Halting state

**Algorithm :**

For a string  $a^n b^n c^n$ , the TM will need  $n$  cycles. In each cycle :

1. Leftmost  $a$  is written as  $x$
2. Leftmost  $b$  is written as  $y$
3. Leftmost  $c$  is written as  $z$

At the end of  $n$  cycles, the tape should contain only  $x$ 's,  $y$ 's and  $z$ 's.

Working of the TM for input  $a^3 b^3 c^3$  is shown in Fig. 7.7(c) :

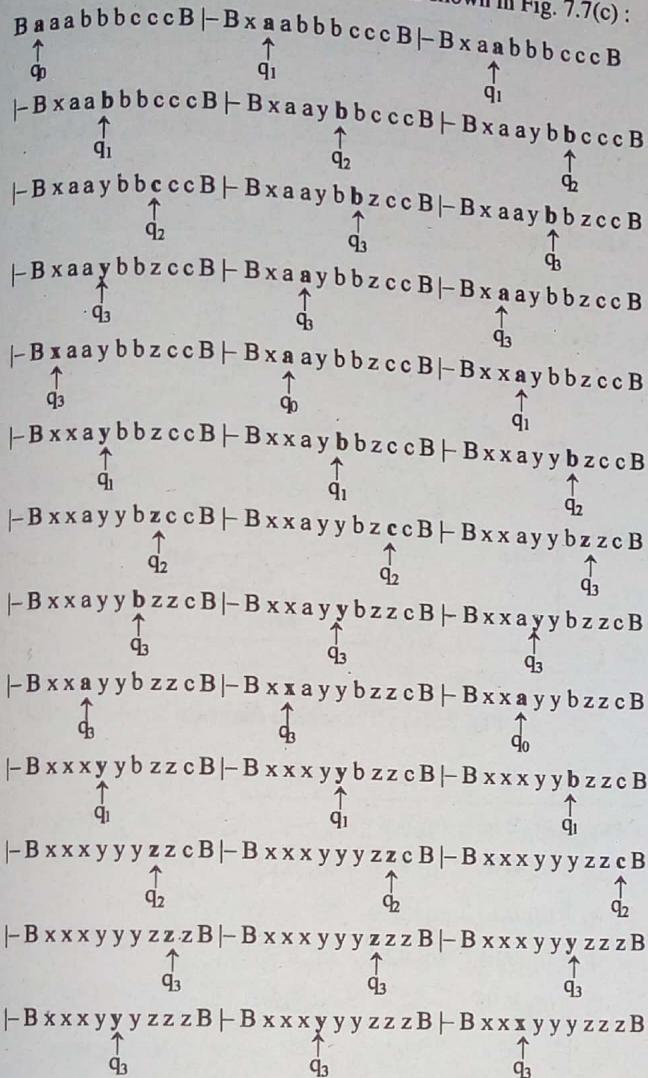


Fig. 7.7(c)

**Q. 8 Design a turing machine to check whether a string over {a,b} contains equal number of a's and b's.**

Dec. 2009, May 2008, Dec. 2015

**Ans. :**

**Algorithm :**

1. Locate first  $a$  or first  $b$ .
2. If it is ' $a$ ' then locate ' $b$ ' rewrite them as  $x$ .
3. If it is ' $b$ ' then locate ' $a$ ' rewrite them as  $x$ .
4. Repeat steps from 1 to 3 till every  $a$  or  $b$  is re-written as  $x$ .

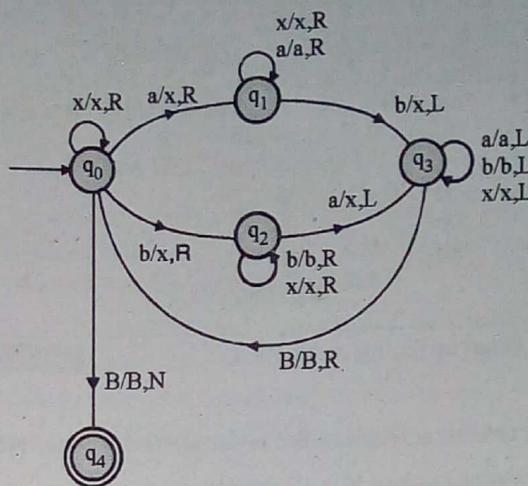


Fig. 7.8(a) : State transition diagram

	a	b	X	B	
$\rightarrow q_0$	$(q_1, X, R)$	$(q_2, X, R)$	$(q_0, X, R)$	$(q_4, B, N)$	
$q_1$	$(q_1, a, R)$	$(q_3, X, L)$	$(q_1, X, R)$		-
$q_2$	$(q_3, X, L)$	$(q_2, b, R)$	$(q_2, X, R)$		-
$q_3$	$(q_3, a, L)$	$(q_3, b, L)$	$(q_3, X, L)$	$(q_0, B, R)$	
$q_4^*$	$q_4$	$q_4$	$q_4$	$q_4$	← Halting state

Fig. 7.8(b) : Transition table

The turing machine M is given by :

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

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

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

$\Gamma = \{a, b, X, B\}$

$q_0$  = Initial state

$B$  = Blank symbol

$F = \{q_4\}$

Working of machine for an input abba is shown in Fig. 7.8(c)

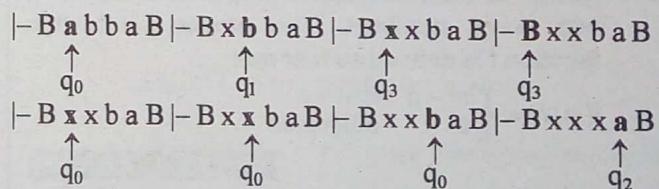


Fig. 7.8(c) Contd....

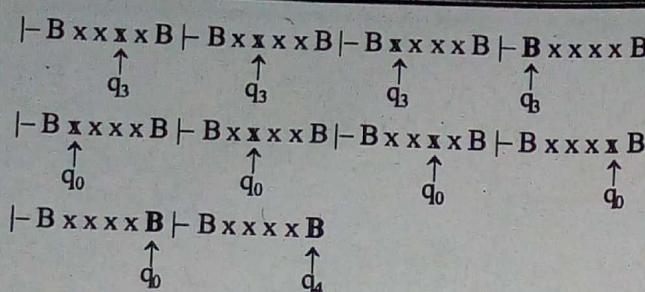


Fig. 7.8(c)

**Q. 9 What is Turing machine ?**

Dec. 2008

**Ans. :****Turing machine : Formal Definition of Turing Machine**

A Turing machine M is a 7-tuple given by :

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

where

1. Q is finite set of states
2.  $\Sigma$  is finite set of input alphabet not containing B.
3.  $\Gamma$  is a finite set of tape symbols. Tape symbols include B.
4.  $q_0 \in Q$  is the initial symbol.
5.  $B \in \Gamma$  is a special symbol representing an empty cell.
6.  $F \subseteq Q$  is the set of final states, final states are also known as halting states.
7. The transition function  $\delta$  is a function from  $Q \times \Gamma$  to  $Q \times \Gamma \times \{L, R, N\}$

A transition in turing machine is written as,

$\delta(q_0, a) = (q_1, b, R)$ , which implies, when in state  $q_0$  and scanning symbol a, the machine will enter state  $q_1$ , it will rewrite a as b and move to the right cell.

A transition  $\delta(q_0, a) = (q_1, a, R)$ , implies that the machine will enter state  $q_1$ , it will not change the symbol being scanned and move to the right cell.

Movement of Read / Write head is given L, R or N

L → Move to left cell

R → Move to right cell

N → Remain in the current cell (No movement)

**Q. 10 Design a TM to compute proper subtraction of two unary numbers. The proper subtraction function f is defined as follows :**

$$f(m, n) = \begin{cases} m - n & \text{if } m > n \\ 0 & \text{otherwise} \end{cases}$$

May 2009, Dec. 2009

**Ans. :**

The working of the TM is being explained with subtraction of 3 from 5.

In unary system, 5 is represented as 00000.

In unary system, 3 is represented as 000.

In unary system, 0 is represented by a blank tape.

Subtraction will require several cycle. In each cycle :

1. Leftmost 0 is erased
2. Rightmost 0 is erased.

Situation of tape after each cycle is shown below :

Initial 

B	0	0	0	0	0	#	0	0	0	B	---
---	---	---	---	---	---	---	---	---	---	---	-----

After 1<sup>st</sup> cycle 

B	B	0	0	0	0	#	0	0	B	B	---
---	---	---	---	---	---	---	---	---	---	---	-----

After 2<sup>nd</sup> cycle 

B	B	B	0	0	0	#	0	B	B	B	---
---	---	---	---	---	---	---	---	---	---	---	-----

After 3<sup>rd</sup> cycle 

B	B	B	B	0	0	#		B	B	---
---	---	---	---	---	---	---	--	---	---	-----

Transition diagram and transition table are given in Fig. 7.9(a) and (b).

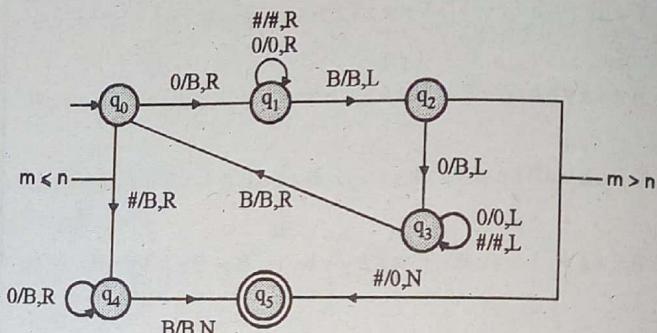


Fig. 7.9(a) : Transition diagram

	0	#	B
$\rightarrow q_0$	$(q_1, B, R)$	$(q_4, B, R)$	-
$q_1$	$(q_1, 0, R)$	$(q_1, \#, R)$	$(q_2, B, L)$
$q_2$	$(q_3, B, L)$	$(q_5, 0, N)$	-
$q_3$	$(q_3, 0, L)$	$(q_3, \#, L)$	$(q_0, B, R)$
$q_4$	$(q_4, B, R)$	-	$(q_5, B, N)$
$q_5^*$	$q_5$	$q_5$	$q_5$ ← Halting state

Fig. 7.9(b) : Transition table

The Turing machine M is given by :

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

where,

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

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

$$\Gamma = \{0, 1, \#, B\}$$

The transition function  $\delta$  is given in Fig. 7.9(a) and (b)

$$q_0 = \text{initial state,}$$

B = blank symbol

F =  $(q_s)$ , Halting state

The working of TM is being simulated for 5-3 is shown in Fig. Ex. 7.3(c) :

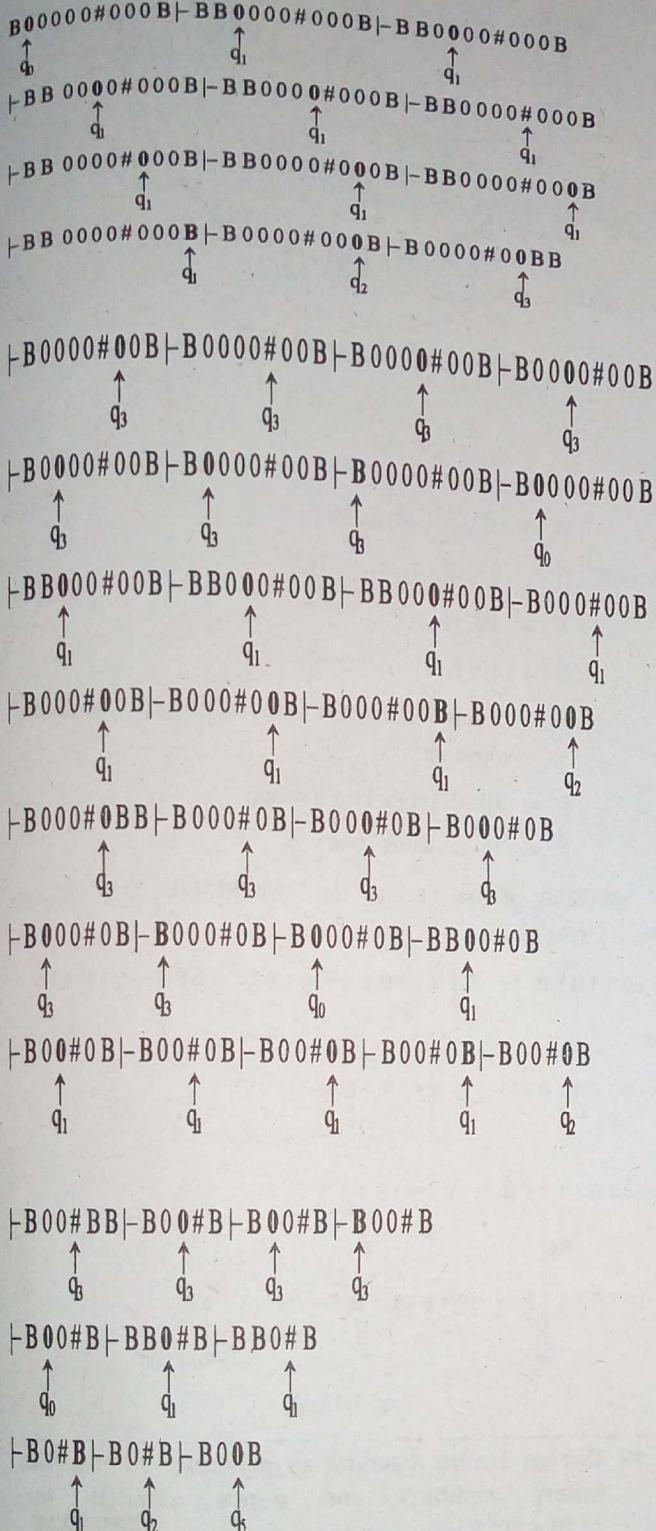


Fig. 7.9(c)

**Q. 11 Write short note on Variants of TM.**

Dec. 2006, Dec. 2008, Dec. 2009, Dec. 2010,  
May 2014, May 2015, May 2017

**Ans. :****1. Two-way Infinite Turing Machine**

In a standard turing machine number of positions for leftmost blanks is fixed and they are included in instantaneous description, where the right-hand blanks are not included.

In the two way infinite Turing machine, there is an infinite sequence of blanks on each side of the input string. In an instantaneous description, these blanks are never shown.

**2. A Turing Machine with Multiple Heads**

A turing machine with single tape can have multiple heads. Let us consider a turing machine with two heads  $H_1$  and  $H_2$ . Each head is capable of performing read/write /move operation independently.

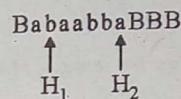


Fig. 7.10 : A Turing machine with two heads

The transition behavior of 2-head one tape Turing machine can be defined as given below :

$\delta$  (State, Symbol under  $H_1$ , Symbol under  $H_2$ ) = (New state,  $(S_1, M_1)$ ,  $(S_2, M_2)$ )

Where,

$S_1$  is the symbol to be written in the cell under  $H_1$ .

$M_1$  is the movement (L, R, N) of  $H_1$ .

$S_2$  is the symbol to be written in the cell under  $H_2$ .

$M_2$  is the movement (L, R, N) of  $H_2$ .

**3. Multi-Tape Turing Machine**

Multi-Tape turing machine has multiple tuples with each tape having its own independent head. Let us consider the case of a two tape turing machine. It is shown in Fig. 7.11.

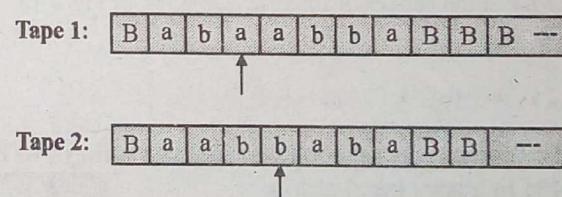


Fig. 7.11 : A two-tape turing machine

The transition behavior of a two-tape Turing machine can be defined as :

$$\delta(q_1, a_1, a_2) = (q_2, (S_1, M_1), (S_2, M_2))$$

Where,

## Theory of Comp. Sci. (MU-Sem. 5-Comp.)

$q_1$  is the current state,

$q_2$  is the next state,

$a_1$  is the symbol under the head on tape 1,

$a_2$  is the symbol under the head on tape 2,

$S_1$  is the symbol written in the current cell on tape 1,

$S_2$  is the symbol written in the current cell on tape 2,

$M_1$  is the movement (L, R, N) of head on tape 1,

$M_2$  is the movement (L, R, N) of head on tape 2.

## 4. Non-deterministic Turing Machine

Non-deterministic is a powerful feature. A non-deterministic TM machine might have, on certain combinations of state and symbol under the head, more than one possible choice of behaviour.

Non-deterministic does not make a TM more powerful.

For every non-deterministic TM, there is an equivalent deterministic TM.

It is easy to design a non-deterministic TM for certain class of problems.

A string is said to be accepted by a NDTM, if there is at least one sequence of moves that takes the machine to final state.

An example of non-deterministic move for a TM is shown in

Fig. 7.12.

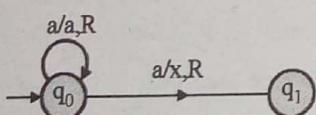


Fig. 7.12 : A sample move for NDTM

The transition behaviour for state  $q_0$  for TM of Fig. 7.12 can be written as

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

Q. 12 Design a turing machine to replace string 110 by 101 in binary input string. May 2010

Ans. :

The turing machine will look for every occurrence of the string 110.

State  $q_2$  is for previous two symbols as 11.

Next symbol as 0 in state  $q_2$ , will initiate the replacement process to replace 110 by 101.

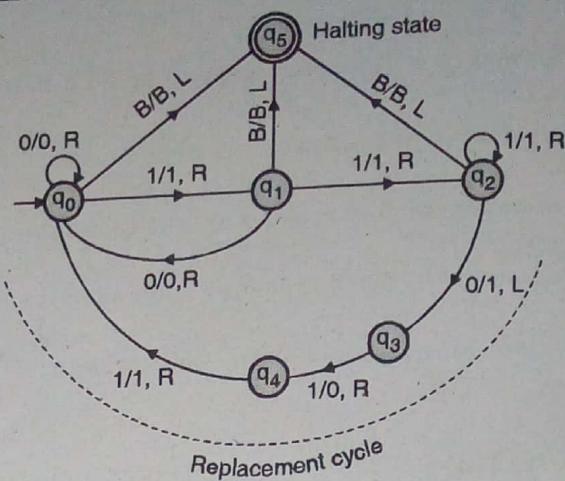


Fig. 7.13

The turing machine M is given by :

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

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

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

$$\Gamma = \{0, 1, B\}$$

$\delta$  = Transition function is shown using the transition diagram

B = Blank symbol for the tape

F =  $\{q_5\}$ , halting state

Working of the machine for input 0101101 is shown in Fig. 7.13(a) :

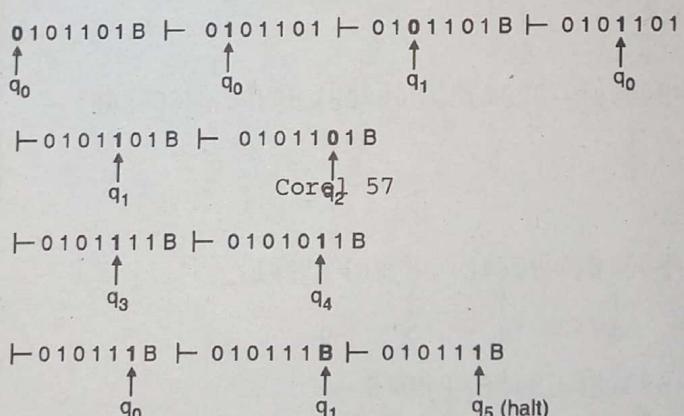


Fig. 7.13(a)

Q. 13 Design Turing machine as generator to add two binary numbers and hence simulate for "110 + 10". Dec. 2014

Ans. :

This problem can be solved using a 3-tape Turing machine.

First machine T1 stores the first binary number. Second machine T2 stores the second binary number. Third machine T3 stores the result.

The Turing machine will have 3 states :  
 q<sub>0</sub> - previous carry as 0  
 q<sub>1</sub> - previous carry as 1  
 q<sub>2</sub> - Halting state

(0, 0, L) (0, 0, L) (B, 0, L)	(1, 1, L) (0, 0, L) (B, 0, L)
(1, 1, L) (0, 0, L) (B, 1, L)	(1, 1, L) (B, B, L) (B, 0, L)
(0, 0, L) (1, 1, L) (B, 1, L)	(0, 0, L) (1, 1, L) (B, 0, L)
(B, B, L) (0, 0, L) (B, 0, L)	(B, B, L) (1, 1, L) (B, 0, L)
(0, 0, L) (B, B, L) (B, 0, L)	(1, 1, L) (1, 1, L) (B, 1, L)
(B, B, L) (1, 1, L) (B, 1, L)	
(1, 1, L) (B, B, L) (B, 1, L)	

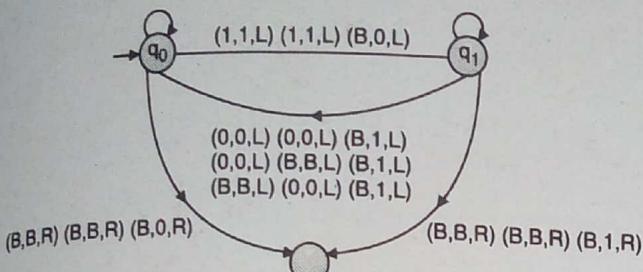


Fig. 7.14

Simulation for 110 + 10

B	B	1	0	
B	1	1	0	
B	B	B	B	

↑

State q<sub>0</sub>

B	B	1	0	
B	1	1	0	
B	B	B	0	

↑

State q<sub>0</sub>

B	B	1	0	
B	1	1	0	
B	B	0	0	

↑

State q<sub>1</sub>

B	B	1	0	
B	1	1	0	
B	0	0	0	

↑

State q<sub>1</sub>

B	B	B	1	0	
B	B	1	1	0	
B	1	0	0	0	

↑  
q<sub>2</sub> (Halt)

Q. 14 Design a Turing machine as acceptor for the language  $\{a^n b^m \mid n, m \geq 0 \text{ and } m \geq n\}$ . [Dec. 2014]

Ans. :

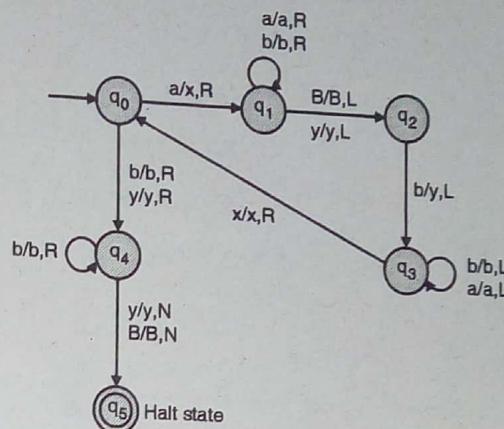


Fig. 7.15

Q. 15 Construct turning machine that accepts the string over  $\Sigma = \{0, 1\}$  and converts every occurrence of 111 to 101. [May 2015]

Ans. :

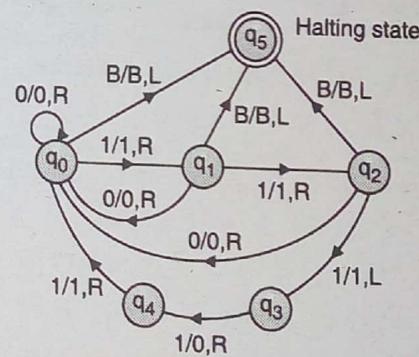


Fig. 7.16

The turing machine M is given by :

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

Where, Q = {q<sub>0</sub>, q<sub>1</sub>, q<sub>2</sub>, q<sub>3</sub>, q<sub>4</sub>, q<sub>5</sub>}

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

$$\Gamma = \{0, 1, B\}$$

$\delta$  = Transition function is shown using the transition diagram

B = Blank symbol for the tape

F = {q<sub>5</sub>}, halting state

**Q. 16** Construct a TM for checking well formedness of parentheses. [May 2012, May 2016, May 2017]

**Ans. :**  
In each cycle, the left-most ')' is written as 'x', then the head moves left to locate the nearer '(' and it is changed to 'x'.

The cycles of computation are shown below.

Input string is assumed to be "())()".

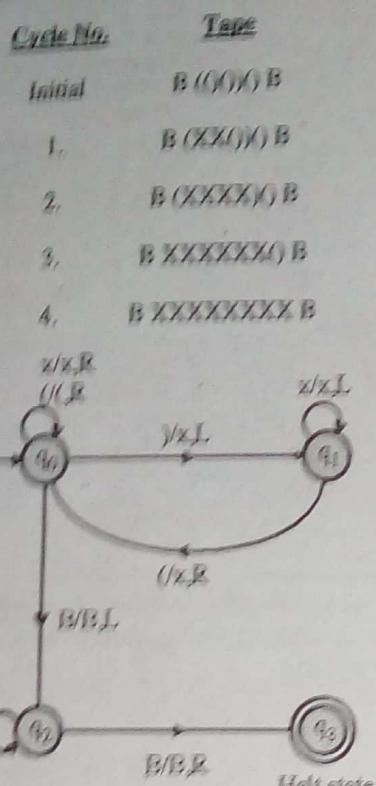


Fig. 7.17(a) : State transition diagram

	(	)	x	B
$q_0$	$(q_0, R)$	$(q_1, x, L)$	$(q_0, x, R)$	$(q_0, B, L)$
$q_1$	$(q_0, x, R)$	-	$(q_1, x, L)$	-
$q_2$	-	-	$(q_0, x, L)$	$(q_2, B, R)$
$q_3^*$	$q_0$	$q_0$	$q_0$	$q_0$
↓				

Halting state

Fig. 7.17(b) : State transition table

The Turing machine M is given by :

$$M = (Q, Z, \Gamma, \delta, q_0, B, F)$$

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

$$Z = \{(), x\}$$

$$\Gamma = \{(), x, B\}$$

$\delta$  is given in Fig. 7.17(a) or 7.17(b)

$q_0$  = Initial state

B = Blank symbol

F =  $\{q_3\}$ , halting state

Making of the machine for input (00)0 is given in Fig. 7.17(c) :

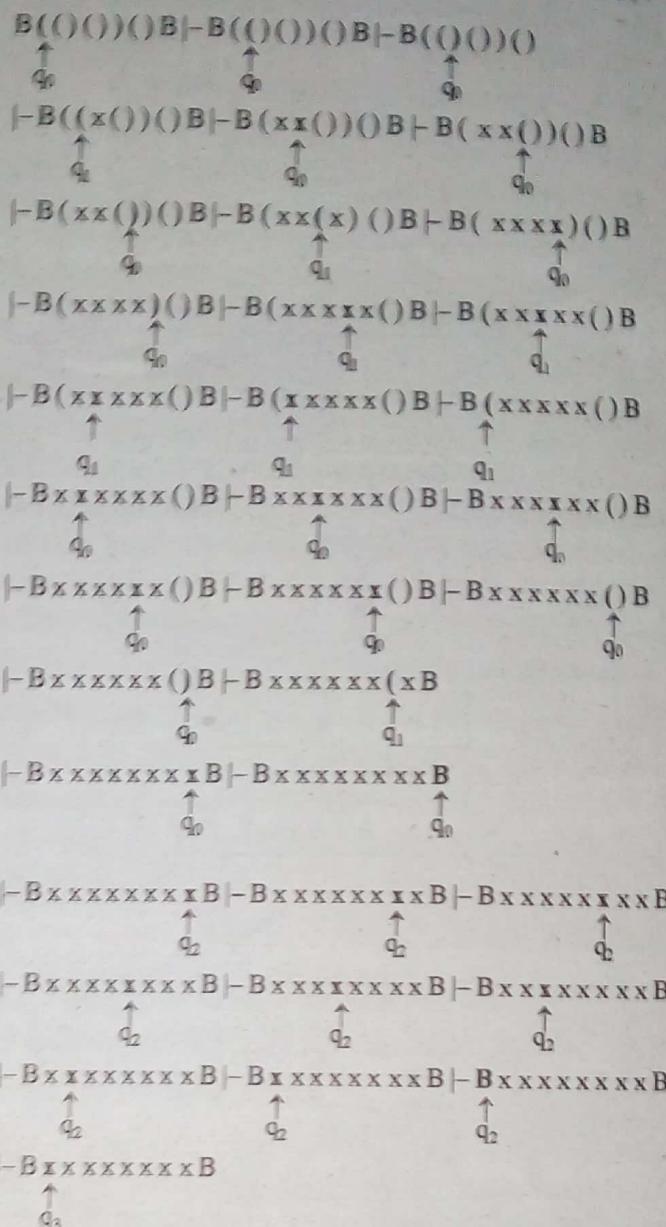


Fig. 7.17(c)

**Q. 17** Design a turing machine to check whether a string over {a,b} contains equal number of a's and b's. [Dec. 2009, May 2008, Dec. 2015]

**Ans. :**

**Algorithm :**

1. Locate first a or first b.
2. If it is 'a' then locate 'b' rewrite them as x.
3. If it is 'b' then locate 'a' rewrite them as x.
4. Repeat steps from 1 to 3 till every a or b is re-written as x.

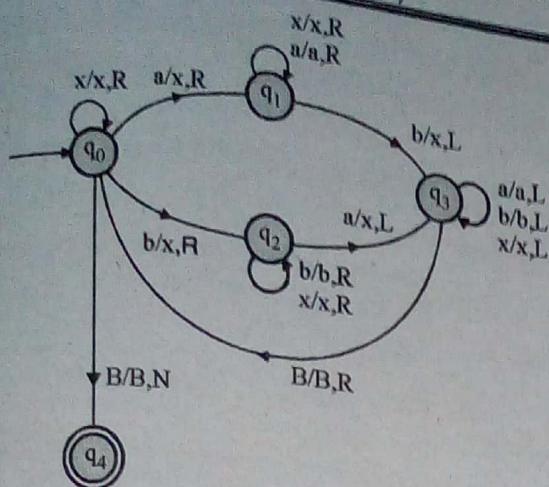


Fig. 7.18(a) : State transition diagram

	a	b	X	B	
$\rightarrow q_0$	$(q_1, X, R)$	$(q_2, X, R)$	$(q_0, X, R)$	$(q_4, B, N)$	
$q_1$	$(q_1, a, R)$	$(q_3, X, L)$	$(q_1, X, R)$	-	
$q_2$	$(q_3, X, L)$	$(q_2, b, R)$	$(q_2, X, R)$	-	
$q_3$	$(q_3, a, L)$	$(q_3, b, L)$	$(q_3, X, L)$	$(q_0, B, R)$	
$q_4^*$	$q_4$	$q_4$	$q_4$	$q_4$	$\leftarrow$ Halting state

Fig. 7.18(b) : Transition table

The turing machine M is given by :

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

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

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

$\Gamma = \{a, b, X, Y, B\}$

$q_0$  = Initial state

B = Blank symbol

F =  $\{q_4\}$

Working of machine for an input abba is shown in Fig. 7.18(c) :

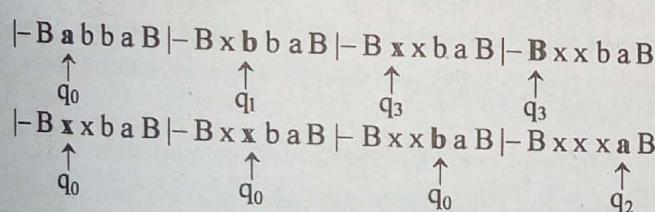


Fig. 7.18(c) Contd....

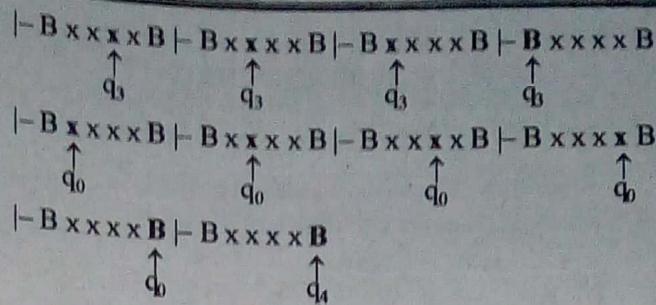


Fig. 7.18(c)

Q. 18 Design a Turing machine as an acceptor for the language

$$(a^n b^m \mid n, m \geq 0 \text{ and } m \geq n)$$

May 2016

Ans. :

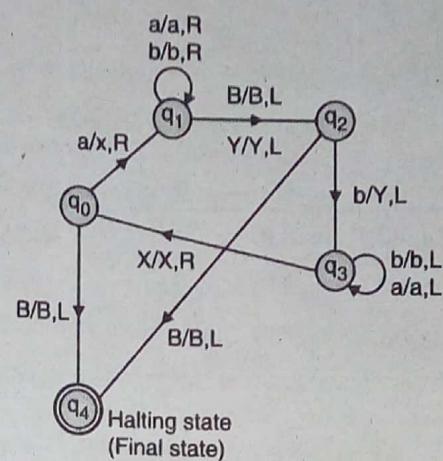


Fig. 7.19

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

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

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

$\Gamma = \{a, b, X, Y, B\}$

$q_0$  = initial state

B = Blank symbol

F =  $\{q_4\}$

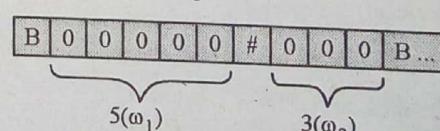
Q. 19 Design a TM to add two unary numbers.

May 2011, Dec. 2016

Ans. :

Addition of two unary numbers can be performed through append operation. To add two numbers 5 (say  $\omega_1$ ) and 3 (say  $\omega_2$ ) will require following steps :

- Initial configuration of tape :



2.  $\omega_1$  is appended to  $\omega_2$ .

B	0	0	0	0	0	0	0	0	Y	...
										$\omega_2$

While every '0' from  $\omega_1$  is getting appended to  $\omega_2$ , '0' from  $\omega_1$  is erased.  $\omega_2$  contains 8 '0's, which is sum of 5 and 3.

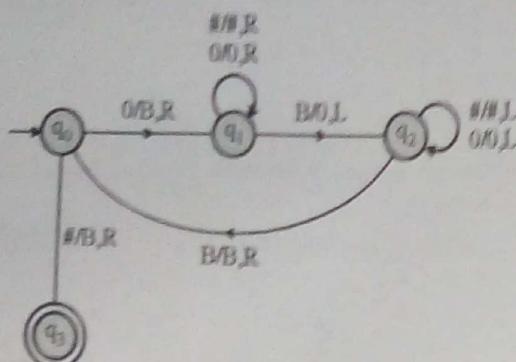


Fig. 7.20(a) : Transition diagram

	0	#	B
$\rightarrow q_0$	$(q_1, B, R)$	$(q_3, B, R)$	-
$q_1$	$(q_1, 0, R)$	$(q_1, \#, R)$	$(q_2, 0, L)$
$q_2$	$(q_2, 0, L)$	$(q_2, \#, L)$	$(q_3, B, R)$
$q_3^*$	$q_3$	$q_3$	$q_3$ $\leftarrow$ Halting state

Fig. 7.20(b) : Transition table

The turing machine M is given by :

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

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

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

$$\Gamma = \{0, \#, B\}$$

$\delta$  = Transition function is given in

Fig. Ex. 7.3.10 (a), (b)

$q_0$  = initial state

B = blank symbol

F =  $\{q_3\}$ , halting state.

Q. 20 Write short note on : Church-Turing Thesis.

May 2017

Ans. :

#### Church-Turing Thesis

The Turing machine is a general model of computation. Any algorithmic procedure can be solved by a computer can also be solved by a TM. Problems computed by a computer or a TM are also known as partial recursive functions. Some enhancements to TM made the Church-Turing thesis acceptable. These enhancements are :

1. Multi-tape
2. Multi-head
3. Infinite tapes
4. Non-determinism.

Since the introduction of TM, no one has suggested an algorithm than can be solved by a computer but cannot be solved by a TM.

## Chapter 8 : Undecidability

Q. 1 Write short note on : Recursive and Recursively Enumerable Languages.

Dec. 2005, Dec. 2009, Dec. 2010, May 2014, Dec. 2014,

May 2015, Dec. 2015, May 2016, Dec. 2016,

Dec. 2017

Ans. :

#### Recursive and Recursively Enumerable Languages

There is a difference between recursively enumerable (Turing Acceptable) and recursive (Turing Decidable) language.

Following statements are equivalent :

1. The language L is Turing acceptable.
2. The language L is recursively enumerable.

Following statements are equivalent

1. The language L is Turing decidable.
2. The language L is recursive.
3. There is an algorithm for recognizing L.

Every Turing decidable language is Turing acceptable.

Every Turing acceptable language need not be Turing decidable.

#### Turing Acceptable Language

A language  $L \subseteq \Sigma^*$  is said to be a Turing Acceptable language if there is a Turing machine M which halts on every  $\omega \in L$  with an answer 'YES'. However, if  $\omega \notin L$ , then M may not halt.

#### Turing Decidable Language

A language  $L \subseteq \Sigma^*$  is said to be turing being decidable if there is a turing machine M which always halts on every  $\omega \in \Sigma^*$ . If  $\omega \in L$  then M halts, with answer 'YES', and if  $\omega \notin L$  then M halts, with answer 'NO'.

A set of solutions for any problem defines a language.

A problem P is said to be decidable /solvable if the language  $L \subseteq \Sigma^*$  representing the problem (set of solutions) is turing decidable.

If  $P$  is solvable / decidable then there is an algorithm for recognizing  $L$ , representing the problem. It may be noted that an algorithm terminates on all inputs.

Following statements are equivalent :

1. The language  $L$  is Turing decidable.
2. The language  $L$  is recursive.
3. There is an algorithm for recognizing  $L$ .

Every turing decidable language is turing acceptable.

Every turing acceptable language need not be turing decidable.

A language  $L \subseteq \Sigma^*$  many not be turing acceptable and hence not turing decidable. Thus we cannot design a turing machine / algorithm which halts for every  $\omega \in L$ .

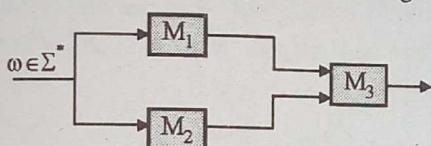
#### **Q. 2 Two recursive languages $L_1$ and $L_2$ is recursive : $L_1 \cup L_2$**

**Ans. :**

$L_1 \cup L_2$  is recursive

Let the turing machine  $M_1$  decides  $L_1$  and  $M_2$  decides  $L_2$ .

If a word  $\omega \in L_1$  then  $M_1$  returns "Y" else it returns "N". Similarly, if a word  $\omega \in L_2$  then  $M_2$  returns "Y" else it returns "N". Let us construct a turing machine  $M_3$  as shown in Fig. 8.1.



**Fig. 8.1 : A turing machine for  $L_1 \cup L_2$**

Output of machine  $M_1$  is written on the tape of  $M_3$ .

Output of machine  $M_2$  is written on the tape of  $M_3$ .

The machine  $M_3$  returns "Y" as output, if at least one of the outputs of  $M_1$ , or of  $M_2$  is "Y".

It should be clear that  $M_3$  decides  $L_1 \cup L_2$ . As both  $L_1$  and  $L_2$  are turing decidable, after a finite time both  $M_1$  and  $M_2$  will halt with answer "Y" or "N". The machine  $M_3$  is activated after  $M_1$  and  $M_2$  are halted. The machine  $M_3$  halts with answer "Y" if  $\omega \in L_1$  or  $\omega \in L_2$ , else  $M_3$  halts with output "N".

Thus  $L_1 \cup L_2$  is turing decidable or  $L_1 \cup L_2$  is recursive.

#### **Q. 3 Prove that there exists no algorithm for deciding whether a given CFG is ambiguous.**

**May 2006, Dec. 2007, Dec. 2008**

**Ans. :**

The post correspondence problem can be used to prove the un-decidability of whether a given CFG is ambiguous.

Let us consider two sequences of strings over  $\Sigma$ .

$$A = \{u_1, u_2, u_3 \dots u_m\}$$

$$B = \{v_1, v_2, v_3 \dots v_m\}$$

Let us take a new set of symbols  $a_1, a_2 \dots a_m$  such that

$$\{a_1, a_2 \dots a_m\} \cap \Sigma = \emptyset$$

Symbols  $a_1, a_2 \dots a_m$  are being taken as index symbols. The index symbol  $a_i$  represents a choice of  $u_i$  from  $A$  and  $v_i$  from the list  $B$ .

A string of the form  $u_i u_j u_k \dots u_m a_1 a_2 \dots a_l$  over alphabet  $\Sigma \cup \{a_1, a_2 \dots a_m\}$  can be defined using the set of productions :

$$G_A = \left\{ A \rightarrow u_1 A a_1 | u_2 A a_2 | \dots | u_m A a_m \right\}$$

Similarly a string of the form  $v_i v_j v_k \dots v_m a_1 a_2 \dots a_l$  over alphabet  $\Sigma \cup \{a_1, a_2 \dots a_m\}$  can be defined using the set of productions :

$$G_B = \left\{ B \rightarrow v_1 B a_1 | v_2 B a_2 | \dots | v_m B a_m \right\}$$

Finally, we can combine the languages and grammars of two lists to form a grammar  $G_{AB}$  :

A new start symbol  $S$  is added to  $G_{AB}$

Two new productions are added to  $G_{AB}$

$$S \rightarrow A$$

$$S \rightarrow B$$

All productions of  $G_A$  and  $G_B$  are taken.

Now, we will show that  $G_{AB}$  is ambiguous if and only if an instance (A, B) of PCP has a solution.

#### **Assumption :**

Suppose the sequence  $i_1, i_2, \dots, i_m$  is a solution to this instance of PCP. Two derivations for the above string in  $G_{AB}$  is :

$$S \Rightarrow A \Rightarrow u_{i_1} A a_{i_1} \Rightarrow u_{i_1} u_{i_2} A a_{i_1} a_{i_2} \Rightarrow \dots \Rightarrow$$

$$u_{i_1} u_{i_2} \dots u_{i_m} a_{i_1} a_{i_2} \dots a_{i_m}$$

$$S \Rightarrow B \Rightarrow v_{i_1} B a_{i_1} \Rightarrow v_{i_1} v_{i_2} B a_{i_1} a_{i_2} \Rightarrow \dots \Rightarrow$$

$$v_{i_1} v_{i_2} \dots v_{i_m} a_{i_1} a_{i_2} \dots a_{i_m}$$

Consequently, if  $G_{AB}$  is ambiguous, then the post correspondence problem with the pair (A, B) has a solution. Conversely, if  $G_{AB}$  is unambiguous, then the post correspondence cannot have a solution.

If there exists an algorithm for solving the ambiguous problem, then there exists an algorithm for solving the post correspondence problem. But, since there is no algorithm for the post correspondence problem, the ambiguity of CFG problem is unsolvable.

#### **Q. 4 Write short notes on post correspondence problem and Greibach Theorem.**

**May 2006, Dec. 2006, May 2007, Dec. 2007, May 2008,**

**Dec. 2008, May 2009, May 2010, Dec. 2010,**

**May 2011, Dec. 2011, May 2012, May 2016**

**Ans. :****Post correspondence problem**

**Definition :** Let A and B be two non-empty lists of strings over  $\Sigma$ . A and B are given as below :

$$A = \{x_1, x_2, x_3 \dots x_k\}$$

$$B = \{y_1, y_2, y_3 \dots y_m\}$$

There is a post correspondence between A and B if there is a sequence of one or more integers i, j, k ... m such that :

The string  $x_i x_j \dots x_m$  is equal to  $y_i y_j \dots y_m$ .

**Example :** Does the PCP with two lists :

$$A = \{a, aba^3, ab\} \text{ and}$$

$$B = \{a^3, ab, b\}$$

have a solution ?

So to find a sequence using which when the elements of A and B are listed, will produce identical strings.

The required sequence is (2, 1, 1, 3)

$$A_2 A_1 A_1 A_3 = aba^3 a ab = ab a^6 b$$

$$B_2 B_1 B_1 B_3 = aba^3 a^3 b = aba^6 b$$

Thus, the PCP has solution.

So accept the un-decidability of post correspondence problem without proof.

**Example :**

Determining the solution for following instance of PCP.

	List A	List B
i	$\omega_i$	$x_i$
1	01	0
2	110010	0
3	1	1111
4	11	01

The PCP has a solution. The required sequence is (1, 3, 2, 4, 4, 3)

$$\omega_1 \omega_3 \omega_2 \omega_4 \omega_4 \omega_3 = 01111001011111$$

$$x_1 x_3 x_2 x_4 x_4 x_3 = 01111001011111$$

**Greibach Theorem****The Theorem states that :**

"Let  $\sigma$  be a class of languages that is effectively closed under concatenation with regular sets and union, and for which  $L = \Sigma^*$  is un-decidable for any sufficiently large fixed  $\Sigma$ . Let P be any non-trivial property that is true for all regular sets and that is preserved under a, where a is single symbol in  $\Sigma$ . Then P is un-decidable for  $\sigma$ ".

Greibach theorem can be used to prove that many problems related to CFG are un-decidable.

**Q. 5 Write short notes on : Halting problem.**

Dec. 2006, Dec. 2007, May 2008, Dec. 2008, May 2011,

Dec. 2011, Dec. 2015, Dec. 2016, May 2017

**Ans. :****Halting Problem of a Turing Machine**

The halting problem of a Turing machine states :

Given a Turing machine M and an input  $\omega$  to the machine M, determine if the machine M will eventually halt when it is given input  $\omega$ .

Halting problem of a Turing machine is unsolvable.

**Proof :**

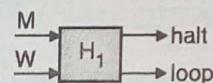
Moves of a turing machine can be represented using a binary number. Thus, a Turing machine can be represented using a string over  $\Sigma^*(0,1)$ . This concept has already been explained in the chapter.

Insolvability of halting problem of a Turing machine can be proved through the method of contradiction.

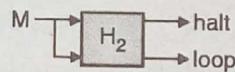
**Step 1 :** Let us assume that the halting problem of a Turing machine is solvable. There exists

1. A string describing M.
2. An input  $\omega$  for machine M.

$H_1$  generates an output "halt" if  $H_1$  determines that M stops on input  $\omega$ ; otherwise  $H_1$  outputs "loop". Working of the machine  $H_1$  is shown below.

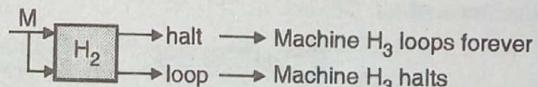


**Step 2 :** Let us revise the machine  $H_1$  as  $H_2$  to take M as both inputs and  $H_2$  should be able to determine if M will halt on M as its input. Please note that a machine can be described as a string over 0 and 1.



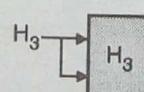
**Step 3 :** Let us construct a new Turing machine  $H_3$  that takes output of  $H_2$  as input and does the following :

1. If the output of  $H_2$  is "loop" than  $H_3$  halts.
2. If the output of  $H_2$  is "halt" than  $H_3$  will loop forever.



$H_3$  will do the opposite of the output of  $H_2$ .

**Step 4 :** Let us give  $H_3$  itself as inputs to  $H_3$ .



If  $H_3$  halts on  $H_3$  as input then  $H_3$  would loop (that is how we constructed it). If  $H_3$  loops forever on  $H_3$  as input  $H_3$  halts (that is how we constructed it).

In either case, the result is wrong.

Hence,

$H_3$  does not exist.

If  $H_3$  does not exist then  $H_2$  does not exist.

If  $H_2$  does not exist then  $H_1$  does not exist.

**Q. 6 Does PCP with following two list : A = (10, 011, 101) and B = (101, 11, 011) have a solution ? Justify your answer.**

May 2009

**Ans. :**

$A_2$  and  $A_3$  differ from  $B_2$  and  $B_3$  at the first of place. Therefore, we must pick  $A_1$  and  $B_1$ .

Sequence	String
(1)	$(A_1 = 10) (B_1 = 101)$

The next string to be picked up must be  $A_3$  and  $B_3$ . Any other sequence will not lead to a solution.

Sequence	String
(1, 3)	$(A_1 A_3 = 10101) (B_1 B_3 = 101011)$

The next string to be picked up must be  $A_3$  and  $B_3$ . Any other sequence will not lead to a solution.

Sequence	String
(1, 3, 3)	$(A_1 A_3 A_3 = 10101101) (B_1 B_3 B_3 = 101011011)$

There is only choice of next string. This choice is  $A_3$  and  $B_3$ . This does not lead to a solution. The PCP has no solution.

**Q. 7 Write short note on : Rice Theorem**

Dec. 2012, May 2013, May 2014, May 2015, Dec. 2015,  
May 2016, Dec. 2016, May 2017, Dec. 2017

**Ans. :**

**Rice Theorem**

"Every property that is satisfied by some but not all recursively enumerable language is un-decidable". Any property that is satisfied by some recursively enumerable language but not all is known as nontrivial property. We have seen many properties of R.E. languages that are un-decidable. These properties include :

- Given a TM M, is  $L(M)$  nonempty ?
- Given a TM M, is  $L(M)$  finite ?
- Given a TM M, is  $L(M)$  regular ?
- Given a TM M, is  $L(M)$  recursive ?

The Rice's theorem can be proved by reducing some other unsolvable problem to nontrivial property of recursively enumerable language.

