mood-book

7

TURING MACHINES

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

- Turing Machine
- Design of TM
- Computable functions
- Recursively Enumerable languages
- Church's Hypothesis & Counter machine
- Types of Turing Machines

7.1 INTRODUCTION

The Turing machine is a generalized machine which can recognize all types of languages viz, regular languages (generated from regular grammar), context free languages (generated from context free grammar) and context sensitive languages (generated from context sensitive grammar). Apart from these languages, the Turing machine also accepts the language generated from unrestricted grammar. Thus, Turing machine can accept any generalized language. This chapter mainly concentrates on building the Turing machines for any language.

7.2 TURING MACHINE MODEL

The Turing machine model is shown in below figure . It is a finite automaton connected to read - write head with the following components :

- Tape
- Read - write head
- Control unit

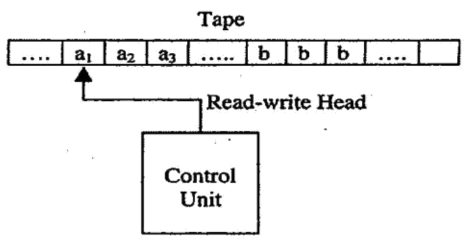


FIGURE : Turing machine model

Tape : It is a temporary storage and is divided into cells. Each cell can store the information of only one symbol. The string to be scanned will be stored from the left most position on the tape. The string to be scanned should end with infinite number of blanks.

Read - write head : The read - write head can read a symbol from where it is pointing to and it can write into the tape to where the read - write head points to.

Control Unit : The reading / writing from / to the tape is determined by the control unit. The different moves performed by the machine depends on the current scanned symbol and the current state. The read - write head can move either towards left or right i.e., movement can be on both the directions. The various moves performed by the machine are :

1. Change of state from one state to another state
2. The symbol pointing to by the read - write head can be replaced by another symbol.
3. The read - write head may move either towards left or towards right.

The Turing machine can be represented using various notations such as

- Transition table
- Instantaneous description
- Transition diagram

7.2.1 Transition Table

The table below shows the transition table for some Turing machine. Later sections describe how to obtain the transition table.

δ	Tape Symbols (Γ)				
States	a	b	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	(q_1, a, R)	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	(q_2, a, L)	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

Note that for each state q, there can be a corresponding entry for the symbol in Γ. In this table the symbols a and b are input symbols and can be denoted by the symbol Σ. Thus $\Sigma \subseteq \Gamma$ excluding the symbol B. The symbol B indicates a blank character and usually the string ends with infinite number of B's i. e., blank characters. The undefined entries indicate that there are no - transitions defined or there can be a transition to dead state. When there is a transition to the dead state, the machine halts and the input string is rejected by the machine. It is clear from the table that

$$\delta : Q \times \Gamma \; to \; (Q \times \Gamma \times \{L, R\})$$

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

$\Gamma = \{a, b, X, Y, B\}$

q_0 is the initial state; B is a special symbol indicating blank character

$F = \{q_4\}$ which is the final state.

Thus, a Turing Machine M can be defined as follows.

Definition : The Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where

Q is set of finite states

Σ is set of input alphabets

Γ is set of tape symbols

δ is transition function $Q \times \Gamma \; to \; (Q \times \Gamma \times \{L, R\})$

q_0 is the initial state

B is a special symbol indicating blank character

$F \subseteq Q$ is set of final states.

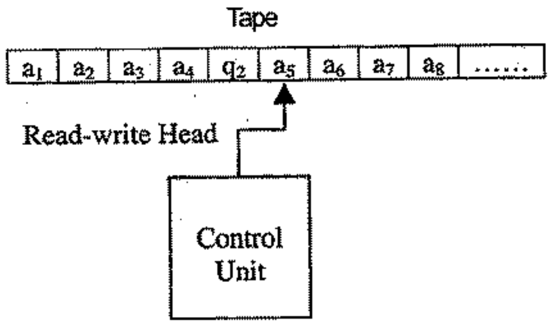
7.2.2 Instantaneous description (ID)

Unlike the ID described in PDA, in Turing machine (TM), the ID is defined on the whole string (not on the string to be scanned) and the current state of the machine.

Definition :

An ID of TM is a string in $\alpha q \beta$, where q is the current state, $\alpha \beta$ is the string made from tape symbols denoted by Γ i. e., α and $\beta \in \Gamma^*$. The read - write head points to the first character of the substring β. The initial ID is denoted by $q\alpha\beta$ where q is the start state and the read - write head points to the first symbol of α from left. The final ID is denoted by $\alpha\beta qB$ where $q \in F$ is the final state and the read - write head points to the blank character denoted by B.

Example : Consider the snapshot of a Turing machine



In this machine, each $a_i \in \Gamma$ (i. e., each a_i belongs to the tape symbol). In this snapshot, the symbol a_5 is under read - write head and the symbol towards left of a_5 i. e., q_2 is the current state. Note that, in the Turing machine, the symbol immediately towards left of the read - write head will be the current state of the machine and the symbol immediately towards right of the state will be the next symbol to be scanned. So, in this case an ID is denoted by

$$a_1a_2a_3a_4q_2a_5a_6a_7a_8 \ldots\ldots$$

where the substring $a_1a_2a_3a_4$ towards left of the state q_2 is the left sequence, the substring $a_5a_6a_7a_8\ldots\ldots$ towards right of the state q_2 is the right sequence and q_2 is the current state of the machine. The symbol a_5 is the next symbol to be scanned.

Assume that the current ID of the Turing machine is $a_1a_2a_3a_4q_2a_5a_6a_7a_8\ldots\ldots$ as shown in snapshot of example.

Suppose, there is a transition $\delta(q_2, a_5) = (q_3, b_1, R)$

It means that if the machine is in state q_2 and the next symbol to be scanned is a_5, then the machine enters into state q_3 replacing the symbol a_5 by b_1 and R indicates that the read - write head is moved one symbol towards right. The new configuration obtained is

$$a_1a_2a_3a_4b_1q_3a_6a_7a_8\ldots\ldots$$

This can be represented by a move as $a_1a_2a_3a_4q_2a_5a_6a_7a_8\ldots \mid\!- a_1a_2a_3a_4b_1q_3a_6a_7a_8\ldots\ldots$

Similarly if the current ID of the Turing machine is $a_1a_2a_3a_4q_2a_5a_6a_7a_8\ldots\ldots$

and there is a transition

$$\delta(q_2, a_5) = (q_1, c_1, L)$$

means that if the machine is in state q_2 and the next symbol to be scanned is a_5, then the machine enters into state q_1 replacing the symbol a_5 by c_1 and L indicates that the read - write head is moved one symbol towards left. The new configuration obtained is

$$a_1a_2a_3q_1a_4c_1a_6a_7a_8\ldots\ldots$$

This can be represented by a move as $a_1a_2a_3a_4q_2a_5a_6a_7a_8....$ $\vdash$ $a_1a_2a_3q_1a_4c_1a_6a_7a_8....$

This configuration indicates that the new state is q_1, the next input symbol to be scanned is a_4. The actions performed by TM depends on

1. The current state.
2. The whole string to be scanned
3. The current position of the read - write head

The action performed by the machine consists of

1. Changing the states from one state to another
2. Replacing the symbol pointed to by the read - write head
3. Movement of the read - write head towards left or right.

7.2.3 The move of Turing Machine M can be defined as follows

Definition : Let $M = (Q,\Sigma,\Gamma,\delta,q_0,B,F)$ be a TM. Let the ID of M be $a_1a_2a_3......a_{k-1}qa_ka_{k+1}......a_n$ where $a_j \in \Gamma$ for $1 \le j \le n-1$, $q \in Q$ is the current state and a_k as the next symbol to scanned. If there is a transition $\delta(q, a_k) = (p, b, R)$
then the move of machine M will be $a_1a_2a_3......a_{k-1}qa_ka_{k+1}.....a_n \vdash a_1a_2a_3......a_{k-1}bpa_{k+1}.....a_n$

If there is a transition $\delta(q, a_k) = (p, b, L)$
then the move of machine M will be

$$a_1a_2a_3......a_{k-1}qa_ka_{k+1}....a_n \vdash a_1a_2a_3......a_{k-2}pa_{k-1}ba_{k+1}......a_n$$

7.2.4 Acceptance of a language by TM

The language accepted by TM is defined as follows.

Definition :

Let $M = (Q,\Sigma,\Gamma,\delta,q_0,B,F)$ be a TM. The language L(M) accepted by M is defined as

$$L(M) = \{w \mid q_0w \vdash^* \alpha_1\ p\ \alpha_2 \text{ where } w \in \Sigma^*, p \in F \text{ and } \alpha_1, \alpha_2 \in \Gamma^*\}$$

i.e., set of all those words w in Σ^* which causes M to move from start state q_0 to the final state p. The language accepted by TM is called recursively enumerable language.

The string w which is the string to be scanned, should end with infinite number of blanks. Initially, the machine will be in the start state q_0 with read - write head pointing to the first symbol of w from left. After some sequence of moves, if the Turing machine enters into the final state and halts, then we say that the string w is accepted by Turing machine.

7.2.5 Differences between TM and PDA

Push Down Automa :

1. A PDA is a nondeterministic finite automaton coupled with a stack that can be used to store a string of arbitrary length.
2. The stack can be read and modified only at its top.
3. A PDA chooses its next move based on its current state, the next input symbol and the symbol at the top of the stack.
4. There are two ways in which the PDA may be allowed to signal acceptance. One is by entering an accepting state, the other by emptying its stack.
5. ID consisting of the state, remaining input and stack contents to describe the "current condition" of a PDA.
6. The languages accepted by PDA's either by final state or by empty stack, are exactly the context - free languages.
7. A PDA languages lie strictly between regular languages and CSL's.

Turing Machines :

1. The TM is an abstract computing machine with the power of both real computers and of other mathematical definitions of what can be computed.
2. TM consists of a finite - state control and an infinite tape divided into cells.
3. TM makes moves based on its current state and the tape symbol at the cell scanned by the tape head.
4. The blank is one of tape symbols but not input symbol.
5. TM accepts its input if it ever enters an accepting state.
6. The languages accepted by TM's are called Recursively Enumerable (RE) languages.
7. Instantaneous description of TM describes current configuration of a TM by finite - length string.
8. Storage in the finite control helps to design a TM for a particular language.
9. A TM can simulate the storage and control of a real computer by using one tape to store all the locations and their contents.

7.3 CONSTRUCTION OF TURING MACHINE (TM)

In this section, we shall see how TMs can be constructed.

Example 1 : Obtain a Turing machine to accept the language $L = \{ 0^n 1^n \mid n \geq 1 \}$.

Solution : Note that n number of 0's should be followed by n number of 1's. For this let us take an example of the string $w = 00001111$. The string w should be accepted as it has four zeroes followed by equal number of 1's.

General Procedure :

Let q_0 be the start state and let the read - write head points to the first symbol of the string to be scanned. The general procedure to design TM for this case is shown below :

1. Replace the left most 0 by X and change the state to q_1 and then move the read - write head towards right. This is because, after a zero is replaced, we have to replace the corresponding 1 so that number of zeroes matches with number of 1's.
2. Search for the leftmost 1 and replace it by the symbol Y and move towards left (so as to obtain the leftmost 0 again). Steps 1 and 2 can be repeated.

Consider the situation

$$\begin{array}{c} XX00YY11 \\ \uparrow \\ q_0 \end{array}$$

where first two 0's are replaced by Xs and first two 1's are replaced by Ys. In this situation, the read - write head points to the left most zero and the machine is in state q_0. With this as the configuration , now let us design the TM.

Step 1 : In state q_0, replace 0 by X, change the state to q_1 and move the pointer towards right. The transition for this can be of the form

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

The resulting configuration is shown below .

$$\begin{array}{c} XXX0YY11 \\ \uparrow \\ q_1 \end{array}$$

Step 2 : In state q_1, we have to obtain the left - most 1 and replace it by Y. For this, let us move the pointer to point to leftmost one. When the pointer is moved towards 1, the symbols encountered may be 0 and Y. Irrespective what symbol is encountered, replace 0 by 0, Y by Y, remain in state q_1 and move the pointer towards right. The transitions for this can be of the form

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

When these transitions are repeatedly applied, the following configuration is obtained.

$$\begin{array}{c} XXX0YY11 \\ \uparrow \\ q_1 \end{array}$$

Step 3 : In state q_1, if the input symbol to be scanned is a 1, then replace 1 by Y, change the state to q_2 and move the pointer towards left. The transition for this can be of the form

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

and the following configuration is obtained.

XXX0YYY1
↑
q_2

Note that the pointer is moved towards left. This is because, a zero is replaced by X and the corresponding 1 is replaced by Y. Now, we have to scan for the left most 0 again and so, the pointer was move towards left.

Step 4 : Note that to obtain leftmost zero, we need to obtain right most X first. So, we scan for the right most X. During this process we may encounter Y's and 0's . Replace Y by Y, 0 by 0, remain in state q_2 only and move the pointer towards left. The transitions for this can be of the form

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

The following configuration is obtained

XXX0YYY1
↑
q_2

Step 5 : Now, we have obtained the right most X. To get leftmost 0, replace X by X, change the state to q_0 and move the pointer towards right. The transition for this can be of the form

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

and the following configuration is obtained

XXX0YYY1
↑
q_0

Now, repeating the steps 1 through 5, we get the configuration shown below :

XXXXYYYY
↑
q_0

Step 6 : In state q_0, if the scanned symbol is Y, it means that there are no more 0's. If there are no zeroes we should see that there are no 1's. For this we change the state to q_3, replace Y by Y and move the pointer towards right. The transition for this can be of the form

$$\delta(q_0,Y)=(q_3,Y,R)$$

and the following configuration is obtained

XXXXYYYY

↑

q_3

In state q_3, we should see that there are only Ys and no more 1's. So, as we can replace Y by Y and remain in q_3 only. The transition for this can be of the form

$$\delta(q_3,Y)=(q_3,Y,R)$$

Repeatedly applying this transition, the following configuration is obtained.

XXXXYYYYB

↑

q_3

Note that the string ends with infinite number of blanks and so, in state q_3 if we encounter the symbol B, means that end of string is encountered and there exists n number of 0's ending with n number of 1's. So, in state q_3, on input symbol B, change the state to q_4, replace B by B and move the pointer towards right and the string is accepted. The transition for this can be of the form

$$\delta(q_3,B)=(q_4,B,R)$$

The following configuration is obtained

XXXXYYYYBB

↑

q_4

So, the Turing machine to accept the language $L=\{a^n\ b^n \mid n\geq 1\}$

is given by $M=(Q,\Sigma,\Gamma,\delta,q_0,B,F)$

where

$Q=\{q_0,q_1,q_2,q_3\}$; $\Sigma=\{0,1\}$; $\Gamma=\{0,1,X,Y,B\}$

$q_0 \in Q$ is the start state of machine; $B\in\Gamma$ is the blank symbol.

$F=\{q_4\}$ is the final state.

δ is shown below.

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

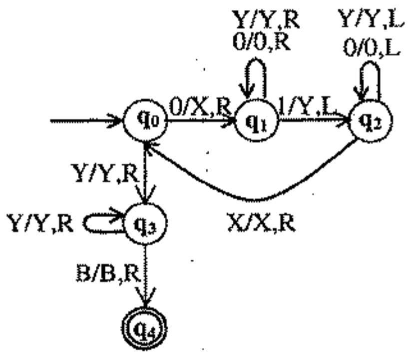
$$\delta(q_1,0)=(q_1,0,R)$$

$$\delta(q_1, Y) = (q_1, Y, R)$$
$$\delta(q_1, 1) = (q_2, Y, L)$$
$$\delta(q_2, Y) = (q_2, Y, L)$$
$$\delta(q_2, 0) = (q_2, 0, L)$$
$$\delta(q_2, X) = (q_0, X, R)$$
$$\delta(q_0, Y) = (q_3, Y, R)$$
$$\delta(q_3, Y) = (q_3, Y, R)$$
$$\delta(q_3, B) = (q_4, B, R)$$

The transitions can also be represented using tabular form as shown below.

δ	Tape Symbols (Γ)				
States	0	1	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	$(q_1, 0, R)$	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	$(q_2, 0, L)$	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

The transition table shown above can be represented as transition diagram as shown below:



To accept the string :

The sequence of moves or computations (IDs) for the string 0011 made by the Turing machine are shown below:

Initial ID

$q_0 0011$	$\vdash Xq_1 011$	$\vdash X0q_1 11$
	$\vdash Xq_2 0Y1$	$\vdash q_2 X0Y1$
	$\vdash Xq_0 0Y1$	$\vdash XXq_1 Y1$
	$\vdash XXYq_1 1$	$\vdash XXq_2 YY$
	$\vdash Xq_2 XYY$	$\vdash XXq_0 YY$
	$\vdash XXYq_3 Y$	$\vdash XXYYq_3$
	$\vdash XXYYBq_4$ (Final ID)	

Example 2 : Obtain a Turing machine to accept the language $L(M) = \{ 0^n 1^n 2^n \mid n \geq 1 \}$

Solution : Note that n number of 0's are followed by n number of 1's which in turn are followed by n number of 2's. In simple terms, the solution to this problem can be stated as follows :

Replace first n number of 0's by X's, next n number of 1's by Y's and next n number of 2's by Z's. Consider the situation where in first two 0's are replaced by X's , next immediate two 1's are replaced by Y's and next two 2's are replaced by Z's as shown in figure 1(a).

(a)	(b)	(c)
XX00YY11ZZ22	XXX0YY11ZZ22	XXX0YY11ZZ22
↑	↑	↑
q_0	q_1	q_1

FIGURE 1 : Various Configurations

Now, with figure 1(a). a as the current configuration, let us design the Turing machine. In state q_0, if the next scanned symbol is 0 replace it by X, change the state to q_1 and move the pointer towards right and the situation shown in figure 1(b) is obtained. The transition for this can be of the form

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

In state q_1, we have to search for the leftmost 1. It is clear from figure 1(b) that, when we are searching for the symbol 1, we may encounter the symbols 0 or Y. So, replace 0 by 0 , Y by Y and move the pointer towards right and remain in state q_1 only. The transitions for this can be of the form

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

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

The configuration shown in figure 1(c) is obtained. In state q_1, on encountering 1 change the state to q_2, replace 1 by Y and move the pointer towards right. The transition for this can be of the form

$$\delta(q_1,1)=(q_2,Y,R)$$

and the configuration shown in figure 2(a) is obtained

(a)	(b)	(c)
XXX0YYY1ZZ22	XXX0YYY1ZZ22	XXX0YYY1ZZZ2
↑ q_2	↑ q_2	↑ q_3

FIGURE 2 : Various Configurations

In state q_2, we have to search for the leftmost 2. It is clear from figure 2(a) that, when we are searching for the symbol 2, we may encounter the symbols 1 or Z. So, replace 1 by 1, Z by Z and move the pointer towards right and remain in state q_2 only and the configuration shown in figure 2(b) is obtained. The transitions for this can be of the form

$$\delta(q_2,1)=(q_2,1,R)$$
$$\delta(q_2,Z)=(q_2,Z,R)$$

In state q_2, on encountering 2, change the state to q_3, replace 2 by Z and move the pointer towards left. The transition for this can be of the form

$$\delta(q_2,2)=(q_3,Z,L)$$

and the configuration shown in figure 2(c) is obtained. Once the TM is in state q_3, it means that equal number of 0's, 1's and 2's are replaced by equal number of X's, Y's and Z's respectively. At this point, next we have to search for the rightmost X to get leftmost 0. During this process, it is clear from figure 2(c) that the symbols such as Z's, 1,s, Y's, 0's and X are scanned respectively one after the other. So, replace Z by Z, 1 by 1, Y by Y, 0 by 0, move the pointer towards left and stay in state q_3 only. The transitions for this can be of the form

$$\delta(q_3,Z)=(q_3,Z,L)$$
$$\delta(q_3,1)=(q_3,1,L)$$
$$\delta(q_3,Y)=(q_3,Y,L)$$
$$\delta(q_3,0)=(q_3,0,L)$$

Only on encountering X, replace X by X, change the state to q_0 and move the pointer towards right to get leftmost 0. The transition for this can be of the form

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

All the steps shown above are repeated till the following configuration is obtained.

XXXXYYYYZZZZ

↑

q_0

In state q_0, if the input symbol is Y, it means that there are no 0's . If there are no 0's we should see that there are no 1's also. For this to happen change the state to q_4, replace Y by Y and move the pointer towards right. The transition for this can be of the form

$$\delta(q_0, Y) = (q_4, Y, R)$$

In state q_4 search for only Y's, replace Y by Y, remain in state q_4 only and move the pointer towards right. The transition for this can be of the form

$$\delta(q_4, Y) = (q_4, Y, R)$$

In state q_4, if we encounter Z, it means that there are no 1's and so we should see that there are no 2's and only Z's should be present. So, on scanning the first Z, change the state to q_5, replace Z by Z and move the pointer towards right. The transition for this can be of the form

$$\delta(q_4, Z) = (q_5, Z, R)$$

But, in state q_5 only Z's should be there and no more 2's. So, as long as the scanned symbol is Z, remain in state q_5, replace Z by Z and move the pointer towards right. But, once blank symbol B is encountered change the state to q_6, replace B by B and move the pointer towards right and say that the input string is accepted by the machine. The transitions for this can be of the form

$$\delta(q_5, Z) = (q_5, Z, R)$$
$$\delta(q_5, B) = (q_6, B, R)$$

where q_6 is the final state.

So, the TM to recognize the language $L = \{ 0^n1^n2^n \mid n \geq 1 \}$ is given by

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

where

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$; $\Sigma = \{0, 1, 2\}$

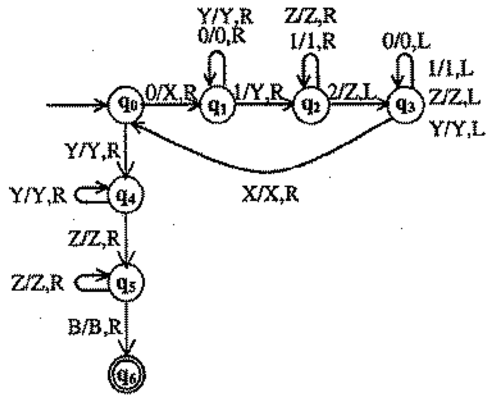
$\Gamma = \{0, 1, 2, X, Y, Z, B\}$; q_0 is the initial state

B is blank character; F = { q_6 } is the final state

δ is shown below using the transition table.

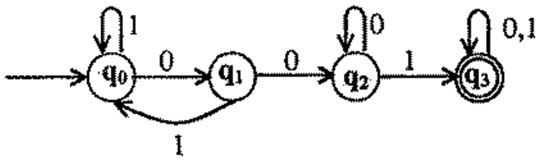
	Γ						
States	0	1	2	Z	Y	X	B
q_0	q_1, X, R				q_4, Y, R		
q_1	q_1, 0, R	q_2, Y,R			q_1, Y,R		
q_2		q_2, 1,R	q_3, Z,L	q_2, Z,R			
q_3	q_3, 0,L	q_3, 1,L		q_3, Z,L	q_3, Y,L	q_0, X,R	
q_4				q_5, Z,R	q_4, Y,R		
q_5				q_5, Z,R			(q_6, B, R)
q_6							

The transition diagram for this can be of the form



Example 3 : Obtain a TM to accept the language $L = \{w \mid w \in (0+1)^*\}$ containing the substring 001.

Solution : The DFA which accepts the language consisting of strings of 0's and 1's having a sub string 001 is shown below :



The transition table for the DFA is shown below :

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

We have seen that any language which is accepted by a DFA is regular. As the DFA processes the input string from left to right in only one direction, TM also processes the input string in only one direction (unlike the previous examples, where the read - write header was moving in both the directions). For each scanned input symbol (either 0 or 1), in whichever state the DFA was in, TM also enters into the same states on same input symbols, replacing 0 by 0 and 1 by 1 and the read - write head moves towards right. So, the transition table for DFA and TM remains same (the format may be different. It is evident in both the transition tables). So, the transition table for TM to recognize the language consisting of 0's and 1's with a substring 001 is shown below :

	0	1	B
q_0	q_1,0, R	q_0, 1, R	-
q_1	q_2, 0, R	q_0, 1, R	-
q_2	q_2, 0, R	q_3, 1, R	-
q_3	q_3, 0, R	q_3, 1, R	q_4, B, R
q_4			

The TM is given by

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

where

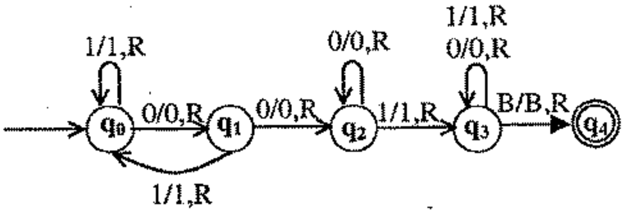
$Q = \{ q_0, q_1, q_2, q_3, q_4 \}$; $\Sigma = \{0, 1\}$

$\Gamma = \{0, 1\}$; δ – is defined already

q_0 is the initial state ; B blank character

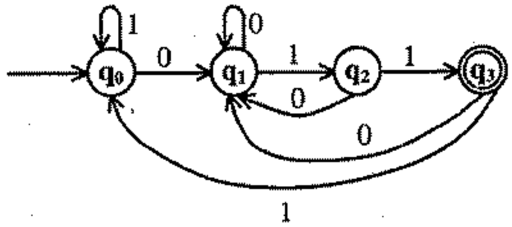
F = { q_4 } is the final state

The transition diagram for this is shown below.



Example 4 : Obtain a Turing machine to accept the language containing strings of 0's and 1's ending with 011.

Solution : The DFA which accepts the language consisting of strings of 0's and 1's ending with the string 001 is shown below :



The transition table for the DFA is shown below :

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

We have seen that any language which is accepted by a DFA is regular. As the DFA processes the input string from left to right in only one direction, TM also processes the input string in only one direction. For each scanned input symbol (either 0 or 1), in whichever state the DFA was in, TM also enters into the same states on same input symbols, replacing 0 by 0 and 1 by 1 and the read - write head moves towards right. So, the transition table for DFA and TM remains same (the format may be different. It is evident in both the transition tables). So, the transition table for TM to recognize the language consisting of 0's and 1's ending with a substring 001 is shown below :

δ	0	1	B
q_0	q_1,0, R	q_0, 1, R	-
q_1	q_1, 0, R	q_2, 1, R	-
q_2	q_1, 0, R	q_3, 1, R	-
q_3	q_1, 0, R	q_0, 1, R	q_4, B, R
q_4	-	-	-

The TM is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
where

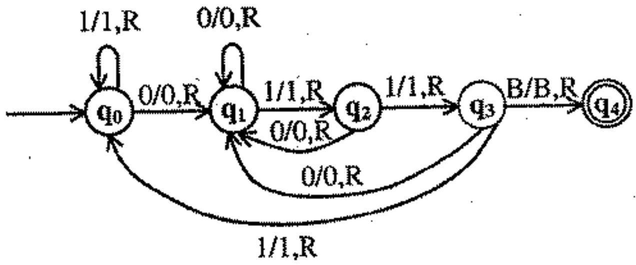
$Q = \{ q_0, q_1, q_2, q_3 \}$; $\Sigma = \{0, 1\}$; $\Gamma = \{0, 1\}$

δ – is defined already

q_0 is the initial state ; B does not appear

F = { q_4 } is the final state

The transition diagram for this is shown below :

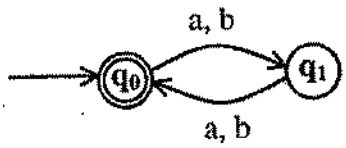


Example 5 : Obtain a Turing machine to accept the language

$L = \{ w | w \text{ is even and } \Sigma = \{ a, b \} \}$

Solution :

The DFA to accept the language consisting of even number of characters is shown below.



The transition table for the DFA is shown below :

	a	b
q_0	q_1	q_1
q_1	q_0	q_0

We have seen that any language which is accepted by a DFA is regular. As the DFA processes the input string from left to right in only one direction, TM also processes the input string in only one direction. For each scanned input symbol (either a or b), in whichever state the DFA was in, TM also enters into the same states on same input symbols, replacing a by a and b by b and the read - write head moves towards right. So, the transition table for DFA and TM remains same (the format may be different). So, the transition table for TM to recognize the language consisting of a's and b's having even number of symbols is shown below :

δ	a	b	B
q_0	q_1 ,a, R	q_1, b, R	q_2, B, R
q_1	q_0, a, R	q_0, b, R	-
q_2	-	-	-

The TM is given by

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

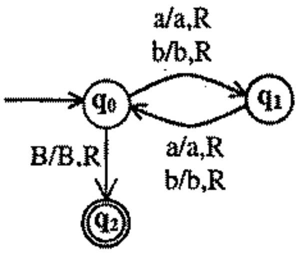
where

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

δ – is defined already ; q_0 is the initial state

B does not appear ; F = { q_2 } is the final state

The transition diagram of TM is given by



Example 6 : Obtain a Turing machine to accept a palindrome consisting of a's and b's of any length.

Solution : Let us assume that the first symbol on the tape is blank character B and is followed by the string which in turn ends with blank character B. Now, we have to design a Turing machine which accepts the string, provided the string is a palindrome. For the string to be a palindrome, the first and the last character should be same. The second character and last but one character in the string should be same and so on. The procedure to accept only string of palindromes is shown below. Let q0 be the start state of Turing machine.

Step 1 : Move the read - write head to point to the first character of the string. The transition for this can be of the form $\delta(q_0, B) = (q_1, B, R)$

Step 2 : In state q_1, if the first character is the symbol a, replace it by B and change the state to q_2 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_1, a) = (q_2, B, R)$$

Now, we move the read - write head to point to the last symbol of the string and the last symbol should be a. The symbols scanned during this process are a's, b's and B. Replace a by a, b by b and move the pointer towards right. The transitions defined for this can be of the form

$$\delta(q_2, a) = (q_2, a, R)$$

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

But, once the symbol B is encountered, change the state to q_3, replace B by B and move the pointer towards left. The transition defined for this can be of the form

$$\delta(q_2, B) = (q_3, B, L)$$

In state q_3, the read - write head points to the last character of the string. If the last character is a, then change the state to q_4, replace a by B and move the pointer towards left. The transitions defined for this can be of the form

$$\delta(q_3, a) = (q_4, B, L)$$

At this point, we know that the first character is a and last character is also a. Now, reset the read - write head to point to the first non blank character as shown in step5.

In state q_3, if the last character is B (blank character), it means that the given string is an odd palindrome. So, replace B by B change the state to q_7 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_3, B) = (q_7, B, R)$$

Step 3 : If the first character is the symbol b, replace it by B and change the state from q_1 to q_5 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_1, b) = (q_5, B, R)$$

Now, we move the read - write head to point to the last symbol of the string and the last symbol should be b. The symbols scanned during this process are a's, b's and B. Replace a by a, b by b and move the pointer towards right. The transitions defined for this can of the form

$$\delta(q_5, a) = (q_5, a, R)$$
$$\delta(q_5, b) = (q_5, b, R)$$

But, once the symbol B is encountered, change the state to q_6, replace B by B and move the pointer towards left. The transition defined for this can be of the form

$$\delta(q_5, B) = (q_6, B, L)$$

In state q_6, the read - write head points to the last character of the string. If the last character is b, then change the state to q_6, replace b by B and move the pointer towards left. The transitions defined for this can be of the form

$$\delta(q_6, b) = (q_4, B, L)$$

At this point, we know that the first character is b and last character is also b. Now, reset the read - write head to point to the first non blank character as shown in step 5.

In state q_6, If the last character is B (blank character), it means that the given string is an odd palindrome. So, replace B by B, change the state to q_7 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_6, B) = (q_7, B, R)$$

Step 4 : In state q_1, if the first symbol is blank character (B), the given string is even palindrome and so change the state to q_7, replace B by B and move the read - write head towards right. The transition for this can be of the form

$$\delta(q_1, B) = (q_7, B, R)$$

Step 5 : Reset the read - write head to point to the first non blank character. This can be done as shown below.

If the first symbol of the string is a, step 2 is performed and if the first symbol of the string is b, step 3 is performed. After completion of step 2 or step 3, it is clear that the first symbol and the last symbol match and the machine is currently in state q_4. Now, we have to reset the read - write head to point to the first nonblank character in the string by repeatedly moving the head towards left and remain in state q_4. During this process, the symbols encountered may be a or b or B (blank character). Replace a by a, b by b and move the pointer towards left. The transitions defined for this can be of the form

$$\delta(q_4, a) = (q_4, a, L)$$
$$\delta(q_4, b) = (q_4, b, L)$$

But, if the symbol B is encountered, change the state to q_1, replace B by B and move the pointer towards right. the transition defined for this can be of the form

$$\delta(q_4, B) = (q_1, B, R)$$

After resetting the read - write head to the first non - blank character, repeat through step 1.

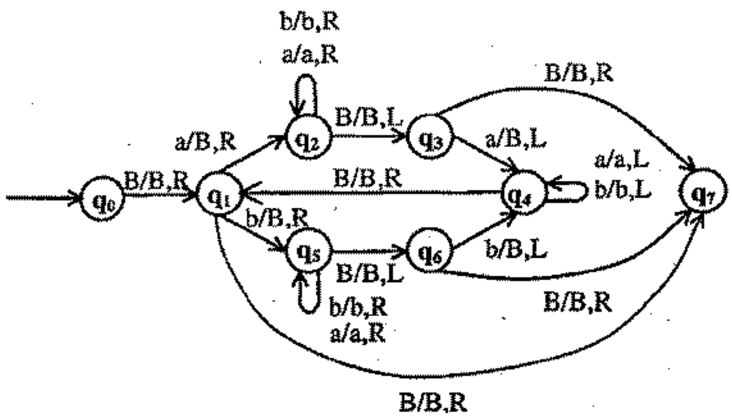
So, the TM to accept strings of palindromes over { a, b } is given by $M = (Q, \Sigma, \delta, q_0, B, F)$

where $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$; $\Sigma = \{a, b\}$; $\Gamma = \{a, b, B\}$; q_0 is the initial state

B is the blank character ; F = { q_7 } ; δ is shown below using the transition table

	Γ		
δ	a	b	B
q_0	-	-	q_1, B, R
q_1	q_2, B, R	q_5, B, R	q_7, B, R
q_2	q_2, a, R	q_2, b, R	q_3, B, L
q_3	q_4, B, L	-	q_7, B, R
q_4	q_4, a, L	q_4, b, L	q_1, B, R
q_5	q_5, a, R	q_5, b, R	q_6, B, L
q_6	-	q_4, B, L	q_7, B, R
q_7	-	-	-

The transition diagram to accept palindromes over { a, b } is given by



The reader can trace the moves made by the machine for the strings abba, aba and aaba and is left as an exercise.

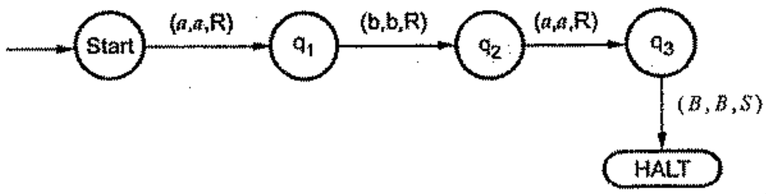
Example 7 : Construct a Turing machine which accepts the language of aba over $\Sigma=\{a,b\}$.

Solution : This TM is only for L = { aba }

We will assume that on the input tape the string 'aba' is placed like this

a	b	a	B	B	....
↑					

The tape head will read out the sequence upto the B character if 'aba' is readout the TM will halt after reading B.



The triplet along the edge written is (input read, output to be printed, direction)

Let us take the transition between start state and q_1 is (a, a, R) that is the current symbol read from the tape is a then as a output a only has to be printed on the tape and then move the tape head to the right. The tape will look like this

a	b	a	B	B	....
	↑				

Again the transition between q_1 and q_2 is (b, b, R). That means read b, print b and move right. Note that as tape head is moving ahead the states are getting changed.

a	b	a	B	B	....
		↑			

The TM will accept the language when it reaches to halt state. Halt state is always a accept state for any TM. Hence the transition between q_3 and halt is (B, B, S). This means read B, print B and stay there or there is no move left or right. Eventhough we write (B, B, L) or (B, B, R) it is equally correct. Because after all the complete input is already recognized and now we simply want to enter into a accept state or final state. Note that for invalid inputs such as abb or ab or bab ..... there is either no path reaching to final state and for such inputs the TM gets stucked in between. This indicates that these all invalid inputs can not be recognized by our TM.

The same TM can be represented by another method of transition table

	a	b	B
Start	(q_1, a, R)	-	-
q_1	-	(q_2, b, R)	-
q_2	(q_3, a, R)	-	-
q_3	-	-	(HALT, B, S)
HALT	-	-	-

In the given transition table, we write the triplet in each row as :

(Next state, output to be printed, direction)

Thus TM can be represented by any of these methods.

Example 8 : Design a TM that recognizes the set $L = \{0^{2n}1^n \mid n \geq 0\}$.

Solution : Here the TM checks for each one whether two 0's are present in the left side. If it match then only it halts and accept the string.

The transition graph of the TM is ,

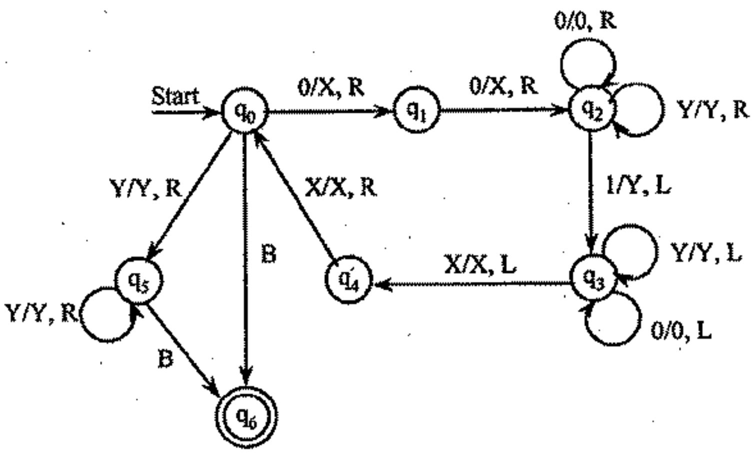


FIGURE : Turing Machine for the given language $L = \{0^{2n}1^n \mid n \geq 0\}$

Example 9 : Design Turing machine to recognize the palindromes of digits { 0, 1 }. Give its state transition diagram also.

Solution : The construction is made by defining moves in the following manner.

i. The machine scans the first input symbol (either 0 or 1), erases (but remembers) it, writes a blank symbol in place and changes state $(q_1 \; or \; q_2)$.

ii. It scans the remaining part without changing the tape symbol until it encounters b. It then moves the read / write head a step left. If the rightmost symbol tallies with the leftmost symbol, the rightmost symbol is erased. Otherwise T. M. halts. The read/write head moves to the left until b is encountered.

iii. The above steps are repeated after changing the states suitably.

The transition table is shown below.

Present State	Tape Symbols 0	1	b
→ q_0	bRq_1	bRq_2	bRq_7
q_1	$0Rq_1$	$1Rq_1$	bLq_3
q_2	$0Rq_2$	$1Rq_2$	bLq_4
q_3	bLq_5	-	bRq_6
q_4	-	bLq_5	bRq_6
q_5	$0Lq_5$	$1Lq_5$	bRq_6
(q_6)	-	-	-

The transition diagram is shown in below figure.

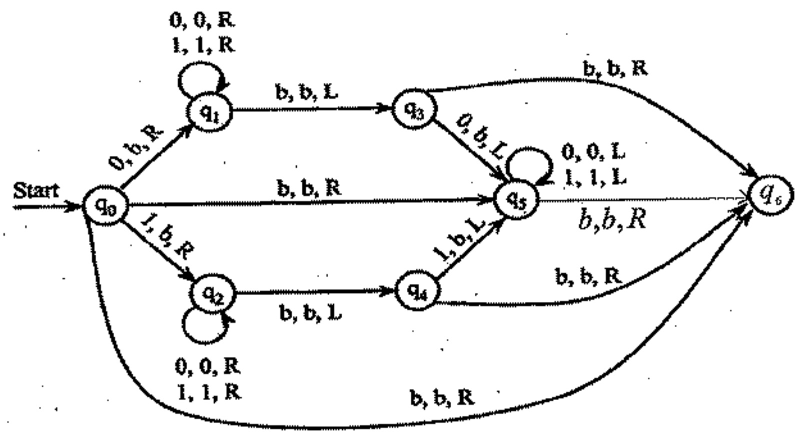


FIGURE : Transition State Diagram for the Palindromes

Example 10 : Design a Turing machine that accepts $L = \{a^n b^n | n \geq 0\}$.

Solution : The logic that we use for the Turing machine to be constructed is,

The Turing machine will remember leftmost a, by replacing it with B, then it moves the tape head right keeping the symbols it scans as it is, until it gets rightmost b, it remembers rightmost b, by replacing it with B, and moves the tape head left keeping the symbols it scans as it is till it reaches the B, on getting B, it moves the tape head one position right and repeats the above cycle if it gets a. If it gets B instead of a, then it is an indication of the fact the string is of the form $a^n b^n$, hence the Turing machine enters into the final state. Therefore, the moves of the Turing machine are given in below table.

	a	b	B
q_0	(q_1, B, R)		(q_4, B, R)
q_1	(q_1, a, R)	(q_1, b, R)	(q_2, B, L)
q_2		(q_3, B, L)	
q_3	(q_3, a, L)	(q_3, b, L)	(q_0, B, R)
q_4			

TABLE : Moves of the Turing Machine for the given language

Therefore, the Turing machine $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{a, b, B\}, \delta, q_0, B, \{q_4\})$, where is given above.
The transition diagram corresponding to the above Table is shown in below figure.

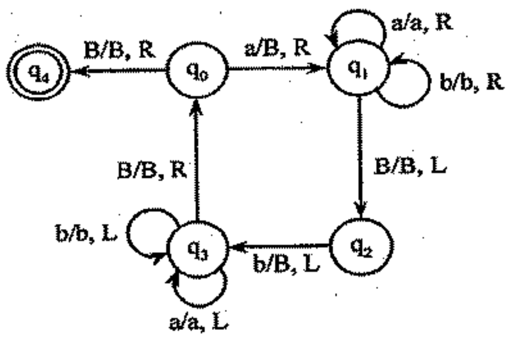


FIGURE : Transition Diagram for the above Table

Example 11 : What does the Turing Machine described by the 5 - tuples,
$(q_0,0,q_0,1,R),(q_0,1,q_1,0,r),(q_0,B,q_2,B,R)$,
$(q_1,0,q_1,0, R)$, $(q_1,1,q_0,1, R)$ *and* (q_1,B,q_2,B,R) .Do when given a bit string as input ?

Solution : The transition diagram of the TM is ,

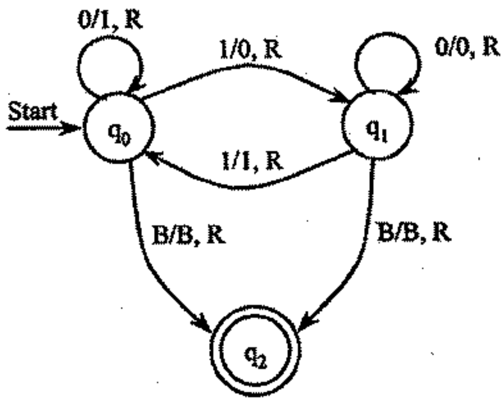


FIGURE : Transition Diagram for the given TM

The TM here reads an input and starts inverting 0's to 1's and 1's to 0's till the first 1. After it has inverted the first 1, it read the input symbol and keeps it as it is till the next 1. After encountering the 1 it starts repeating the cycle by inverting the symbol till next 1. It halts when it encounters a blank symbol.

7.4 COMPUTABLE FUNCTIONS

A Turing machine is a language acceptor which checks whether a string x is accepted by a language L. In addition to that it may be viewed as computer which performs computations of functions from integers to integers. In traditional approach an integer is represented in unary, an integer $i \geq 0$ is represented by the string 0^i .

Example 1 : 2 is represented as 0^2. If a function has k arguments, $i_1, i_2, ... i_k$, then these integers are initially placed on the tape separated by 1's, as $0^{i}10^{i_2}1 .... 10^{i_k}$.

If the TM halts (whether in or not in an accepting state) with a tape consisting of 0's for some m, then we say that $f(i_1, i_2, ... i_k) = m$, where f is the function of k arguments computed by this Turing machine.

Example 2 :

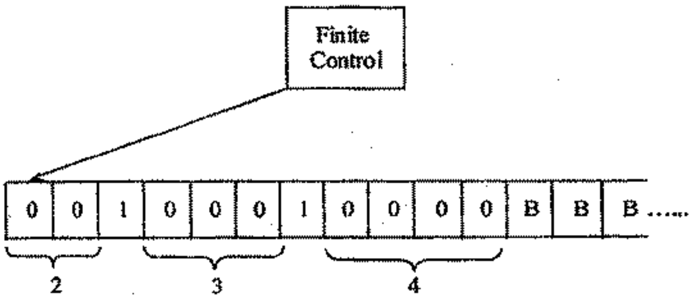
Consider a function in C.

```
int sum ( int x, int y, int z )
{       int s ;
        s = x + y + z ;
        return s;
}
```

Suppose this function is invoked using statement,

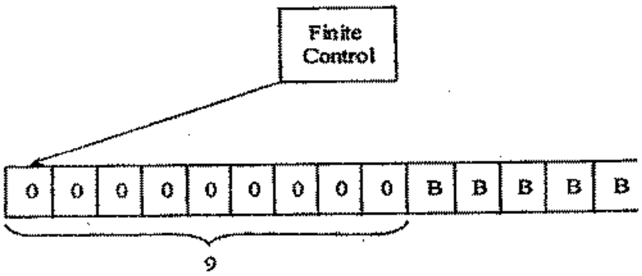
```
        c = sum ( 2, 3, 4 ) ;
```

After invoking sum ( ) , c will have the value 9. The same computation can be performed by Turing machine also. Initially, the Turing machine will have the arguments of sum( ) i.e., 2, 3, 4 on its tape as shown in figure (a).



(a) Before Computation

This Turing machine performs the sum of these arguments. After some moves it halts with the tape containing value 9, as shown in figure (b).



(b) After Computation

FIGURE : Elements on Tape to Compute Sum

Note that a Turing machine may compute a function of one argument, a function of two arguments and so on. The Turing machine given in figure can perform sum of two arguments or three arguments or in general sum of any finite number of arguments.

If TM M computes function f of k arguments i then f need not have a value for all different k - tuples of integers $i_1, i_2, ... i_k$ .if $f(i_1, i_2, ... i_k)$ is defined for all, $i_1 ... i_k$, then we say f is a total recursive function, otherwise we say f is partial recursive function. Total recursive functions are analogues to recursive language because they are computed by TM that always halts. Partial recursive function are analogues to recursively enumerable languages. Because they are computed by TM that may or may not halt. Examples of total recursive functions, all common arithmetic functions on integers, such as multiplication etc, are total recursive functions.

Example 3 : Construct Turing machine to find proper subtraction m - n is defined to be m - n for $m \geq n$ and zero for m < n .

Solution : The TM $M = (\{q_0, q_1, ...q_6\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \phi)$ defined below, started with $0^m / 0^n$ on its tape, halts with 0^{m-n} on its tape. M repeatedly replaces its leading 0 by blank, then searches right for a 1 followed by a 0 and changes the 0 to 1. Next, M moves left until it encounters a blank and then repeats the cycle. The repetition ends if

i. Searching right for a 0, M encounters a blank. Then, the n 0's in $0^m\ 10^n$ have all changed to 1's and n + 1 of the m 0's have been changed to B. M replaces the n + 1 1's by a 0 and n B's leaving m - n 0's on its tape.

ii. Beginning the cycle, M cannot find a 0 to change to a blank, because the first m 0 is already have been changed. Then $n \geq m$. So m - n = 0. M replaces all remaining 1's and 0's by B.

The function δ is described below.

1. $\delta(q_0, 0) = (q, B, R)$

   Begin the cycle, Replace the leading 0 by B.

2. $\delta(q_1, 0) = (q_1, 0, R)$

   $\delta(q_1, 1) = (q_2, 1, R)$

   Search right, looking for the first 1.

3. $\delta(q_2, 1) = (q_2, 1, R)$

   $\delta(q_2, 0) = (q_3, 1, L)$

   Search right past 1's until encountering a 0, change that to 1.

4. $\delta(q_3, 0) = (q_3, 0, L)$

   $\delta(q_3, 1) = (q_3, 1, L)$

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

   Move left to a blank. Enter state q_0 to repeat the cycle.

5. $\delta(q_2, B) = (q_4, B, L)$

$$\delta(q_4,1) = (q_4, B, L)$$
$$\delta(q_4,0) = (q_4, 0, L)$$
$$\delta(q_4,0) = (q_6, 0, R)$$

If in state q_2 a B is encountered before a 0, we have situation (i) described above. Enter state q_4 and move left, changing all 1's to B 's until encountering a 'B'. This B is changed back to a 0, state q_6 is entered, and M halts.

6. $\delta(q_0,1) = (q_5, B, R)$

$$\delta(q_5,0) = (q_5, B, R)$$
$$\delta(q_5,1) = (q_5, B, R)$$
$$\delta(q_5,B) = (q_6, B, R)$$

If in state q_0 a 1 is encountered instead of a 0, the first block of 0's has been exhausted, as in situation (ii) above. M enters state q_5 to erase the rest of the tape, then enters q_6 and halts.

Example 4 : Design a TM which computes the addition of two positive integers.

Solution : Let TM $M = (Q, \{0, 1, \#\}, \delta, s)$ computes the addition of two positive integers m and n. It means, the computed function f (m, n) defined as follows :

$$f(m,n) = \begin{cases} m + n (If\ m, n \geq 1) \\ 0 \qquad (m = n = 0) \end{cases}$$

1 on the tape separates both the numbers m and n. Following values are possible for m and n.

1. $m = n = 0$ (# 1 # ...... is the input),
2. m = 0 and $n \neq 0$ ($\#10^n\#$ .... is the input),
3. $m \neq 0$ and n = 0 ($\#0^m1\#$ ... is the input), and
4. $m \neq 0$ and $n \neq 0$ ($\#0^m10^n\#$ ..... is the input)

Several techniques are possible for designing of M, some are as follows :

(a) M appends (writes) m after n and erases the m from the left end.

(b) M writes 0 in place of 1 and erases one zero from the right or left end . This is possible in case of $n \neq 0$ or $m \neq 0$ only. If m =0 or n = 0 then 1 is replaced by #.

We use techniques (b) given above. M is shown in below figure.

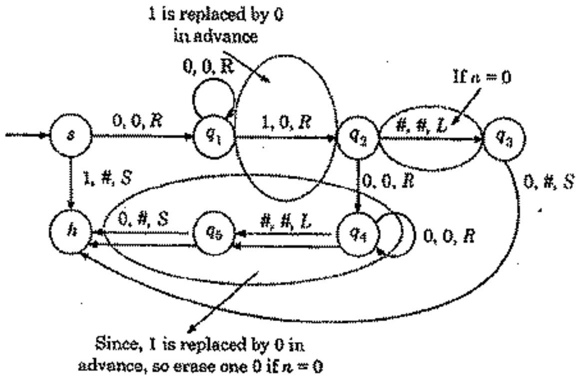


FIGURE : TM for addition of two positive integers

7.5 RECURSIVELY ENUMERABLE LANGUAGES

A language L over the alphabet Σ is called recursively enumerable if there is a TM M that accept every word in L and either rejects (crashes) or loops for every word in language L' the complement of L.

Accept (M) = L

Reject (M) + Loop (M) = L'

When TM M is still running on some input (of recursively enumerable languages) we can never tell whether M will eventually accept if we let it run for long time or M will run forever (in loop).

Example : Consider a language (a + b) * bb (a + b) *.

TM for this language is,

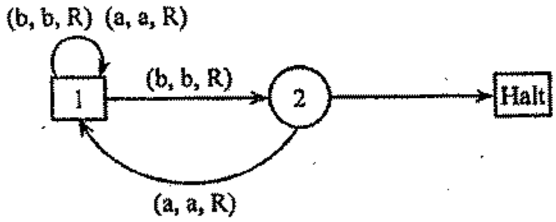


FIGURE : Turing Machine for (a + b) * bb (a + b) *

Here the inputs are of three types.

1. All words with bb = accepts (M) as soon as TM sees two consecutive b's it halts.
2. All strings without bb but ending in b = rejects (M). When TM sees a single b, it enters state2. If the string is ending with b, TM will halt at state 2 which is not accepting state. Hence it is rejected.
3. All strings without bb ending in 'a' or blank 'B' = loop (M) here when the TM sees last a it enters state 1. In this state on blank symbol it loops forever.

Recursive Language

A language L over the alphabet Σ is called recursive if there is a TM M that accepts every word in L and rejects every word in L' i. e.,

accept (M) = L
reject (M) = L'
loop (M) = ϕ.

Example : Consider a language b (a + b) * . It is represented by TM as :

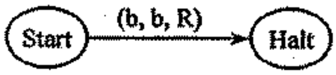


FIGURE : Turing Machine for b (a + b) *

This TM accepts all words beginning with 'b' because it enters halt state and it rejects all words beginning with a because it remains in start state which is not accepting state.

A language accepted by a TM is said to be recursively enumerable languages. The subclass of recursively enumberable sets (r. e) are those languages of this class are said to be recursive sets or recursive language.

7.6 CHURCH'S HYPOTHESIS

According to church's hypothesis, all the functions which can be defined by human beings can be computed by Turing machine. The Turing machine is believed to be ultimate computing machine.

The church's original statement was slightly different because he gave his thesis before machines were actually developed. He said that any machine that can do certain list of operations will be able to perform all algorithms. TM can perform what church asked, so they are possibly the machines which church described.

Church tied both recursive functions and computable functions together. Every partial recursive function is computable on TM. Computer models such as RAM also give rise to partial recursive functions. So they can be simulated on TM which confirms the validity of churches hypothesis.

Important of church's hypothesis is as follows .

1. First we will prove certain problems which cannot be solved using TM.

2. If churches thesis is true this implies that problems cannot be solved by any computer or any programming languages we might every develop.

3. Thus in studying the capabilities and limitations of Turing machines we are indeed studying the fundamental capabilities and limitations of any computational device we might even construct.

It provides a general principle for algorithmic computation and, while not provable, gives strong evidence that no more powerful models can be found.

7.7 COUNTER MACHINE

Counter machine has the same structure as the multistack machine, but in place of each stack is a counter. Counters hold any non negative integer, but we can only distinguish between zero and non zero counters.

Counter machines are off - line Turing machines whose storage tapes are semi - infinite, and whose tape alphabets contain only two symbols, Z and B (blank). Furthermore the symbol Z, which serves as a bottom of stack marker, appears initially on the cell scanned by the tape head and may never appear on any other cell. An integer i can be stored by moving the tape head i cells to the right of Z. A stored number can be incremented or decremented by moving the tape head right or left. We can test whether a number is zero by checking whether Z is scanned by the head, but we cannot directly test whether two numbers are equal.

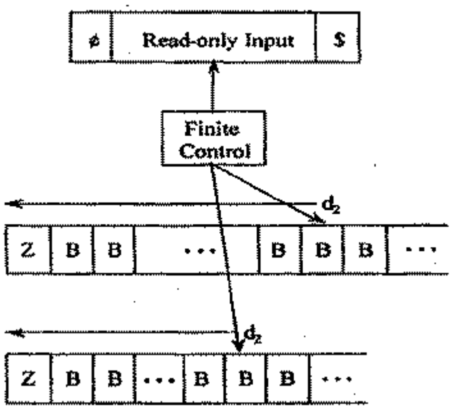


FIGURE : Counter Machine

¢ and \$ are customarily used for end markers on the input. Here Z is the non blank symbol on each tape. An instantaneous description of a counter machine can be described by the state, the input tape contents, the position of the input head, and the distance of the storage heads from the symbol Z (shown here as d_1 and d_2). We call these distances the counts on the tapes. The counter machine can only store a count an each tape and tell if that count is zero.

Power of Counter Machines

- Every language accepted by a counter Machine is recursively enumerable.
- Every language accepted by a one - counter machine is a CFL so a one - counter machine is a special case of one - stack machine i. e., a PDA

7.8 TYPES OF TURING MACHINES

Various types of Turing Machines are :

i. With multiple tapes.
ii. With one tape but multiple heads.
iii. With two dimensional tapes.
iv. Non deterministic Turing machines.

It is observed that computationally all these Turing Machines are equally powerful. That means one type can compute the same that other can. However, the efficiency of computation may vary.

1. Turing machine with Two - Way Infinite Tape :

This is a TM that have one finite control and one tape which extends infinitely in both directions.

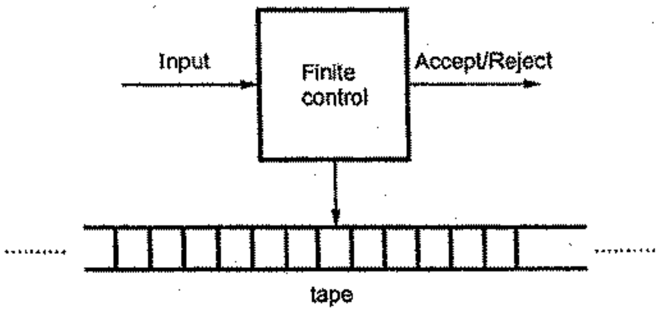


FIGURE : TM with infinite Tape

It turns out that this type of Turing machines are as powerful as one tape Turing machines whose tape has a left end.

2. Multiple Turing Machines :

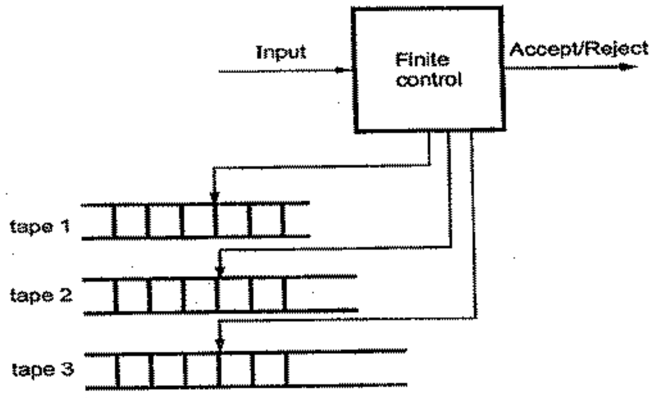


FIGURE : Multiple Turing Machines

A multiple Turing machine consists of a finite control with k tape heads and k tapes, each tape is infinite in both directions. On a single move depending on the state of the finite control and the symbol scanned by each of the tape heads, the machine can

1. Change state.
2. Print a new symbol on each of the cells scanned by its tape heads.
3. Move each of its tape heads, independently, one cell to the left or right or keep it stationary.

Initially, the input appears on the first tape and the other tapes are blank.

3. Nondeterministic Turing Machines :

A nondeterministic Turing machine is a device with a finite control and a single, one way infinite tape. For a given state and tape symbol scanned by the tape head, the machine has a finite number of choices for the next move. Each choice consists of a new state, a tape symbol to print, and a direction of head motion. Note that the non deterministic TM is not permitted to make a move in which the next state is selected from one choice, and the symbol printed and / or direction of head motion are selected from other choices. The non deterministic TM accepts its input if any sequence of choices of moves leads to an accepting state.

As with the finite automaton, the addition of nondeterminism to the Turing machine does not allow the device to accept new languages.

4. Multidimensional Turing Machines :

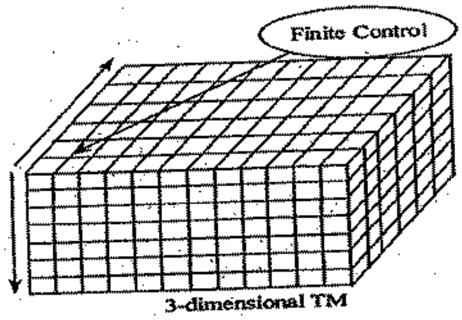


FIGURE : Multidimensional Turing Machine

The multidimensional Turing machine has the usual finite control, but the tape consists of a k - dimensional array of cells infinite in all 2k directions, for some fixed k. Depending on the state and symbol scanned, the device changes state, prints a new symbol, and moves its tape head in one of 2 k directions, either positively or negatively, along one of the k axes. Initially, the input is along one axis, and the head is at the left end of the input.At any time, only a finite number of rows in any dimension contains nonblank symbols, and these rows each have only a finite number of nonblank symbols

5. Multihead Turing Machines :

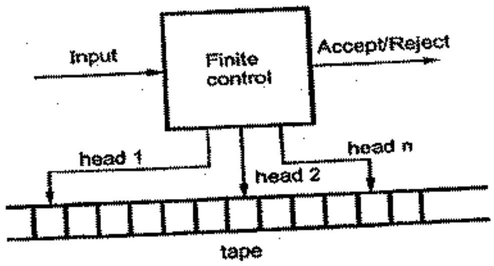


FIGURE : Multihead Turing Machine

A k - head Turing machine has some fixed number, k, of heads. The heads are numbered 1 through k, and a move of the TM depends on the state and on the symbol scanned by each head. In one move, the heads may each move independently left, right or remain stationary.

6. Off - Line Turing Machines :

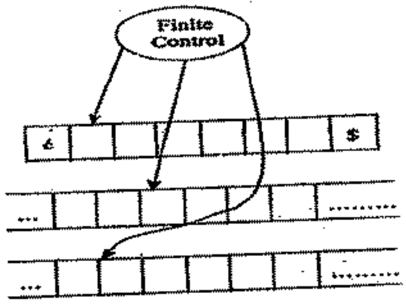


FIGURE : Off - line Turing Machine

An off - line Turing machine is a multitape TM whose input tape is read - only. Usually we surround the input by end markers, ¢ on the left and \$ on the right. The Turing machine is not allowed to move the input tape head off the region between ¢ and \$.

Off - line TM is just a special case of the multitape TM, and is no more powerful than any of the models we have considered. Conversely, an off - line TM can simulate any TM M by using one more tape than M. The first thing the off - line TM does is copy its own input onto the extra tape, and it then simulates M as if the extra tape were M's input.

7. Multistack Machines :

A deterministic two - stack machine is a deterministic Turing machine with a read only input and two storage tapes. If a head moves left on either tape, a blank is printed on that tape.

Multistack machine and counter machines are restricted Turing machines equivalent to the basic model.

7.9 COMPARISON OF FM, PDA AND TM

Basically have discussed three models viz. finite automata or finite machines (FM), Pushdown automata (PDA) and Turing machine (TM). We will now discuss the comparison between these models,

1. The finite machine is of two types - deterministic finite state machine and non deterministic finite state machine. Both of these DFA and NFA accept regular language only. Hence both the machines have equal power i. e. DFA = NFA.
2. We have then learn push down automata again, pushdown automata consists of two types of models deterministic PDA and Non deterministic PDA. The advantage of PDA over FA is that PDA has a memory and hence PDA accepts large class of languages than FA. Hence PDA has more power than FA. The non deterministic PDA accepts the language of context free grammar power of DPDA is less than NPDA as NPDA accepts a larger class of CFL.
3. The class of two stack or n - stack PDA has more power than one stack DPDA or NPDA. Hence two - stack / n - stack PDAS are more powerful.
4. Turing machines can be programmed. Hence TM accepts very very large class of languages. TM, therefore is the most powerful computational model.

TM > PDA > FM

TM accepts regular and non - regular languages ; context free and context sensitive languages as well.

REVIEW QUESTIONS

Q1. Explain Turing machine .

Answer :

For Answer refer to Topic : 7.2, Page No : 7.1.

Q2. Differentiate between TM and PDA.

Answer :

For Answer refer to Topic : 7.2.5, Page No : 7.6.

Q3. Obtain a Turing machine to accept the language $L = \{ 0^n 1^n \mid n \geq 1 \}$.

Answer :

For Answer refer to example - 1 , Page No : 7.6.

Q4. Obtain a Turing machine to accept the language $L(M) = \{ 0^n 1^n 2^n \mid n \geq 1 \}$

Answer :

For Answer refer to example - 2 , Page No : 7.11.

Q5. Obtain a TM to accept the language $L = \{w \mid w \in (0+1)^*\}$ containing the substring 001.

Answer :

For Answer refer to example - 3 , Page No : 7.14.

Q6. Obtain a Turing machine to accept the language containing strings of 0's and 1's ending with 011.

Answer :

For Answer refer to example - 4 , Page No : 7.16.

Q7. Obtain a Turing machine to accept the language $L = \{ w \mid w \text{ is even and } \Sigma = \{ a, b \} \}$

Answer :

For Answer refer to example - 5 , Page No : 7.17.

Q8. Obtain a Turing machine to accept a palindrome consisting of a's and b's of any length.

Answer :

For Answer refer to example - 6 , Page No : 7.19.

Q9. Construct a Turing machine which accepts the language of aba over $\Sigma = \{a,b\}$.

Answer :

For Answer refer to example - 7 , Page No : 7.22.

Q10. Design a TM that recognizes the set $L = \{0^{2n}1^n \mid n \geq 0\}$.

Answer :

For Answer refer to example - 8 , Page No : 7.23.

Q11. Design Turing machine to recognize the palindromes of digits { 0, 1 }. Give its state transition diagram also.

Answer :

For Answer refer to example - 9 , Page No : 7.24.

Q12. Design a Turing machine that accepts $L = \{a^n b^n | n \geq 0\}$.

Answer :

For Answer refer to example - 10 , Page No : 7.25.

Q13. What does the Turing Machine described by the 5 - tuples,

$(q_0,0,q_0,1,R),(q_0,1,q_1,0,r),(q_0,B,q_2,B,R)$,

$(q_1,0,q_1,0,R)$, $(q_1,1,q_0,1,R)$ *and* (q_1,B,q_2,B,R). Do when given a bit string as input ?

Answer :

For Answer refer to example - 11 , Page No : 7.26.

Q14. Write a short notes on computable functions.

Answer :

For Answer refer to Topic : 7.4, Page No : 7.26.

Q15. Construct Turing machine to find proper subtraction m - n is defined to be m - n for $m \geq n$ and zero for m < n .

Answer :

For Answer refer to example - 3 , Page No : 7.28.

Q16. Design a TM which computes the addition of two positive integers.

Answer :

For Answer refer to example - 4 , Page No : 7.29.

Q17. Write about recursively Enumerable Languages .

Answer :

For Answer refer to Topic : 7.5, Page No : 7.30.

Q18. Explain about church's Hypothesis.

Answer :

For Answer refer to Topic : 7.6, Page No : 7.31.

Q19. Explain about counter machine with a neat diagram.

Answer :

For Answer refer to Topic : 7.7, Page No : 7.32.

Q20. List and explain various types of Turing Machines.

Answer :

For Answer refer to Topic : 7.8, Page No : 7.33.

OBJECTIVE TYPE QUESTIONS

1. The no.of symbols necessary to simulate any TM with m symbols & n states is
   (a) $4mn+m$ (b) mn (c) $8mn+4m$ (d) $m+n$
2. Find the false statement.
   (a) Turing machine is simple mathematical model of general purpose computer.
   (b) Turing machine is not capable to performing any calculation which can be performed by computer
   (c) We construct Turing machine to accept a given language
   (d) We construct Turing machine to carry out some algorithm
3. Which of the following classes of Turing machine is not equivalent to the class of standard Turing machine?
   (a) Non-deterministic Turing machines
   (b) Turing machines with stay option
   (c) Turing machines with semi-infinite tapes
   (d) All of these
4. Choose the correct statements
   (a) Every recursive language is recursively enumerable
   (b) $L\{a^n b^n c^n\}$ is recursively enumerable
   (c) Recursive languages are closed under union
   (d) All
5. A TM is more powerful than Finite state machine because
   (a) it has the capability to remember arbitrary long input symbols
   (b) tape movement is confined to one direction
   (c) it has no finite state control
   (d) none
6. An Finite state machine can be considered to be a TM
   (a) a finite tape length, rewinding capability and bi-directional tape movement.
   (b) a finite tape length, without rewinding and bi-directional movement
   (c) a finite tape length, without rewinding capability and unidirectional tape movement
   (d) a finite tape length, with rewinding and unidirectional movement

7. Turing machines can move how in memory?

(a) It cannot move. (b) forward and backward

(c) backward (d) forward

8. Turing machines use what as their memory?

(a) infinite tape (b) finite tape

(c) RAM (d) ROM

9. Turing machines can do----------

(a) less than a real computer can do

(b) everything that a real computer can do

(c) more than a real computer can do.

(d) Nothing

10. Turing machines are similar to finite automaton but have ------

(a) unlimited and read-write memory

(b) finite and read-write memory

(c) unlimited and read-only memory

(d) finite and read-only memory.

11. Comparing TM and computers we find

(a) They cannot be compared

(b) Both are Equivalent

(c) TM have more computational power

(d) Computers have more computational power

12. The class of TMs is equivalent to the class of

(a) Type 3 Grammars (b) Type 2 Grammars

(c) Type 1 Grammars (d) Type 0 Grammars

13. The class of unrestricted languages corresponds to

(a) FA (b) PDA (c) LBA (d) TM

14. Any TM with m symbols & n states can be simulated by another TM with just 2 symbols & less than

(a) mn states (b) $8mn+4$ states (c) $4mn+8$ states (d) $8mn$ states

15. Which statement is false?

    (d) Turing machine is simple mathematical model of general purpose computer.

    (c) Turing machine is not capable of performing any calculation which can be performed by computer

    (b) We construct Turing machine to accept a given language

    (a) we construct Turing machine to carry out some algorithm

16. By giving Turing machine more complex power we can increase the power of the Turing Machine

    (a) Absolutely False (b) May not be True

    (c) May be True (d) Absolutely True

17. The definition of Turing machines is robust because.....

    (a) Turing machine has nothing to do with robustness.

    (b) certain changes (such as many tapes) result in machines of equivalent power.

    (c) turing machines will not crash for any input string.

    (d) functional testing of turing machines finds no errors.

18. A Turing machine computes by going from one configuration to another. We say that configuration C_1 yields configuration C_2 if the Turing Machine can legally move from C_1 to C_2 in ------

    (a) an infinite number of steps

    (b) a single step

    (c) a finite number of steps

    (d) none of the above

19. In a Turing machine for a state q and two strings u and v over the tape alphabet writing 'uqv', specifies that the current state is q------

    (a) the tape contents are uv, and the current head location is the first symbol of v.

    (b) the tape contents are uv, and the current head location is the first symbol of u.

    (c) the tape contents are uqv, and the current head location is the first symbol of v.

    (d) The tape contents are uqv, and the current head location is the first symbol of u.

20. Turing machines output accept if they enter an accept state. When do Turing machine output reject?

    (a) Never

    (b) When they enter a reject state

    (c) When they never end

    (d) When they are not in an accept state and halts

21. Consider the Turing Machine M described the transition table.

Present State	Tape Symbols				
	0	1	x	y	b
q_1	xR_{q2}		-		bR_{q5}
q_2	$0bR_{q2}$	yL_{q3}	-	xR_{q2}	
q_3	$0L_{q4}$		xR_{q5}	xR_{q3}	
q_4	$0L_{q4}$		xR_{q1}		
q_5				xR_{q5}	bR_{q6}
q_6					

q_6 is the final state.

Refer to the Turing Machine whose transition diagram is given above. What is the final ID when string 011 is processed?

(a) $xyq_5 1$ (b) $xyq_6 yx$ (c) $xxyybq_6$ (d) $xyq_6 1$

22. Consider the transition table of a Turing machine :

Present State	Tape symbols		
	b	o	1
q_1	$1L_{q2}$	$0R_{q1}$	
q_2	bR_{q3}	$0L_{q2}$	$1L_{q2}$
q_3	bR_{q4}	bR_{q5}	
q_4	$0R_{q5}$	$0R_{q4}$	$1R_{q4}$
q_5	$0L_{q2}$		

q_5 is the final state.

Computation sequence of string 00 leads to?

(a) Error (b) $bbbb_{q5}0000$ (c) $bbb_{q5}000$ (d) $bb_{q5}00$

23. The grammar generated by production rules $S \to aSBc \mid abc,\ cB \to Bc,\ aB \to aa$ is

(a) $a^n b^n c^n\ n \leq 0$ (b) $a^n b^n c^n, n > 0$

(c) $a^n b^n c^n, n \geq 0$ (d) $a^n b^n c^n, n > 1$

24. The grammar generated by production rules $S \to A \mid Sc, A \to ab \mid aAb$ is

(a) $a^n b^n c^i \; n \geq 0 \; and \; c > 0$ (b) $a^n b^n c^i, n > 0 \; and \; c > 0$

(c) $a^n b^n c^i \; n \geq 0 \; and \; c \geq 0$ (d) $a^n b^n c^i, n > 0 \; and \; c \geq 0$

25. Consider a new type of turing machine where the head can move left and move right but cannot stay put. This new type of turing machine is.........

(a) not comparable.

(b) More powerful than the original Turing mahcine

(c) equivalent in power to the original Turing machine

(d) less powerful than the original Turing mahcine

26. The statement "Standard TM accepts the same languages as are accepted by a stay TM" is

(a) Always false. (b) True for all languages

(c) True only if languages is regular (d) True only if languages is a CFL

27. Find the false statement

(a)Standard TM is equivalent to linear bounded automata

(b) Standard Turing machine(TM) is equivalent to multi tape TM

(c) Standard TM is equivalent to non deterministic TM

(d) None

28. Which of the following is true: Read Write head can move

(a) to the left of right endmarker in LBA

(b) to the right of right endmarker in LBA

(c) to the right of left endmarker in LBA

(d)to the left of left endmarker in LBA

29. LBA is

(a) restricted T.M. from both sides (b) unrestricted T.M.

(c) restricted T.M. from one side (d) none

30. Which automata is associated with Context Sensitive Language? (Give the best answer)

(a) Linear Bounded Automata (b) Pushdown Automata

(c) Finite Automata (d) Turing Machine

31. Refer to the Turing Machine whose transition diagram as given above in question 21, What is the final ID when string 0011 is processed?

(a) $xyq_5 1$ (b) $xyq_6 yx$ (c) $xxyybq_6$ (d) $xyq_6 1$

32. Which of the following is not a variant of the standard Turing Mahine
    (a) Universal Turing Machine (b) Linear Bounded Automata
    (c) Pushdown Automata (d) None of the above.
33. Let B be a Linear bounded automata. Then grammar corresponding to L(B) is
    (a) Regular grammar (b) Unrestricted grammar
    (c) Context free language (d) Context sensitive language
34. The Linear bounded automata is a variant of
    (a) Finite Automata (b) Turing Machine
    (c) Pushdown Automata (d) None of these
35. Non-Deterministic Turing Machines are more powerful than deterministic Turing Machine
    (a) Absolutely False (b) May not be True
    (c) May be True (d) Absolutely True
36. Many models of general purpose computation exist. Some are very similar to the original Turing machine, others can be very different than the original.
    All of these models are equivalent in power if................
    (a) there is no model if everything is equivalent to everything else!
    (b) they have unrestricted access to unlimited memory, and satisfy certain reasonable requirements like performing only a finite amount of work in a single step.
    (c) satisfy certain reasonable requirements like performing only a finite amount of work in a single step.
    (d) they have unrestricted access to unlimited memory.

ANSWER KEY

1.(b)	2.(c)	3.(d)	4.(b)	5.(a)	6.(a)	7.(a)	8.(d)	9.(d)	10.(b)
11.(d)	12.(d)	13.(d)	14.(a)	15.(d)	16.(a)	17.(c)	18.(a)	19.(d)	20.(b)
21.(b)	22.(b)	23.(b)	24.(b)	25.(a,c)	26.(b)	27.(a)	28.(a,c)	29.(a)	30.(a)
31.(b)	32.(c)	33.(d)	34.(b)	35.(a)	36.(b)				

Formal Languages And Automata Theory

UNIT 5

COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF AERONAUTICAL ENGINEERING

DUNDIGAL, HYDERABAD - 500 043

8

COMPUTABILITY THEORY

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

- Chomsky hierarchy of Languages
- Linear Bounded Automata and CSLs
- LR (0) Grammar
- Decidability of problems
- UTM and PCP
- P and NP problems

8.1 CHOMSKY HIERARCHY OF LANGUAGES

Chomsky has classified all grammars in four categories (type 0 to type 3) based on the right hand side forms of the productions.

(a) Type 0

These types of grammars are also known as phrase structured grammars, and RHS of these are free from any restriction. All grammars are type 0 grammars.

Example : productions of types $AS \rightarrow aS,\ SB \rightarrow Sb, S \rightarrow \in$ are type 0 production.

(b) Type 1

We apply some restrictions on type 0 grammars and these restricted grammars are known as type 1 or **context - sensitive grammars** (CSGs). Suppose a type 0 production $\gamma\alpha\delta \rightarrow \gamma\beta\delta$ and the production $\alpha \rightarrow \beta$ is restricted such that $|\alpha| \leq |\beta|$ and $\beta \neq \in$. Then these type of productions is known as type 1 production. If all productions of a grammar are of type 1 production, then grammar is known as type 1 grammar. The language generated by a context - sensitive grammar is called context - sensitive language (CSL).

In CSG, there is left context or right context or both. For example, consider the production $\alpha A\beta \rightarrow \alpha a\beta$. In this, α is left context and β is right context of A and A is the variable which is replaced.

The production of type $S \rightarrow \in$ is allowed in type 1 if $\in$ is in L(G), but S should not appear on right hand side of any production.

Example : productions $S \rightarrow AB, S \rightarrow \in, A \rightarrow c$ are type 1 productions, but the production of type $A \rightarrow Sc$ is not allowed. Almost every language can be thought as CSL.

Note : If left or right context is missing then we assume that $\in$ is the context.

(c) Type 2

We apply some more restrictions on RHS of type 1 productions and these productions are known as type 2 or context - free productions. A production of the form $\alpha \rightarrow \beta$, where $\alpha, \beta \in (V \cup \Sigma)^*$ is known as type 2 production. A grammar whose productions are type 2 production is known as type 2 or context - free grammar (CFG) and the languages generated by this type of grammars is called context - free languages (CFL).

Example : $S \rightarrow S + S, S \rightarrow S * S, S \rightarrow id$ are type 2 productions.

(d) Type 3

This is the most restricted type. Productions of types $A \rightarrow a$ or $A \rightarrow aB|Ba$, where $A, B \in V$, and $a \in \Sigma$ are known as type 3 or regular grammar productions. A production of type $S \rightarrow \in$ is also allowed, if $\in$ is in generated language.

Example : productions $S \rightarrow aS,\ S \rightarrow a$ are type 3 productions.

Left - linear production : A production of type $A \rightarrow Ba$ is called left - linear production.

Right - linear production : A production of type $A \rightarrow aB$ is called right - linear production. A left - linear or right - linear grammar is called regular grammar. The language generated by a regular grammar is known as regular language.

A production of type $A \to w$ or $A \to wB$ or $A \to Bw$, where $w \in \Sigma^*$ can be converted into the forms $A \to a$ or $A \to aB$ or $A \to Ba$, where $A, B \in V$ and $a \in \Sigma$.

Example : $A \to 10A$ can be replaced by productions $A \to 1B$, where B is a new variable and $B \to 0A$.

In general, if $A \to a_1 a_2 a_3 \ldots\ldots a_n a_{n+1} B$, then this production can replaced by the following productions.

$$A \to a_1 B_1,$$
$$B_1 \to a_2 B_2,$$
$$B_2 \to a_3 B_3,$$
$$\ldots$$
$$B_n \to a_{n+1} B$$

Similar result is obtained for left - linear grammars also.

8.1.1 Hierarchy of grammars

Type 0 or Phrase structured grammar

⇓ Restrictions applied

Type 1 or Context - sensitive grammar

⇓ Restrictions applied

Type 2 or Context - free grammar

⇓ Restrictions applied

Type 3 or Regular grammar

Example : Consider the following and find the type of the grammar.

(a) $S \to Aa,\ A \to c \mid Ba,\ B \to abc$

(b) $S \to aSa \mid c$

(c) $S \to aAS \mid SBb,\ AS \to aAS \mid aS,\ SB \to Sb \mid SBb$

Solution :

(a)

Production			Type
S	$\rightarrow$	Aa	Type 3
A	$\rightarrow$	c	Type 3
A	$\rightarrow$	Ba	Type 3
B	$\rightarrow$	abc	Type 3

So, given productions are of type 3 and hence grammar is regular.

(b)

S	$\rightarrow$	aSa	Type 2
S	$\rightarrow$	c	Type 3

So, given productions are of type 2 and hence grammar is CFG.

Note : We select the higher type and higher type between type 3 and type 2 is type 2).

(c)

S	$\rightarrow$	aAS	Type 2
S	$\rightarrow$	SBb	Type 2
AS	$\rightarrow$	aAS	Type 1
AS	$\rightarrow$	aS	Type 1
SB	$\rightarrow$	Sb	Type 1
SB	$\rightarrow$	SBb	Type 1

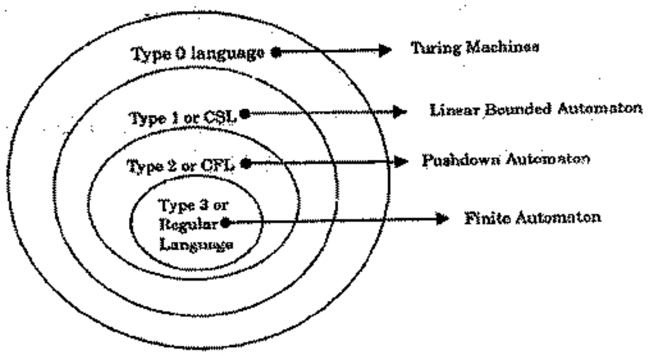
So, given productions are of type 1 and hence grammar is CSG.

8.1.2 Relation Among Grammars and Languages

Type 0 is the super set and type 1 is contained in type 0, type 2 is contained in type 1, and type 3 is contained in type 2.

$$\text{Type } 0 \subseteq \text{Type } 1 \subseteq \text{Type } 2 \subseteq \text{Type } 3$$

8.1.3 Languages and Their Related Automaton



FIGIURE : Languages and their related Automaton

8.2 LINEAR BOUNDED AUTOMATA

The Linear Bounded Automata (LBA) is a model which was originally developed as a model for actual computers rather than model for computational process. A linear bounded automaton is a restricted form of a non deterministic Turing machine.

A linear bounded automaton is a multitrack Turing machine which has only one tape and this tape is exactly of same length as that of input.

The linear bounded automaton (LBA) accepts the string in the similar manner as that of Turing machine does. For LBA halting means accepting. In LBA computation is restricted to an area bounded by length of the input. This is very much similar to programming environment where size of variable is bounded by its data type.

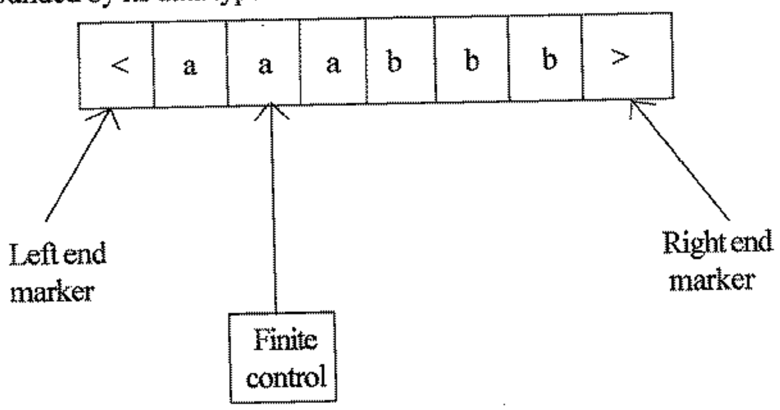


FIGURE : Linear bounded automaton

The LBA is powerful than NPDA but less powerful than Turing machine. The input is placed on the input tape with beginning and end markers. In the above figure the input is bounded by < and >.

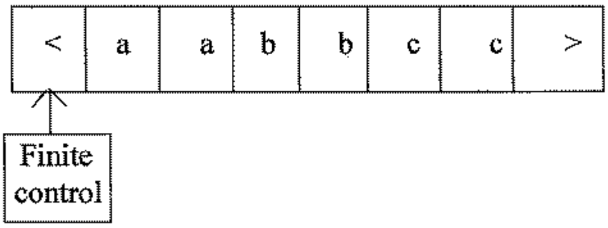
A linear bounded automata can be formally defined as :

LBA is 7 - tuple on deterministic Turing machine with

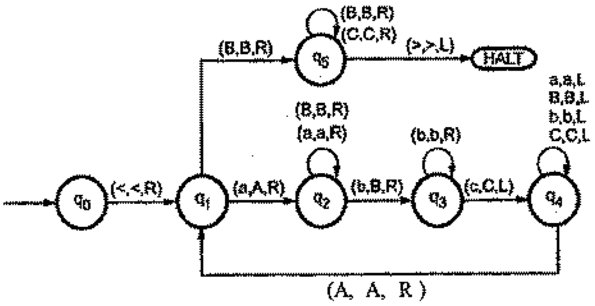
$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ having

1. Two extra symbols of left end marker and right end marker which are not elements of Γ.
2. The input lies between these end markers.
3. The TM cannot replace < or > with anything else nor move the tape head left of < or right of >.

Example : We can construct a language $L = \{a^n\ b^n\ c^n \mid n \geq 1\}$ using LBA as follows.



The input is placed on the input tape which is enclosed within left end marker and right end marker. We will apply the simple logic as : when we read 'a' convert it to A then move right by skipping all a's. On encountering first 'b' we will convert it to B. Then move right by skipping all b's. On receiving first c convert it to C. Move in left direction unless you get A. Repeat the above procedure and convert equal number of a's, b's, and c's to corresponding A's, B's and C's. Finally move completely to the rightmost symbol if it is '>' a right end marker,then HALT. The machine will be :



Simulation : Consider input aabbcc

< aabbcc > ↑	Move right.
< aabbcc > ↑	Convert to A, move right.
< Aabbcc > ↑	Move right.
< Aabbcc > ↑	Convert to B, move right.
< AaBbcc > ↑	Move right.

< AaBbcc > ↑	Convert to C, move left.
< AaBbCc > ↑	Move left
< AaBbCc > ↑	Move left.
< AaBbCc > ↑	Move left.
< AaBbCc > ↑	Move right.
< AaBbCc > ↑	Convert to A, Move right.
< AABbCc > ↑	Move right.
< AABbCc > ↑	Convert to B, Move right.
< AABBCc > ↑	Move right.
< AABBCc > ↑	Convert to C, Move left.
< AABBCC > ↑	Move left continuously by skipping B's.
< AABBCC > ↑	Move right.
< AABBCC > ↑	If we get B, we will move right to check whether all b's and c's are converted to B and C.
< AABBCC > ↑	If we get right end marker '>' then we HALT by accepting the input aabbcc.

Thus in LBA the length of tape exactly equal to the input string and tape head can not move left of '<' right of '>'.

8.3 CONTEXT SENSITIVE LANGUAGES (CSLs)

The context sensitive languages are the languages which are accepted by linear bounded automata. These type of languages are defined by context sensitive grammar. In this grammar more than one terminal or non terminal symbol may appear on the left hand side of the production rule. Along with it, the context sensitive grammar follows following rules :

i. The number of symbols on the left hand side must not exceed number of symbols on the right hand side.
ii. The rule of the form $A \rightarrow \in$ is not allowed unless A is a start symbol. It does not occur on the right hand side of any rule.

The classic example of context sensitive language is $L = \{a^n\ b^n\ c^n \mid n \geq 1\}$. The context sensitive grammar can be written as :

S	→	aBC
S	→	SABC
CA	→	AC
BA	→	AB
CB	→	BC
aA	→	aa
aB	→	ab
bB	→	bb
bC	→	bc
cC	→	cc

Now to derive the string aabbcc we will start from start symbol :

S	rule S →	SABC
SABC	rule S →	aBC
aBCABC	rule CA →	AC
aBACBC	rule CB →	BC
aBABCC	rule BA →	AB
aABBCC	rule aA →	aa
aaBBCC	rule aB →	ab
aabBCC	rule bB →	bb
aabbCC	rule bC →	bc
aabbcC	rule cC →	cc
aabbcc		

Note : The language $a^n\ b^n\ c^n$ where $n \geq 1$ is represented by context sensitive grammar but it can not be represented by context free grammar.

Every context sensitive language can be represented by LBA.

8.4 LR (k) GRAMMARS

Before going to the topic of LR (k) grammar, let us discuss about some concepts which will be helpful understanding it.

In the unit of context free grammars you have seen that to check whether a particular string is accepted by a particular grammar or not we try to derive that sentence using rightmost derivation or leftmost derivation. If that string is derived we say that it is a valid string.

Example :

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to id \mid (E)$$

Suppose we want to check validity of a string id + id * id . Its rightmost derivation is

$$\begin{aligned} E &\Rightarrow E + T \\ &\Rightarrow E + T * F \\ &\Rightarrow E + T * id \\ &\Rightarrow E + F * id \\ &\Rightarrow E + id * id \\ &\Rightarrow T + id * id \\ &\Rightarrow F + id * id \\ &\Rightarrow id + id * id \end{aligned}$$

FIGURE(a) : Rightmost Derivation of id + id * id

Since this sentence is derivable using the given grammar. It is a valid string. Here we have checked the validity of string using process known as derivation.

The validity of a sentence can be checked using reverse process known as reduction. In this method for a given x, inorder to know whether it is valid sentence of a grammar or not, we start with x and replace a substring x_1 with variable A if $A \to X_1$ is a production. We repeat this process until we get starting state.

Consider the grammar,

$$E \to E + T \mid T$$
$$E \to T * F \mid F$$
$$F \to (E) \mid id$$

Let us check the validity of string id + id * id.

F + id * id Replaced F with id since F $\to$ id is a production
T + id * id Replaced F with T using production T $\to$ F
E + id * id Replaced T with E using production E $\to$ T
E + F * id Replaced id with F using production F $\to$ id
E + T * id Replaced F with using production T $\to$ F
E + T * F Replaced id with F using production F $\to$ id
E + T Replaced T * F with T using production T $\to$ T * F
E Replaced E + T with E using production E $\to$ E + T

FIGURE(b) : Reduction of id + id * id

Here since we are able to reduce to starting state E, so that id + id * id is accepted by the given grammar.

Note : There may be different ways of selecting as substring in sentential form. In our reduction we have used reverse of rightmost derivation shown in Figure(a).

The substring in right sentential form which causes reduction to starting state is known as handle and corresponding production is known as handle production. For example, in right sentential form E + T * id of Figure(b) we can either replace substring T with F using T $\to$ F or replace id with F using F $\to$ id. If we use the first reduction, the sentential form will become E + F * id. This will not lead to starting state. Hence here F is not handle. Where as if we reduce, the sentential form will be E + T * F which can be reduced to starting state using subsequent reductions. Hence here F is a handle and F $\to$ id is handle production.

In reduction process we have seen that we repeat the process of substitution until we get starting state. But some times several choices may be available for replacement. In this case we have to backtrack and try some other substring. For certain grammars it is possible to carry out the process in deterministic. (i. e., having only one choice at each time). LR grammars form one such subclass of context free grammars. Depending on the number of look ahead symbolized to determine whether a substring must be replaced by a non terminal or not, they are classified as LR(0), LR(1).... and in general LR(k) grammars.

LR(k) stands for left to right scanning of input string using rightmost derivation in reverse order (we say reverse order because we use reduction which is reverse of derivation) using look ahead of k symbols.

8.4.1 LR(0) Grammar

LR(0) stands for left to right scanning of input string using rightmost derivation in reverse order using 0 look ahead symbols.

Before defining LR(0) grammars, let us know about few terms.

Prefix Property : A language L is said to have prefix property if whenever w in L, no proper prefix of w is in L. By introducing marker symbol we can convert any DCFL to DCFL with prefix property. Hence $L\$ = \{ w\$ \mid w \in L \}$ is a DCFL with prefix property whenever w is in L.

Example : Consider a language L = { cat, cart, bat, art, car } . Here, we can see that sentence cart is in L and its one of the prefixes car is also is in L. Hence, it is not satisfying property. But L$ = { cat $, cart $, bat $, art $, car $ }

Here, cart $ is in L$ but its prefix cart or car are not present in L$. Similarly no proper prefix is present in L$. Hence, it is satisfying prefix property.

Note : LR(0) grammar generates DCFL and every DCFL with prefix property has a LR(0) grammar.

LR Items

An item for a CFG is a production with dot any where in right side including beginning or end. In case of $\in$ production, suppose $A \rightarrow \in, A \rightarrow .$ is an item.

Example :

Consider the grammar,

$S' \rightarrow S$

$S \rightarrow cAd$

$A \rightarrow a \mid \in$

The items for this grammar are ,

$S' \rightarrow .S$

$S' \rightarrow S.$

$S \rightarrow .cAd$

$S \rightarrow c.Ad$

$S \rightarrow cA.d$

$S \rightarrow cAd.$

$A \rightarrow .a$

$A \rightarrow a.$

$A \rightarrow .$

An item indicates how much of a production we have seen at a given point in parsing process.

Valid Item : We say in item $A \rightarrow \alpha . \beta$ is valid for a viable prefix (i. e., most possible prefix) γ there is a rightmost derivation $S \underset{rm}{\overset{*}{\Rightarrow}} \delta Aw \underset{rm}{\Rightarrow} \delta\ \alpha\ \beta\, w$ and $\delta\alpha = \gamma$.

Example :

$S \rightarrow cAt$

$A \rightarrow ar$

The sentence cart belongs to this grammar,

$S \overset{*}{\Rightarrow} CAt \Rightarrow cart$

The possible or viable prefixes for cart are { c, ca, car, cart } for the prefix ca $A \rightarrow a.r.$ is valid item and for viable prefix car $A \Rightarrow ar$ is valid item.

Computing Valid Item Sets

The main idea here is to construct from a given grammar a deterministic finite automata to recognize viable prefixes. We group items together into sets which give to states of DFA. The items may be viewed as states of NFA and grouped items may be viewed as states of DFA obtained using subset construction algorithm.

To compute valid set of items we use two operations goto and closure.

Closure Operation

It I is a set of items for a grammar G, then closure (I) is the set of items constructed from I by two rules.

1. Initially, every item I is added to closure (I).
2. If $A \rightarrow \alpha.B\beta$ is in closure (I) and $B \rightarrow \delta$ is production then add item $B \rightarrow \delta$ to I, if it is not already there. We apply this rule until no more new items can be added to closure (I).

Example : For the grammar,

$$S' \rightarrow S$$
$$S \rightarrow cAd$$
$$A \rightarrow a$$

If $S' \rightarrow S$ is set of one item in state I then closure of I is,

$$I_1 : S' \rightarrow .s$$
$$S \rightarrow .cAD$$

The first item is added using rule 1 and $S \rightarrow .cAd$ is added using rule 2. Because '.' is followed by nonterminal S we add items having S in LHS. In $S \rightarrow .cAd$ '.' is followed by terminal so no new item is added.

Goto Function : It is written as goto (I, X) where I is set of items and X is grammar symbol.

If $A \rightarrow \alpha.X\beta$ is in some item set I then goto (I, X) will be closure of set of all item $A \rightarrow \alpha.X.\beta$.

For example,

goto (I_1, c)

closure $(S \to c.Ad)$

i. e., $S \to c.Ad$

$A \to a$

now let us see how all the valid sets of items are computed for the given grammar in example 1.

Initially I_0 will be the starting state. It contains only the item $S' \to .S$ we find its closure to find set of items in this state for each state I_1 and symbol β after '.' we apply goto (I_i, β), goto (I_0, S) and find its closure. This constitutes next state I_j. We continue this process goto (I_0, a) until no new states are obtained.

$I_0 : S' \to .S$

$S \to .Ad$

$I_1 : S' \to S.$

$I_2 : S \to c.Ad$

$A \to .a$

goto (I_2, A)

$I_3 : S \to cA.d$

goto (I_2, a)

$I_4 : A \to a.$

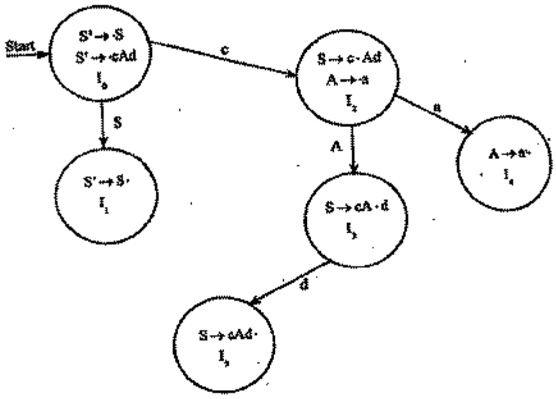
goto (I_3, d)

$I_5 : S \to cAd.$

This process is stopped because all possible complete items are obtained. A complete item is the one which has dot in rightmost position.

Ecah item set corresponds to a state of DFA. Hence, the DFA for given grammar will have six states corresponding to I_0 to I_5.

DFA :



FIGURE(a) : DFA whose States are the Sets of Valid Items

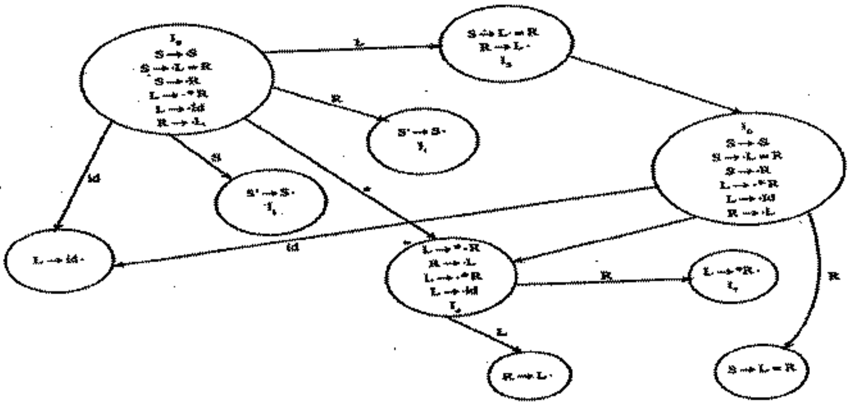
Definition of LR(0) Grammar : We say G is an LR (0) grammar if,

1. Its start symbol does not appear on the right hand side of any production and
2. For every viable prefix γ of G, whenever $A \to \alpha$ is a complete item valid for γ, then no other complete item nor any item with terminal to the right of the dot is valid for γ.

Condition 1 : For a grammar to be LR(0) it should satisfy both the conditions. The first condition can be made to satisfy by all grammars by introduction of a new production $S' \to S$ is known augmented grammar.

Condition 2 : For the DFA shown in Figure(a), the second condition is also satisfied because in the item sets I_1, I_4 and I_5 each containing a complete item, there are no other complete items nor any other conflict.

Example : Consider the DFA given in figure(b).



FIGURE(b) : DFA for the given Grammar

DFA for grammar,

$$S \to L = R$$
$$S \to R$$
$$L \to *R$$
$$L \to id$$
$$R \to L$$

i. The first condition of LR(0) grammar is satisfied.

ii. Consider state I_2 and viable prefixes of L = R { L, L = and L = R } for prefix LR → L. is a complete item and there is another item having the prefix L i. e., S → L . = R followed by terminal. Hence, violating second rule. So it is not LR(0) grammar.

8.5 DECIDABILITY OF PROBLEMS

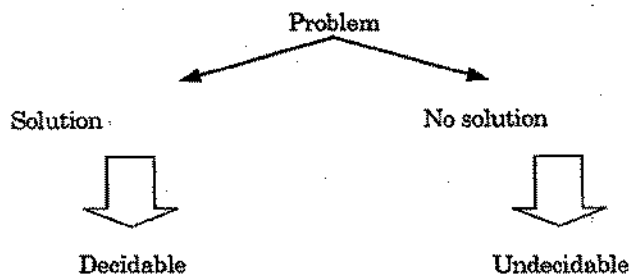
In our general life, we have several problems and some of these have solution also, but some have not. Simply, we say a problem is decidable if there is a solution otherwise undecidable.

Example : consider following problems and their possible answers.

1. Does the sun rise in the east ? (YES)
2. Does the earth move around the sun ? (YES)
3. What is your name ? (FLAT)
4. Will tomorrow be a rainy day ? (No answer)

We have solutions (answers) for all problems except the last. We can not answer the last problem, because we have no way to tell about the weather of tomorrow, but to some extent we can only predict. So, the last problem is undecidable and remaining problems are decidable.

So, if a problem can be solved or answered based on some algorithm then it is decidable otherwise undecidable.



Each problem P is a pair consisting of a set and a question, where the question can be applied to each element in the set. The set is called the domain of the problem, and its elements are called the instances of the problem.

Example :

Domain = { All regular languages over some alphabet Σ },
Instance : L = { w : w is a word over Σ ending in abb},
Question : Is union of two regular languages regular ?

8.5.1 Decidable and Undecidable Problems

A problem is said to be decidable if
1. Its language is recursive, or
2. It has solution

Other problems which do not satisfy the above are undecidable. We restrict the answer of decidable problems to " YES" or "NO" . If there is some algorithm exists for the problem, then outcome of the algorithm is either "YES" or "NO" but not both. Restricting the answers to only "YES" or "NO" we may not be able to cover the whole problems, still we can cover a lot of problems. One question here. Why we are restricting our answers to only "YES" or "NO"? The answer is very simple ; we want the answers as simple as possible.

Now, we say " If for a problem, there exists an algorithm which tells that the answer is either "YES" or "NO" then problem is decidable."

If for a problem both the answers are possible ; some times "YES" and sometimes "NO", then problem is undecidable.

8.5.2 Decidable Problems for FA, Regular Grammars and Regular Languages

Some decidable problems are mentioned below :
1. Does FA accept regular language ?
2. Is the power of NFA and DFA same ?
3. L_1 and L_2 are two regular languages. Are these closed under following :
   (a) Union
   (b) Concatenation
   (c) Intersection
   (d) Complement

(e) Transpose
(f) Kleene Closure (positive transitive closure)

4. For a given FA M and string w over alphabet Σ, is $w \in L(M)$? This is decidable problem.
5. For a given FM, is $L(M) = \phi$? This is a decidable problem.
6. For a given FA M and alphabet Σ, is $L(M) = \Sigma^*$? This is a decidable problem.
7. For given two FA M_1 and M_2, $L(M_1), L(M_2) \in \Sigma^*$, is $L(M_1) = L(M_2)$? This is a decidable problem.
8. For given two regular languages L_1 and L_2 over some alphabet Σ, is $L_1 \subset L_2$? This is a decidable problem.

8.5.3 Decidable And Undecidable Problems About CFLs, And CFGs

Decidable Problems

Some decidable problems about CFLs and CFGs are given below.

1. If L_1 and L_2 are two CFLs over some alphabet Σ, then $L_1 \cup L_2$ is CFL.
2. If L_1 and L_2 are two CFLs over some alphabet Σ, then $L_1 L_2$ is CFL.
3. If L is a CFL over some alphabet Σ, then L* is a CFL.
4. If L_1 is a regular language, L_2 is a CFL then $L_1 \cup L_2$ is CFL.
5. If L_1 is a regular language, L_2 is a CFL over some alphabet Σ, then $L_1 \cap L_2$ is CFL.
6. For a given CFG G, is $L(G) = \phi$ or not?
7. For a given CFG G, finding whether L(G) is finite or not, is decidable.
8. For a given CFG G and a string w over Σ, checking whether $w \in L(G)$ or not is decidable.

Undecidable Problems

Following are some undecidable problems about CFGs and CFLs:

1. For two given CFLs L_1 and L_2, whether $L_1 \cap L_2$ is CFL or not, is undecidable.
2. For a given CFL L over some alphabet Σ, whether complement of L i. e. E * - L is CFL or not, is undecidable.
3. For a given CFG G, is L(G) ambiguous? This is undecidable problem.
4. For two arbitrary CFGs G_1 and G_2, deciding $L(G_1) \cap L(G_2) = \phi$ or not, is undecidable.
5. For two arbitrary CFGs G_1 and G_2, deciding $L(G_1) \subseteq L(G_2)$ or not, is undecidable.

8.5.4 Decidability and Undecidability About TM

We have considered TM as a most powerful machine that can compute anything, which can recognize any language. So, from where undecidability comes and why ? These questions are really interesting. According to Church - Turing Thesis, we have considered TM as an algorithm and an algorithm as a TM . So, for a problem, if there is an algorithm (solution to find answer) then problem is decidable and TM can solve that problem. We have several problems related to computation and recognization that have no solution and these problems are undecidable.

Partial Decidable and Decidable Problems

A TM M is said to partially solve a given problem P if it provides the answer for each instance of the problem and the problem is said to be partially solvable. If all the computations of the TM are halting computations for P, then the problem P is said to solvable.

A TM is said to partially decide a problem if the following two conditions are satisfied.

(a) The problem is a decision problem, and

(b) The TM accepts a given input if and only if the problem has an answer "YES" for the input, that is the TM accepts the language L = { x : x is an instance of the problem, and the problem has the answer "YES" for x }.

A TM is said to decide a problem if it partially decides the problem and all its computations are halting computations.

The main difference between a TM M_1 that partially solves (partially decides) a problem and a TM M_2 that solves (decides) the same problem is that M_1 might reject an input by a non - halting computation, whereas M_2 can reject the input only by a halting computation.

A problem is said to be unsolvable if no algorithm can solve it, and a problem is said to be undecidable if it is a decision problem and no algorithm can decide it.

Decidable Problems about Recursive and Recursive Enumerable Languages

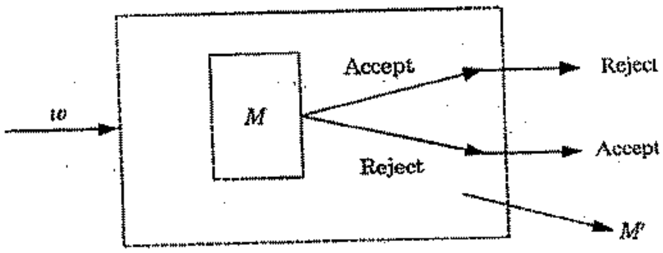
As we have discussed earlier that if a problem has a solution then it is decidable. In this section, we will discuss some decidable problems about recursive and recursive enumerable languages.

1. The complement of a recursive language L over some alphabet Σ is recursive.

Proof : We will discuss a constructive algorithm to prove that complement of a recursive language is also recursive i. e. recursive languages are closed under complementation.

As we know that for all strings $w \in L$, a TM always halts and rejects those strings that are not in L. So, " for all strings $w \in L$ " is always decidable.

We construct a TM M, which recognizes the language L. We construct another TM M' based on M such that M' accepts those strings which are rejected by M. It means, if M accepts then M' does not. M' rejects those strings that are accepted by M. It means, all strings $x \notin L$ are accepted by M' and for all strings $w \in L$ are rejected. So, M' also follows same kind of algorithm to decide whether a string $w \in L$ or not. Hence, complement of recursive language L i. e. $\Sigma^* - L$ is also recursive. The logic diagram of M' is shown in Figure(a).

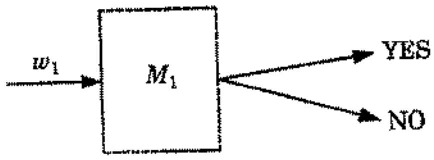


Figure(a)

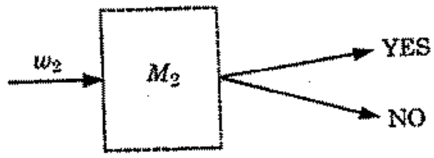
In general, recursive languages are closed under complement operation.

2. The union of two recursive languages is recursive.

Proof : Let L_1 and L_2 be two recursive languages and Turing machines M_1 and M_2 recognize L_1 and L_2 respectively shown in Figure(b) and Figure(c).

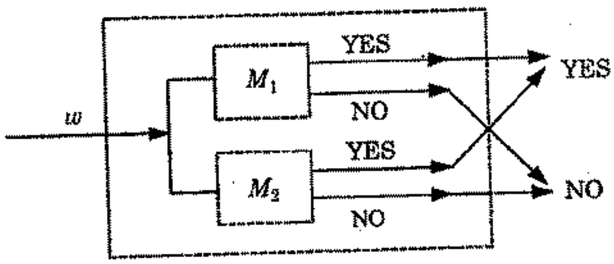


Figure(b)



Figure(c)

We construct a third TM M_3, which follows either M_1 or M_2 as shown in figure(d).



Figure(d)

TM M_3 accepts if either M_1 accepts or M_2 accepts and rejects if either M_1 rejects or M_2 rejects. Since, M_1 and M_2 are based on algorithms, so M_3 is also based on the same kind of algorithm. Therefore, union of two recursive languages L_1 and L_2 is also recursive. In general, recursive languages are closed under union operation.

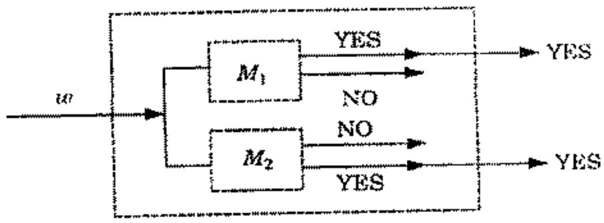
3. A language is recursive if and only if its complement is recursive.

4. The union of two recursively enumerable languages is recursive enumerable.

Proof : Let L_1 and L_2 be two recursively enumerable languages and recognized by M_1 and M_2 Turing machines. We construct another TM M_3, which accepts either L_1 or L_2. Now, as we know the problem about recursive enumerable languages that if w is not in L_1 and L_2, then M_3 can not decide. So, the problem of recursive languages is persistent with M_3 also. So, $N(M_3)$ is recursive enumerable language and hence $L_1 \cup L_2$ is recursive enumerable languages. In general, recursive enumerable languages are closed under union operation.

5. If a language L over some alphabet Σ and its complement $\overline{L} = \Sigma^* - L$ is recursive enumerable, then L and $\overline{L}$ are recursive languages.

Proof : We construct two Turing machines M_1 for L and M_2 for $\overline{L}$. Now, we construct a third TM M_3 based on M_1 and M_2 as shown in figure(e). TM M_3 accepts w if TM M_1 accepts and rejects w if M_2 accepts. It means, if $w \in L$, then w is accepted and if $w \notin L$ then it is rejected. Since, for all w, either w is accepted or rejected. Hence, M_3 is based on algorithm and produces either "YES" or "NO" for input string w, but not both. It means, M_3 decides all the strings over Σ. Hence, L is recursive. As we know that complement of a recursive language is also recursive and hence $\overline{L}$ is also recursive.



Figure(e)

6. We have following co - theorem based on above discussion for recursive enumerable and recursive languages.

Let L and $\overline{L}$ are two languages, where $\overline{L}$ the complement of L, then one of the following is true :

(a) Both L and $\overline{L}$ are recursive languages,

(b) Neither L nor $\overline{L}$ is recursive languages,

(c) If L is recursive enumerable but not recursive, then $\overline{L}$ is not recursive enumerable and vice versa.

Undecidable Problems about Turing Machines

In this section, we will first discuss about halting problem in general and then about TM.

Halting Problem (HP)

The **halting problem** is a decision problem which is informally stated as follows :

"Given a description of an algorithm and a description of its initial arguments, determine whether the algorithm, when executed with these arguments, ever halts. The alternative is that a given algorithm runs forever without halting."

Alan Turing proved in 1936 that there is no general method or algorithm which can solve the halting problem for all possible inputs. An algorithm may contain loops which may be infinite or finite in length depending on the input and behaviour of the algorithm. The amount of work done in an algorithm usually depends on the input size. Algorithms may consist of various number of loops, nested or in sequence. The HP asks the question :

Given a program and an input to the program, determine if the program will eventually stop when it is given that input ?

One thing we can do here to find the solution of HP. Let the program run with the given input and if the program stops and we conclude that problem is solved. But, if the program doesn't stop in a reasonable amount of time, we can not conclude that it won't stop. The question is : " how long we can wait .... ?" . The waiting time may be long enough to exhaust whole life. So, we can not take it as easier as it seems to be. We want specific answer, either "YES" or "NO", and hence some algorithm to decide the answer.