# 7 Turing Machines

**Objectives :**

- Introduction
- Turing Machine Model
- Definition of TM
- Design of TM
- Power of TM
- Universal TM
- Composite and iterated TM
- Church's hypothesis.
- Properties of recursive and recursively enumerable langauges
- Post's correspondence problem

## 7.1 Introduction

**Alan Turing** is father of such a model which has computing capability of general purpose computer. Hence this model is popularly known as **Turing machine**. This machine has following **features** -

1. It has external memory which remembers arbitrarily long sequence of input.
2. It has unlimited memory capability.
3. The model has a facility by which the input at left or right on the tape can be read easily.
4. The machine can produce certain output based on its input. Sometimes it may be required that the same input has to be used to generate the output. So in this machine the distinction between input and output has been removed. Thus a common set of alphabets can be used for the Turing machine.

## 7.2 Turing Machine Model

The Turing machine can be modelled with the help of following representation.

1) The input tape having infinite number of cells, each cell containing one input symbol, and thus the input string can be placed on a tape. The empty tape is filled by blank characters.

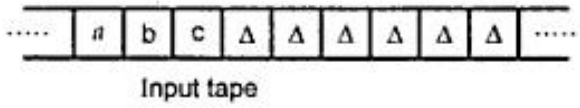


**Fig. 7.1 Input tape**

2) The finite control and the tape head which is responsible for reading the current input symbol. The tape head can move to left or right.

3) A finite set of states through which machine has to undergo.

4) Finite set of symbols called external symbols which are used in building the logic of turing machine.

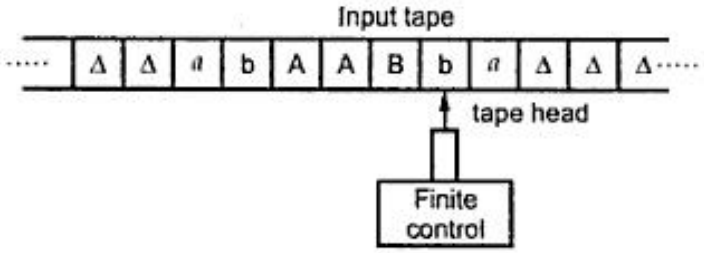


**Fig. 7.2 Turing machine**

## 7.3 Definition of Turing Machine (TM)

The Turing machine is a collection of following components.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \Delta \text{ or } B, F)$$

1) Q is a finite set of states.

2) $\Gamma$ is finite set of external symbols.

3) $\Sigma$ is a finite set of input symbols.

4) $\Delta$ or b or $B \in \Gamma$ is a blank symbol majorly used as end marker for input.

5) $\delta$ is a transition or a mapping function.

The mapping function shows the mapping from states of finite automata and input symbol on the tape to the next states, external symbols and the direction for moving the tape head. This is known as a **'triple'** or a **'program'** for Turing machine.

For example -

$$(q_0, a) \rightarrow (q_1, A, L)$$

This means that if currently we are reading the input symbol '*a*' and we are in $q_0$ state then we can go to $q_1$ state by replacing or printing '*a*' by A and now move ahead to left.

6) $q_0$ be the initial state where $q_0 \in Q$.

7) F is a set of final states. The final state is such a state where turing machine halts.

Thus turing machine is such machine which is used compute a large class of language.

## 7.4 Design of Turing Machines

Construction of turing machine is a process of writing out the complete set of states and next move function. This is a totally conceptual phenomenon. The turing machine can be designed with the help of some conceptual tools. Let us discuss some of these tools.

### 7.4.1 Storage in Finite Control

As shown in Fig. 7.2, the model of turing machine has a finite control. This finite control can be used to hold some amount of information. The finite automata stores the information in pair of elements such as the current state and the current symbol pointed by the tape head. This is just a conceptual arrangement.

For example - The function δ can be written as follows.

$$\delta([q_0, 0], 1) \rightarrow ([q_1, 1], \Delta, R)$$

This means that if finite control shows the initial state is $q_0$ and stores the current symbol 0 if it reads the symbol 1 then the machine goes to next state $q_1$ replace that 1 by Δ and moves to right. This helps in building the transition graph of the language.

**Example 7.1 :** *Construct a turing machine M for* $\Sigma = \{a, b\}$ *which will convert lower case letters to upper case.*

**Solution :**

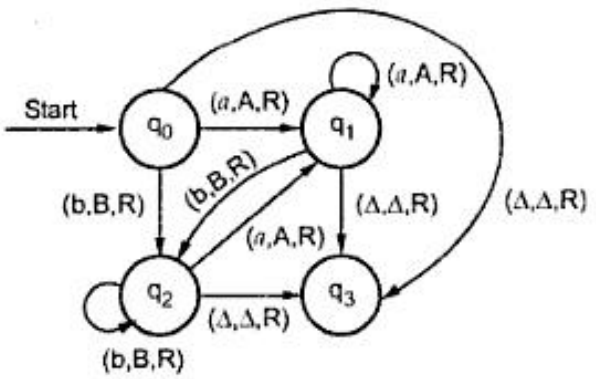


**Fig. 7.3**

The turing machine is designed such that we will get the equivalent upper case string to the input. The $q_0$ is a initial state and $q_3$ is final state. The $\delta$ function is written along each edge. For the sake of understanding consider the transition between $q_0$ and $q_1$, if the finite control reads on the tape A will be printed there and the move will be in right direction. After converting the given string to upper case we reach to the final state $q_3$. The $q_3$ is a halt state.

### 7.4.2 Multiple Tracks

If the input tape is divided into multiple tracks then the input tape will be as follows -

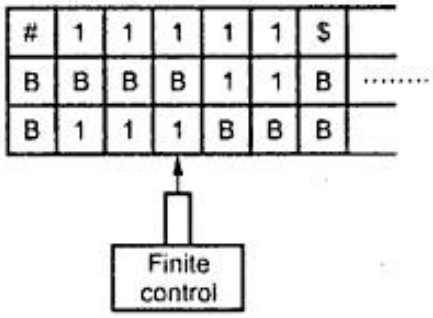


**Fig. 7.4 Multiple tracks**

**For example :**

As shown in Fig. 7.4 the input tape has multiple tracks on the first track. The input which is placed is surrounded by # and $. The unary number equivalent to 5 is placed on the input tape, on the first track. On the second track unary 2 is placed. If we construct a TM which subtracts 2 from 5 we get the answer on the third track and that is 3, in unary form. Thus this TM is for subtracting two unary numbers with the help of multiple tracks.

### 7.4.3 Checking of Symbols

Checking of symbols is an effective way of recognizing the language by TM. The symbols are to be placed on the input tape. The symbol which is read is marked by any special character. The tape head can be moved to the right or left. Let us take some example and we will see how to build turing machine by checking of symbols.

**Example 7.2 :** *Construct a turing machine* $M=(Q,\Sigma,\Gamma,\delta,q_0,B,F)$ *which recognizes the language* $L=\{wcw|w\in(a+b)^+\}$.

**Solution :** In this language the input set is $\Sigma$ = {a, b}. The string which when will be placed on the input tape it will have two distinct parts separated by letter c, such as

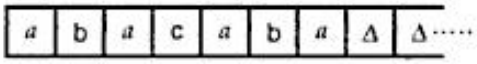


**Fig. 7.5**

In the checking off symbols, each symbol is marked by special character. The simple logic in construction of this TM will be we mark the first letter and then move to the right till we not get c, the first letter after c will be compared with the marked letter. If it is the same as which we have marked then mark this symbol otherwise goto reject state. It can be shown as below,

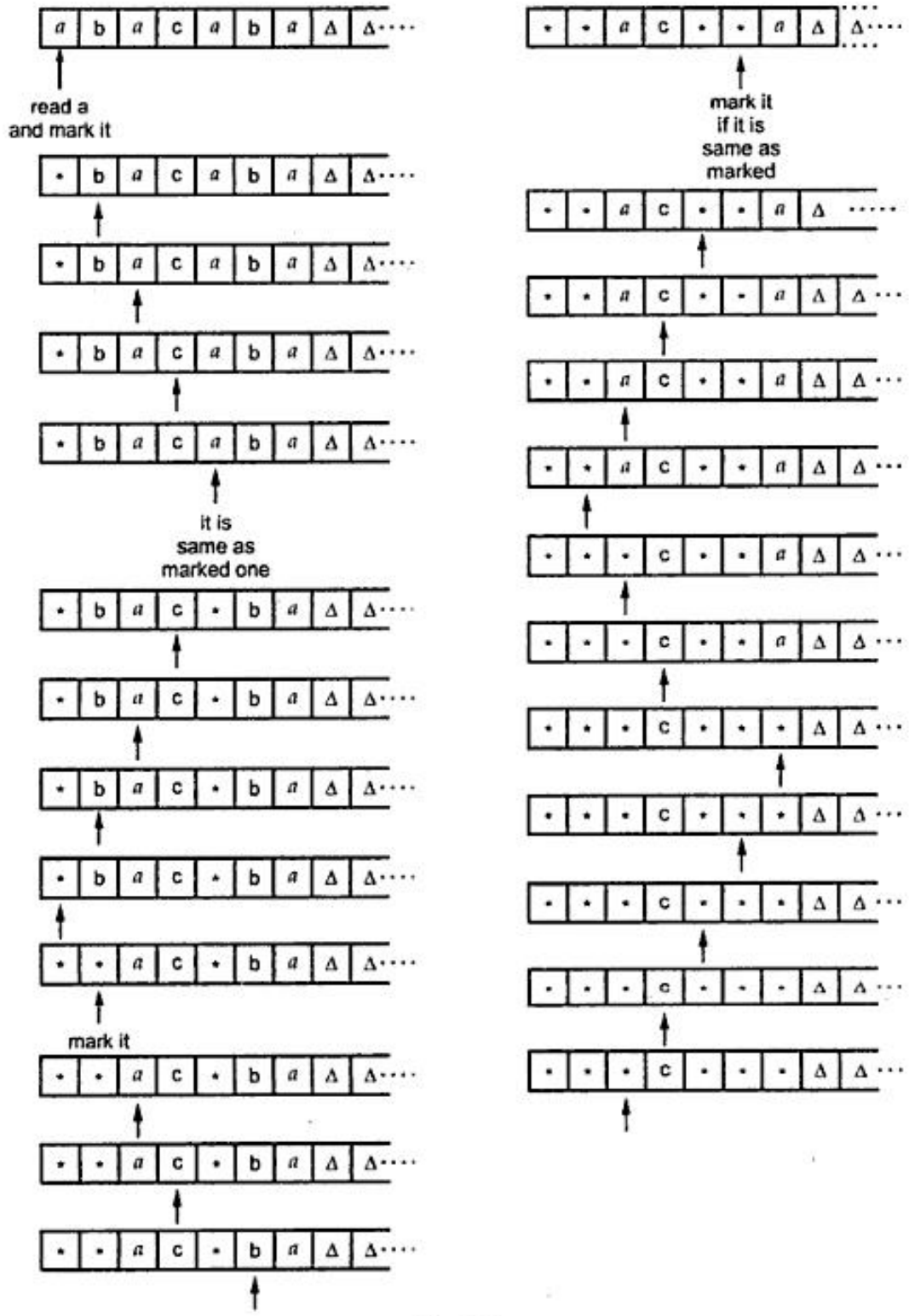


Fig. 7.6

Now machine goes to accept state. Thus in this TM we are scanning each symbol and trying to recognize the string.

### 7.4.4 Subroutine

In the high level languages use of subroutines built the modularity in the program development process. The same type of concept can be introduced in construction of TM. We can write the subroutines as a turing machine. Let us see how it works with the help of some example.

➠ **Example 7.3 :** *Construct a TM for the subroutine*
$f(a, b) = a * b$
*where a and b are unary numbers.*

**Solution :** The unary number are represented by 1's. That if $a = 2$ and b = 3 then

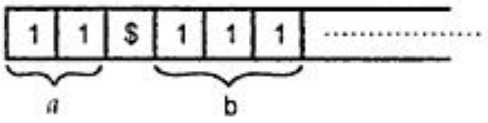


**Fig. 7.7**

For this subroutine we will perform a copy operation. That is marking first 1 of $a$ and copying it after Δ at b number of times, similarly marking second 1 of $a$ and copying it at the rightmost end (i.e. after previously copied 1's) for b number of times. This makes the multiplication function complete

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1 ↑ | 1 | $ | 1 | 1 | 1 | Δ | |
| | | X | 1 ↑ | $ | 1 | 1 | 1 | Δ | Moving to right upto $ |
| | | X | 1 | $ | 1 ↑ | 1 | 1 | Δ | Copy this 1 after Δ by marking it as Y. |
| | X | 1 | $ | Y | 1 | 1 | Δ | 1 ↑ | Now move to left upto Y |
| | X | 1 | $ | Y ↑ | 1 | 1 | Δ | 1 | Δ |
| | X | 1 | $ | Y | 1 ↑ | 1 | Δ | 1 | Δ Mark this 1 as Y and copy this 1 at the rightmost end. |
| X | 1 | $ | Y | Y | 1 | Δ | 1 | 1 ↑ | Δ Now move to left upto Y |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | X | 1 | $ | Y | Y | 1 | Δ | 1 | 1 | Δ | |
| | | | | | ↑ | | | | | | |
| | X | 1 | $ | Y | Y | 1 | Δ | 1 | 1 | Δ | Mark this 1 as Y and copy this 1 at the rightmost end. |
| | | | | | | ↑ | | | | | |
| X | 1 | $ | Y | Y | Y | Δ | 1 | 1 | 1 | Δ | Now move to left upto Y |
| | X | 1 | $ | Y | Y | Y | Δ | 1 | 1 | 1 | |
| | | | | | | ↑ | | | | | |

The head is pointing to Y next to Y is Δ that means all the 1's are over we will convert these Y to 1's and repeat the same procedure for the 1 which is left to $.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | X | 1 | $ | 1 | 1 | 1 | Δ | 1 | 1 | 1 | |
| | X | X | $ | 1 | 1 | 1 | Δ | 1 | 1 | 1 | Convert this 1 to X and move to right upto after the $ symbol. |
| | | ↑ | | | | | | | | | |
| | X | X | $ | 1 | 1 | 1 | Δ | 1 | 1 | 1 | Mark it Y and copy this 1 to rightmost end |
| | | | | ↑ | | | | | | | |
| | X | X | $ | Y | 1 | 1 | Δ | 1 | 1 | 1 | Move to left upto Y |
| | | | | | | | | | | ↑ | |
| | X | X | $ | Y | 1 | 1 | Δ | 1 | 1 | 1 | Convert this 1 to Y and move to right, copy this 1 to rightmost end |
| | | | | | ↑ | | | | | | |
| X | X | $ | Y | Y | 1 | Δ | 1 | 1 | 1 | 1 | Move to left upto Y |
| | | | | | | | | | | ↑ | |

The above steps will be repeated and all the 1's between $ and Δ are copied to the rightmost end. This will be like this

X X $ Y Y Y Δ 1 1 1 1 1 1

Now we will move to left by converting all Y's and X's to 1's. Finally input tape will have

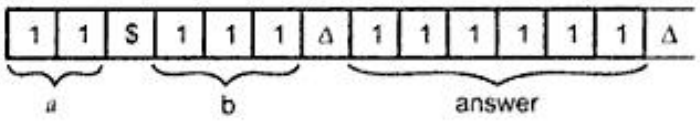


**Fig. 7.8**

We put these actions together and draw the transition graph and model out the turing machine.

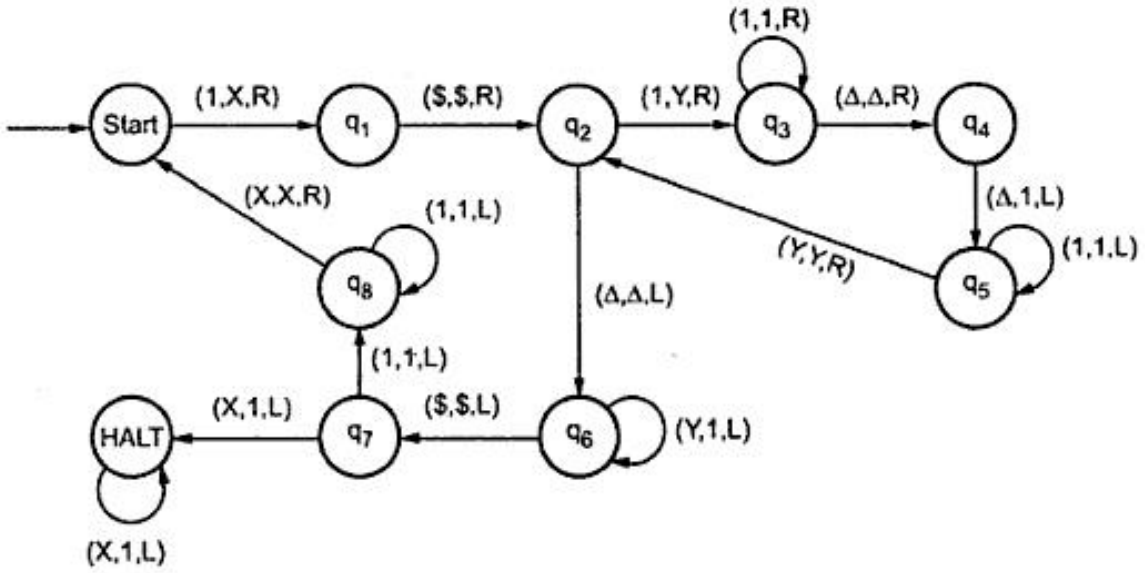


**Fig. 7.9 Turing machine for multiplication subroutine**

The HALT state is a final state in the above TM.

## 7.5 Computable Languages and Functions

The turing machine accepts all the languages even though there are recursively enumerable. Recursive means repeating the same set of rules for any number of times. And enumerable means a list of elements. The TM also accepts the computable functions, such as addition, multiplication, subtraction, division, power function, square function, logarithmic function and many more. We will solve some examples based on languages and function for construction of turing machine.

➠ **Example 7.4 :** *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

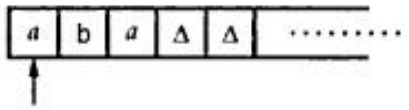


**Fig. 7.10**

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

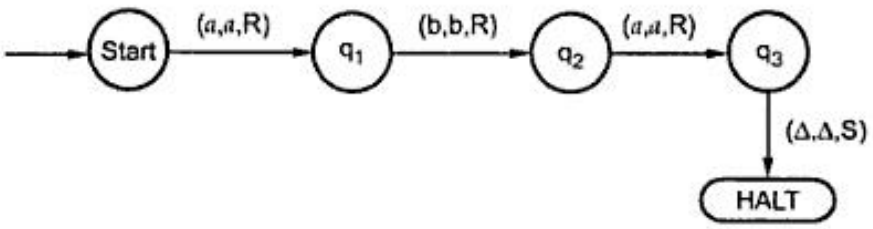


**Fig. 7.11 TM for 'aba'**

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

Consider the transition between start state and $q_1$ which is (*a*, *a*, R) that is the current symbol read from the tape is *a* then output it as *a* and then move the tape head to the right. The tape will look like this

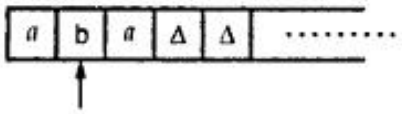


**Fig. 7.12**

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.

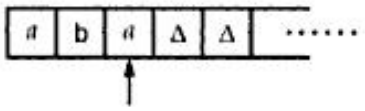


**Fig. 7.13**

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 (Δ,Δ, S). This means read Δ, print Δ and stay there or there is no move left or right. Eventhough we write (Δ, Δ, L) or (Δ, Δ, 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 *a*bb or *a*b or b*a*b ... 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 | Δ |
|---|---|---|---|
| Start | $(q_1, a, R)$ | – | – |
| $q_1$ | – | $(q_2, b, R)$ | – |
| $q_2$ | $(q_3, a, R)$ | – | – |
| $q_3$ | – | – | (HALT, Δ, 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 7.5 :** *Construct TM for language consisting of strings having any number of 0's and only even number of 1's over the input set* $\Sigma = \{0, 1\}$.

**Solution :** The number of zero's can be any ! but the even number of 1's are to be allowed. We have drawn FSM for this problem in chapter 2.

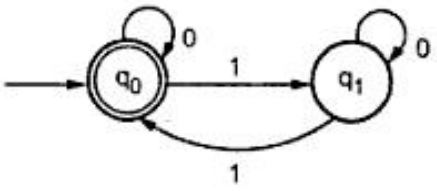


**Fig. 7.14 FSM for even number of 1's**

The same idea can be used to draw TM. We always assume that at the end of input Δ is placed.

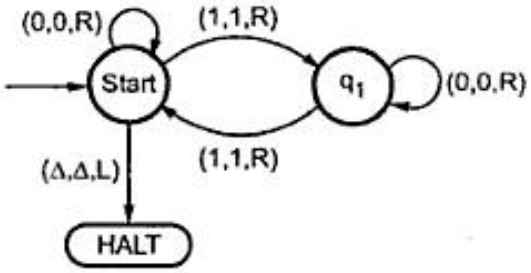


**Fig. 7.15**

Let us simulate the above TM for the input 1 1 0 1 0 1 which has even number of 1's. We assume that this input is placed on a input tape.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Input | 1 | 1 | 0 | 1 | 0 | 1 | Δ |
| head | ↑ | | | | | | |
| state | $q_1$ | | | | | | |
| | 1 | 1 | 0 | 1 | 0 | 1 | Δ |
| | | ↑ | | | | | |
| | | Start | | | | | |
| | 1 | 1 | 0 | 1 | 0 | 1 | Δ |
| | | | ↑ | | | | |
| | | | Start | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | Δ |
| | | | ↑ | | | |
| | | | $q_1$ | | | |
| 1 | 1 | 0 | 1 | 0 | 1 | Δ |
| | | | | ↑ | | |
| | | | | $q_1$ | | |
| 1 | 1 | 0 | 1 | 0 | 1 | Δ |
| | | | | | ↑ | |
| | | | | | Start | |
| 1 | 1 | 0 | 1 | 0 | 1 | Δ |
| | | | | | | ↑ |
| | | | | | | HALT |

Thus this input is accepted by TM.

**Example 7.6 :** *Design a turing machine which recognizes the input language having a substring as 101 and replaces every occurrence of 101 by 110.*

**Solution :** Replacement of any symbol by some another one means after reading of that specific symbol we should print the replacement symbol. In this case 101 has to be replaced by 110. The TM has to be constructed considering 101 as a substring and leaving 110 substring after complete scan of the input.

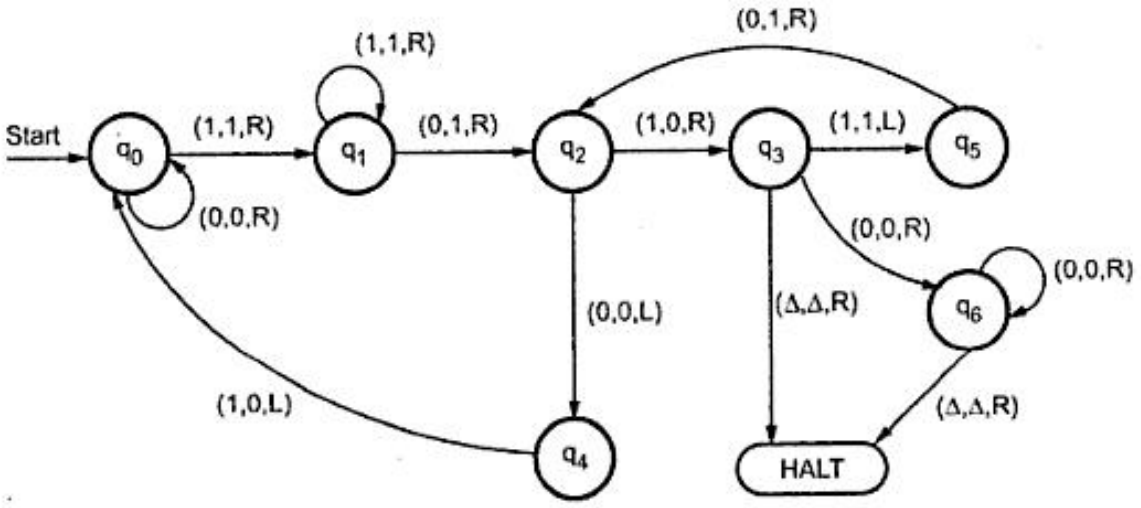


**Fig. 7.16**

Let us simulate it for the input.

| | |
|---|---|
| 1 0 0 1 0 1 0 Δ<br>↑ | read 1, print 1, goto $q_1$ |
| 1 0 0 1 0 1 0 Δ<br>↑ | goto R and next state is $q_0$ |
| 1 0 0 1 0 1 0 Δ<br>↑ | goto R, next state is $q_0$ |
| 1 0 0 1 0 1 0 Δ<br>↑ | goto R, next state $q_1$ |
| 1 0 0 1 1 1 0 Δ<br>↑ | print 1, goto R and next state is $q_2$ |
| 1 0 0 1 1 1 0 Δ<br>↑ | print 0, goto R and next state is $q_3$ |
| 1 0 0 1 1 0 0 Δ<br>↑ | print 0, goto R and next state is $q_6$ |
| 1 0 0 1 1 0 0 Δ<br>↑ | since we read current symbol as Δ we goto accept state i.e. HALT. |

**Example 7.7 :** *Construct a TM for the language of even number of 1's and even number of 0's over* $\Sigma=\{0, 1\}$.

**Solution :** For this type of problem even we have drawn FSM.

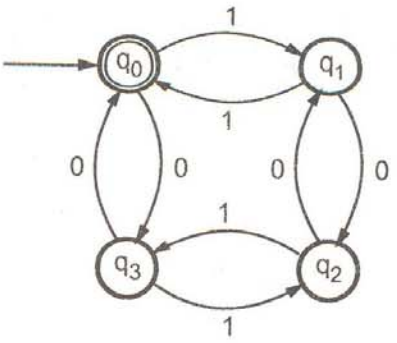


**Fig. 7.17**

And now it will be very much easy for us to convert this FSM to TM.

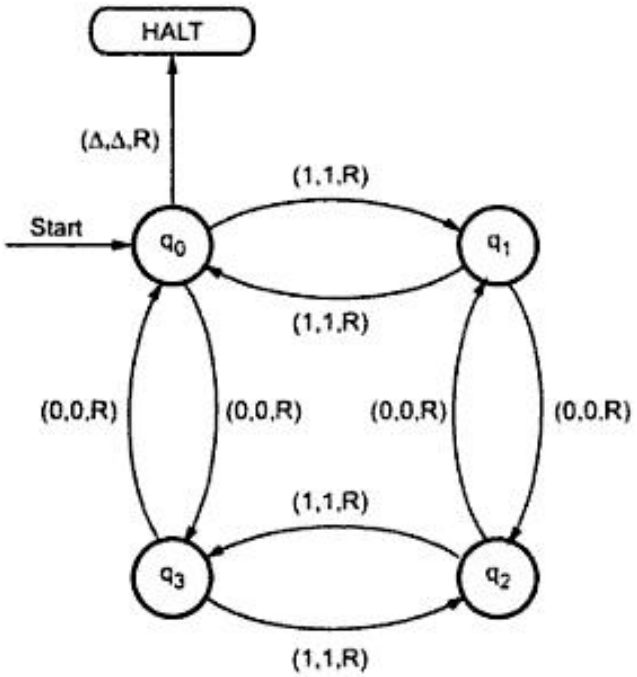


**Fig. 7.18**

Note that for any FSM we can construct turing machine. Thus turing machine more powerful than TM. In the next example we will see that the problem which is not solvable by FSM and solvable by PDA can be solved by TM as well.

**Example 7.8 :** *Construct TM for the language* $L=\{a^n b^n\}$ *where* $n \geq 1$.

**Solution :** We have already solved this problem by PDA. In PDA we have stack to remember the previous symbol. The main advantage of TM is we have a tape head which can be moved forward or backward, and the input symbol can be scanned. The simple logic which we will apply is read out each *a* mark it by A and then move ahead along the input tape and find out the b convert it to B. Repeat this process for all *a*'s and b's. If there is some kind of unequality in number of *a*'s and b's the TM will not end up in HALT state. Let us formulize the logic for $a^n$ $b^n$ and then construct it,

| | |
|---|---|
| *a a* b b Δ<br>↑ | Convert it A and move right in search of b. |
| A *a* b b Δ<br>↑ | Skip it, move ahead |
| A *a* b b Δ<br>↑ | Convert it to B and move left till A |
| A *a* B b Δ<br>↑ | Move left |
| A *a* B b Δ<br>↑ | Move right |

| | |
|---|---|
| A *a* B b Δ<br>↑ | Convert *a* to A and move right in search of b |
| A A B b Δ<br>↑ | Move right |
| A A B b Δ<br>↑ | Convert b to B and move left |
| A A B B Δ<br>↑ | Move left |
| A A B B Δ<br>↑ | Now immediately before B is A that means all the *a*'s are marked by A. So we will move right to ensure that no b is present |
| A A B B Δ<br>↑ | Move right |
| A A B B Δ<br>↑ | Move right |
| A A B B Δ<br>↑ | HALT |

Thus equality between *a*'s and b's can be checked by moving tape head to and fro. Let us try to design a TM using above logic.

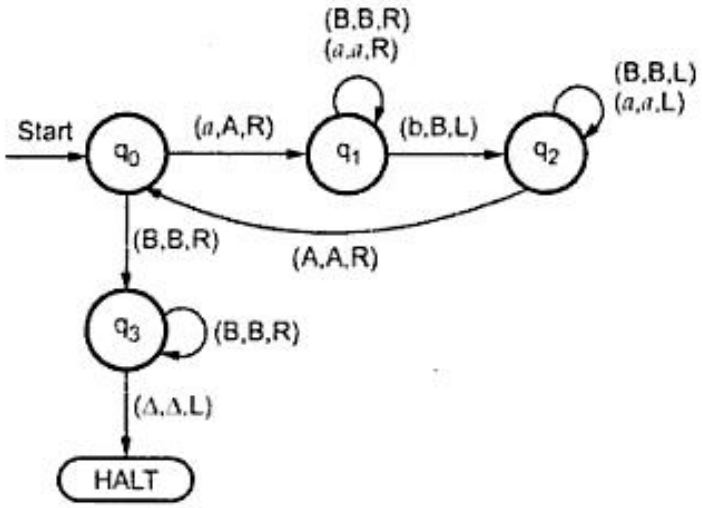


**Fig. 7.19**

Note that for the invalid inputs the machine does not go to HALT state.

**For example**

| | |
|---|---|
| $\underset{\uparrow}{a}\ a\ b\ b\ b\ \Delta$ | Convert to A, move right and next state = $q_1$ |
| $A\ \underset{\uparrow}{a}\ b\ b\ b\ \Delta$ | Move right, keeping $a$ as it is, state = $q_1$ |
| $A\ a\ \underset{\uparrow}{b}\ b\ b\ \Delta$ | Convert to B, move to left |
| $A\ \underset{\uparrow}{a}\ B\ b\ b\ \Delta$ | Move to left, state = $q_2$ |
| $\underset{\uparrow}{A}\ a\ B\ b\ b\ \Delta$ | Move to right and state = $q_0$ |
| $A\ \underset{\uparrow}{a}\ B\ b\ b\ \Delta$ | Convert to A, move to right now in state $q_1$ |
| $A\ A\ \underset{\uparrow}{B}\ b\ b\ \Delta$ | Move right keeping B as it is. |
| $A\ A\ B\ \underset{\uparrow}{b}\ b\ \Delta$ | Convert b to B and move left state is $q_2$ |
| $A\ A\ \underset{\uparrow}{B}\ B\ b\ \Delta$ | Go on moving left |
| $A\ \underset{\uparrow}{A}\ B\ B\ b\ \Delta$ | Now state $q_0$ move right |
| $A\ A\ \underset{\uparrow}{B}\ B\ b\ \Delta$ | Move right skipping all B's |
| $A\ A\ B\ B\ \underset{\uparrow}{b}\ \Delta$ | From state $q_3$ there is no path for input b and TM stops there |

Let us design the transition table for the same.

| | $a$ | b | A | B | $\Delta$ |
|---|---|---|---|---|---|
| $q_0$ | $(q_1, A, R)$ | – | – | $(q_3, B, R)$ | – |
| $q_1$ | $(q_1, a, R)$ | $(q_2, B, L)$ | – | $(q_1, B, R)$ | – |
| $q_2$ | $(q_2, a, L)$ | – | $(q_0, A, R)$ | $(q_2, B, L)$ | – |
| $q_3$ | – | – | – | $(q_3, B, R)$ | (HALT, $\Delta$, L) |
| HALT | – | – | – | – | – |

**Table 7.1 Transition table for Ex. 7.8**

Thus TM does not lead to HALT state for such a invalid input.

**Example 7.9 :** *Construct a TM for* $L = \{a^n b^n c^n \mid n \geq 1\}$.

**Solution :** This problem cannot be solved even by PDA. We have solved this type of problem in chapter 5 by two stack PDA (we have solved $a^n b^n a^n$) but turing

machine solves this type of problem even ! The same logic which we have used in previous example could be used now. Let us assume that the input is

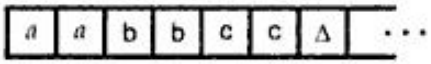


Fig. 7.20

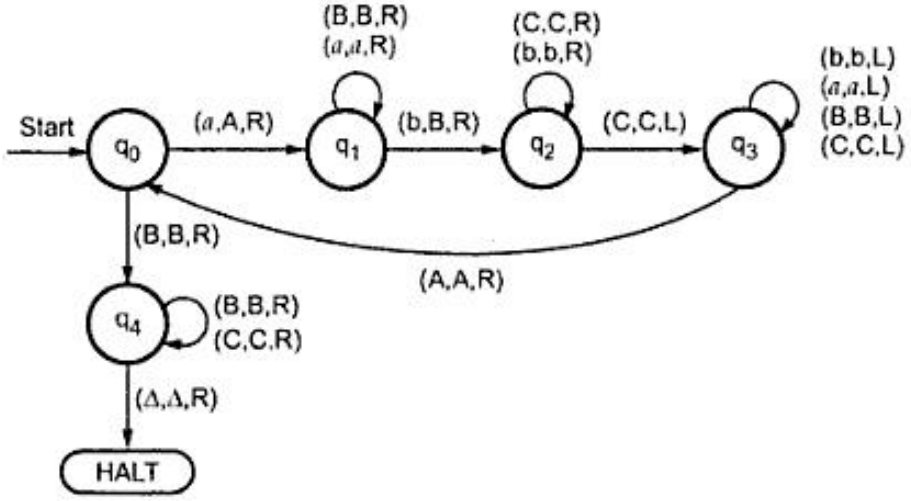


Fig. 7.21

The simulation for *a a* b b c c can be shown as below.

| | |
|---|---|
| *a a* b b c c Δ<br>↑ | Convert *a* to A, move right transition from $q_0$ to $q_1$ |
| A *a* b b c c Δ<br>↑ | Move right upto c |
| A *a* b b c c Δ<br>↑ | Convert b to B, move right transition from $q_1$ to $q_2$ |
| A *a* B b c c Δ<br>↑ | Move right upto c |
| A *a* B b c c Δ<br>↑ | Convert c to C, move right transition from $q_2$ to $q_3$ |
| A *a* B b C c Δ<br>↑ | Move left upto C |
| A *a* B b C c Δ<br>↑ | Move left till A, skipping *a*, b, B, C, see state $q_3$ then move one step right |
| A *a* B b C c Δ<br>↑ | Convert *a* to A, move right |

| | |
|---|---|
| A *a* B b C c Δ<br>↑ | Move right |
| A A B b C c Δ<br>↑ | Convert b to B and move right, transition from $q_1$ to $q_2$ |
| A A B B C c Δ<br>↑ | Move right |
| A A B B C c Δ<br>↑ | Convert c to C, move left till A, refer $q_3$ state |
| A A B B C C Δ<br>↑ | Move right, keep B as it is refer transition from $q_0$ to $q_4$ |
| A A B B C C Δ<br>↑ | Go on moving right by ignoring B, C refer $q_4$ state |
| A A B B C C Δ<br>↑ | Since Δ is read TM will reach to HALT state |
| A A B B C C Δ<br>↑ | |

Thus TM can very effectively recognizes $a^n\ b^n\ c^n$ and more powerful than PDA.

**Example 7.10 :** *Construct a TM machine for checking the palindrome of the string of even length.*

**Solution :** The same type problem we have solved for pushdown automata. Now we are going to construct TM for it. The logic is that we will read first symbol from left and then we compare it with the first symbol from right to check whether it is the same.

Again we compare second symbol from left with the second symbol from right. We repeat this process for all the symbols. If we found any symbol not matching, we cannot lead the machine to HALT state. The simulation of machine for a valid input such as

| | |
|---|---|
| *a* b *a* b b *a* b *a* Δ<br>↑ | We will mark it by * and move to right end in search of *a* |
| * b *a* b b *a* b *a* Δ<br>↑ | Move right |
| * b *a* b b *a* b *a* Δ<br>↑ | Moved right upto Δ |
| * b *a* b b *a* b *a* Δ<br>↑ | Moved left, checked if it is *a* |
| * b *a* b b *a* b Δ Δ<br>↑ | if it is *a*, replaced it by Δ |

| | |
|---|---|
| * b *a* b b *a* b Δ<br>↑ | Move to left, upto * |
| * b *a* b b *a* b Δ<br>↑ | Move right, read it |
| * b *a* b b *a* b Δ<br>↑ | Convert it to * and move right |
| * * *a* b b *a* b Δ<br>↑ | Move right upto Δ in search of b |
| * * *a* b b *a* b Δ<br>↑ | Move left, if left symbol is b convert it to Δ |
| * * *a* b b *a* Δ<br>↑ | Move left till * |
| * * *a* b b *a* Δ<br>↑ | Goto right |
| * * *a* b b *a* Δ<br>↑ | replace *a* by * and move right, upto Δ |
| * * * b b *a* Δ<br>↑ | We will move left and check if it is *a*, then replace it by Δ |
| * * * b b *a* Δ<br>↑ | It is *a* so replace by Δ |
| * * * b b Δ<br>↑ | Move to left till * |
| * * * b b Δ<br>↑ | Move right |
| * * * b b Δ<br>↑ | replace it by * |
| * * * * b Δ<br>↑ | Move right upto Δ |
| * * * * b Δ<br>↑ | Move left to see if it is b, if so then replace it by Δ |
| * * * * Δ<br>↑ | Move left till * |
| * * * * Δ<br>↑ | Move right and check whether it is Δ, if so |
| * * * * Δ<br>↑ | Goto HALT state |

us draw TM for it,

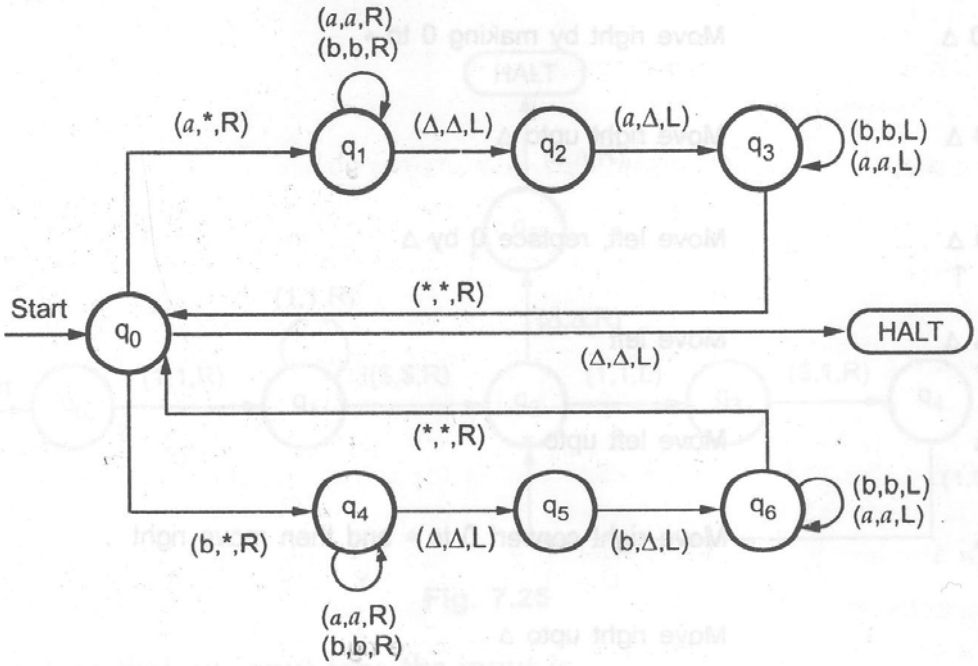


**Fig. 7.22**

ou observe the turing machine drawn for palindrome, you will find it very y constructed. Even there is no need to insert any special symbol in between to e out two halves.

**xample 7.11 :** *Construct a TM for checking the palindrome of a string of odd alindrome for* $\Sigma=\{0,1\}$.

**n :** The TM will be very much similar to previous one but with a slight difference.

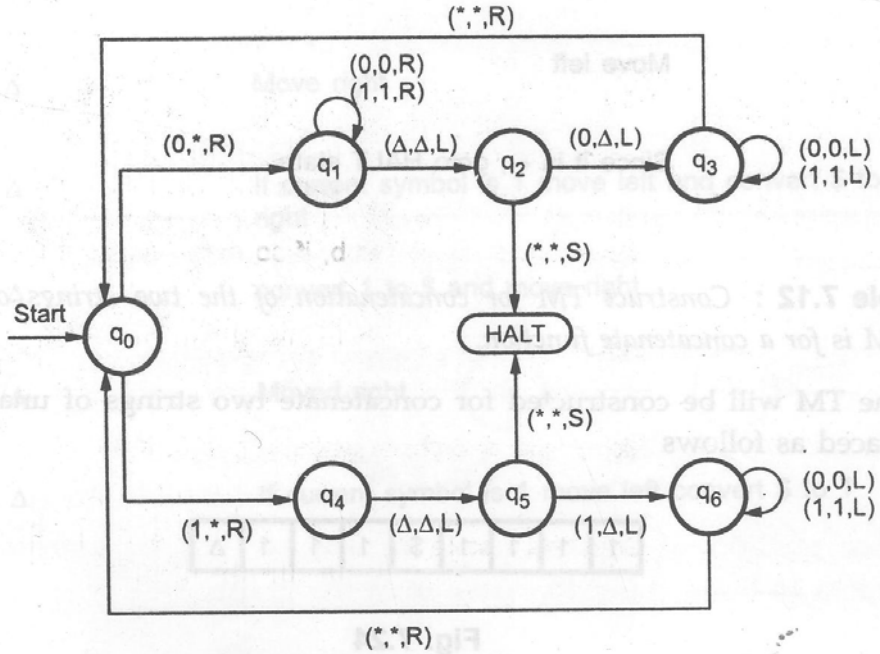


**Fig. 7.23**