UNIT 1 TURING MACHINE

Structure **Page Nos.**

1.0 INTRODUCTION

Not every problem can be solved through computational means
Gödel (1931)

Every system—natural or man-made, must be continuously, involved in some form of **computation** in its attempt at preserving its identity as a system.

The earth, revolves around the Sun along almost identical paths, revolution after revolution; being almost at the corresponding points in the paths after a specific period of time within the revolutions. So is true of every planet in the solar system. To be at the corresponding points in their paths, revolution after revolution, must involve some computation within the solar system. But, the same should be true of any system, not just of the solar system. Thus, phenomenon of **computation** is as universal as is the phenomena of **motion**. In order to have better understanding of the phenomena of motion, we think of different approaches, use some models and formulate some principles.

If a problem can be solved by some computational means, then there is a Turing Machine that solves the problem…..Turing Machine is an ultimate model of computation
Church-Turing Thesis (1936)

Similarly, attempts at *capturing the essence of the universal phenomenon of computation* are made through various approaches, models and principles.

As happens in the case of modeling of *motion*, inadequacy of one model (*e.g. Newtonian model*) in capturing essence of motions leads to another, more robust model *(e.g. Einstein's model),* so happens in the case of modeling of computation, as is discussed below.

In the previous units, we discussed two of the major *approaches* to modeling of computation viz. the automata/machine approach and linguistic/grammatical approach. Under grammatical approach, we discussed two models viz Regular Languages and Context-free Languages.

Under automata approach, we discussed two *models* viz. Finite Automata and Pushdown Automata.

Further, we defined the concept of **computational equivalence** and established that

Turing Machine is named so, in Honour of its inventor **Alan Mathison Turing** (1921-1954).
A.M. Turing, a British, was one of the greatest scholars of the twentieth century, and made profound contributions to the foundations of computer science. On the lines of Nobel prize, in memory of Alfred B. Nobel, for some scientific disciplines; ACM, to commemorate A.M. Turing, presents since 1966 annually Turing Award to an individual for contributions of a technical nature that are judged to be of lasting and major importance to the field of computing science.

(i) **Finite Automata** computational model is computationally equivalent to Regular Language Model.

(ii) **Push-down Automata Model** is computationally equivalent to context-Free language model.

(iii) Pushdown Automata (or equivalently Context-Free Language) model **is more powerful** computational model **in comparison** to Finite Automata (or equivalently Regular Language) model *in the sense that* every language accepted by Finite Automata is also recognized by Pushdown Automata. However, there are languages, viz. the language $\{x^n\ y^n : n \in N\}$, which are recognized by pushdown automata but not by Finite Automata.

(iv) There are languages, including the language $\{x^n\ y^n\ z^n: n \quad N\}$, which **are not accepted even** by Push-down automata.

… The limits of mathematics discovered by Turing sound more serious, more dangerous than the ones that Godel found. …And this is the invention of the computer, for this crazy kind of theoretical argument! You don't see billions and billions of dollars of technology in this 1936 paper, but it was all there in embryonic form, as von Neumann kept emphasizing: the universal Turing machine is really the notion of a general-purpose programmable computer…

Chaitin[#]

This prompts us to discuss other, still more powerful, automata models and corresponding grammar models of computation.

Turing machine (*TM*) is the next more powerful model of *automata approach* which recognizes more languages than Pushdown automata models do. Also **Phrase-structure model** is the corresponding grammatical model that matches Turing machines in computational power.

In this unit, we attempt a facile and smooth introduction to the concept of Turing Machine in the following order:

We give a formal definition of the concept and then illustrate the involved ideas through a number of examples and remarks.

We show how to realize some mathematical functions as TMs.

Further, we discuss how to construct more and more complex TMs through the earlier constructed TMs, starting with actual constructions (mathematically)of some simple TMs.

In a later unit, we discuss other issues like extensions, (formal) languages, properties and equivalences, in context of TMs.

Key words: Turing Machine (**TM**), Deterministic Turing Machine, Non-Deterministic Turing Machine, Turing Thesis, Computation, Computational Equivalence, Configuration of TM, Turing-Acceptable Language, Turing Decidable Language, Recursively Enumerable Language, Turing Computable Function

Notations

TM	:	Turing Machine
	:	Set of tape symbols, includes #, the blank symbol
	:	Set of input/machine symbols, does not include #
Q	:	the finite set of states of TM
F	:	Set of final states
a,b,c…	:	Members of
σ	:	Variable for members of
x or $\bar{x}$	:	Any symbol of *other than* x
#	:	The blank symbol
α, β, γ	:	Variables for String over
L	:	Move the Head to the Left
R	:	Move the Head to the Right
q	:	A state of TM, i.e, $q \in Q$
s or q_0	:	The start/initial state

# *Exploring Randomness* By Gregory J. Chaitin, Springer-Verlag (2001)

Halt or h:	The halt state. The same symbol h is used for the purpose of denoting halt state for all halt state versions of TM. And then h is not used for other purposes.
e or ∊ :	The empty string

$C_1 \vdash_M C_2$: Configuration C_2 is obtained from configuration C_1in *one* move Of the machine M

$C_1 \vdash^* C_2$: Configuration C_2 is obtained from configuration C_1in *finite* number of moves.

$w_1 \underline{a} w_2$: The symbol a is the symbol currently being scanned by the Head

Or

$w_1 \underset{\uparrow}{a} w_2$: The symbol a is the symbol currently being scanned by the Head

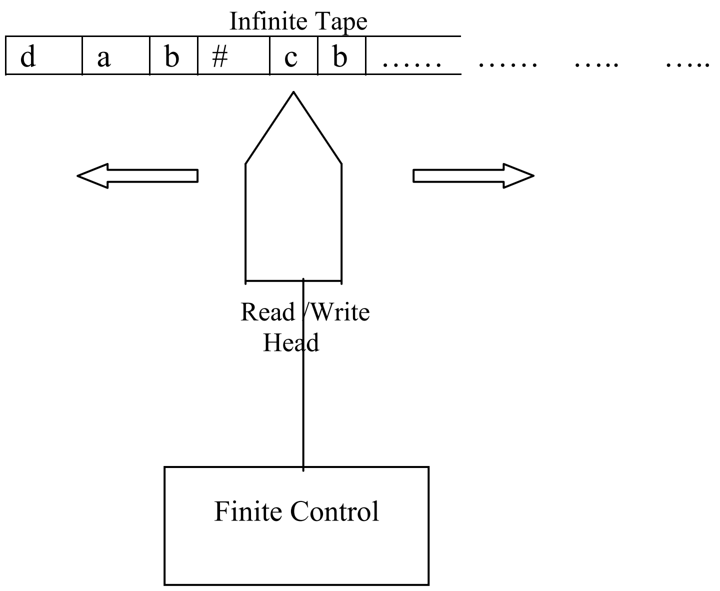
1.1 OBJECTIVES

After going through this unit, you should be able to:

- define and explain various terms mentioned under the title *key words* in the previous section.
- construct TMs for simple computational tasks
- realize some simple mathematical functions as TMs
- apply modular techniques for the construction of TMs for more complex functions and computational tasks from TMs already constructed for simple functions and tasks

1.2 PRELUDE TO FORMAL DEFINITION

In the next section, we will notice through a formal *definition of TM* that a TM is an *abstract* entity constituted of *mathematical objects* like sets and a (partial) function. However, in order to help our understanding of the subject-matter of TMs, we can *visualize* a TM as a physical computing device that can be represented as a *diagram as shown in1.2.1 below.*



TURING MACHINE
Fig. 1.2.1

Such a view, in addition to being more comprehensible to human beings, can be a quite *useful aid* in the design of TMs accomplishing some computable tasks, by allowing *informal* explanation of the various steps involved in arriving at a particular design. Without physical view and informal explanations, whole design process would be just a sequence of derivations of new formal symbolic expressions from earlier known or derived symbolic expressions   not natural for human understanding.

According to this view of TM, it consists of

(i) ***a tape***, with an end on the left but infinite on the right side. The tape is divided into squares or *cells*, with each cell capable of holding one of the tape symbols including the blank symbol #. At any time, there can be only *finitely* many cells of the tape that can contain non-blank symbols. The set of **tape symbols** is denoted by

As the very first step in the sequence of operations of a TM, **the input, as a finite sequence of the input symbols is placed in the left-most cells of the tape**. The set of **input symbols** denoted by , does not contain the blank symbol #. However, during operations of a TM, a cell may contain a ***tape symbol*** which is not necessarily an input symbol.
There are versions of TM, to be discussed later, in which the tape may be infinite in both left and right sides   having neither left end nor right end.

(ii) ***a finite control***, which can be in any one of the finite number of states.
The states in TM can be divided in three categories viz.

(a) the ***Initial state,*** *the state of the control just at the time when TM starts its operations. The initial state of a TM is generally denoted by q_0 or s.*
(b) the ***Halt state, which*** *is the state in which TM stops all further operations. The halt state is generally denoted by h. The halt state is distinct from the initial state. Thus, a TM HAS AT LEAST TWO STATES.*
(c) ***Other states***

(iii) ***a tape head*** *(or simply Head)*, is always stationed at one of the tape cells and provides communication for interaction between the tape and the finite control. The Head can ***read or scan*** the symbol in the cell under it. The symbol is communicated to the finite control. The control taking into consideration the symbol and its current state decides for further course of action including

the change of the symbol in the cell being scanned and/or

change of its state and/or

moving the head to the Left or to the Right. The control may decide not to move the head.

The course of action is called **a *move* of the Turing Machine. In other words, the move is a function of current state of the control and the tape symbol being scanned.**

In case the control decides for change of the symbol in the cell being *scanned*, then the *change is carried out by the head*. This change of symbol in the cell being scanned is called ***writing of the cell by the head***.

Initially, the head scans the left-most cell of the tape.

Now, we are ready to consider a **formal definition** of a Turing Machine in the next section.

1.3 TURING MACHINE: FORMAL DEFINITION AND EXAMPLES

There are a number of versions of a TM. We consider below *Halt State version* of formal definition a TM.

Definition: Turing Machine (*Halt State Version*)
A Turing Machine is a sextuple of the form (Q, , , , q_o, h), where

(i) Q is the finite set of states,

(ii) is the finite set of non-blank information symbols,

(iii) is the set of tape symbols, including the blank symbol #

(iv) is the **next-move** *partial* function *from* Q *to* Q {L, R, N}, where '***L***' denotes *the tape Head moves to the left adjacent cell*, '***R***' denotes *tape Head moves to the Right adjacent cell* and '***N***' denotes *Head does not move*, i.e., continues scanning the same cell.

In other words, for q_i Q and a_k , there exists (*not necessarily always, because is a partial function*) some q_j Q and some a_l such that (q_i a_k) = (q_j, a_l, x), where x may assume any one of the values 'L', 'R' and 'N'.

The **meaning** of (q_i, a_k) = (q_j, a_l, x) is that if q_i is the current state of the TM, and a_k is cell currently under the Head, then TM writes a_l in the cell currently under the Head, enters the state q_j and the *Head moves to the right adjacent cell, if the value of x is R*, *Head moves to the left adjacent cell, if the value of x is L* and continues scanning the same cell, if the value of x is N.

(v) q_0 Q, is the initial/start state.

(vi) h Q is the '*Halt State*', in which the machine stops any further activity.

Remark 1.3.1

Again, there are a number of variations in literature of even the above version of TM. For example, some authors allow at one time only one of the two actions viz. (i) writing of the current cell and (ii) movement of the Head to the left or to the right. However, this restricted version of TM can easily be seen to be computationally equivalent to the definition of TM given above, because one move of the TM given by the definition can be replaced by at most two moves of the TM introduced in the Remark.

In the next unit, we will discuss different versions of TM and issues relating to equivalences of these versions.

In order to illustrate the ideas involved, let us consider the following simple examples.

Example 1.3. 2:

Consider the Turing Machine (Q, , , , q_o, h) defined below that erases all the non-blank symbols on the tape, where the sequence of non-blank symbols does not contain any blank symbol # in-between**:**

Q= {q_o, h} = {a, b}, = {a, b, #}
and the next-move function is defined by the following table:

q: State	: Input Symbol	(q,)
q_0	a	{q_0, #, R}
q_0	b	{q_0, #, R}
q_0	#	{h, #, N}
h	#	ACCEPT

Next, we consider how to design a Turing Machine to accomplish some computational task through the following example. For this purpose, we need the definition.

A string Accepted by a TM

A string over is said to be accepted by a TM M = (Q, , , , q_0, h) **if** when the string is placed in the left-most cells on the tape of M and TM is started in the initial state q_0 **then** after a finite number of moves of he TM as determined by , Turing Machine is in state h (and hence stops an further operations. The concepts will be treated in more details later on. Further, a *string is said to be rejected* if under the conditions mentioned above, the TM enters a state q h and scans some symbol x, then (q, x) is not defined.

Example 1.3.3

Design a TM which accepts all strings of the form b^n d^n for n 1 and rejects all other strings.

Let the TM M to be designed is given by M = (Q, , , , q_0, h) with = { b, d}. **The values of Q, , , shall be determined by the design process explained below. However to begin with we take = {b, d, #}.**

We *illustrate the design process* by considering various types of strings which are to be accepted or rejected by the TM.

As input, we consider only those strings which are over {b, d}. Also, it is assumed that, when moving from left, occurrence of first # indicates termination of strings over

Case I: When the given string is of the form b^n d^m (b d)* for **n ≥ 1**, **m ≥ 1**as shown below for n = 2 m = 1

We are considering this particular type of strings, because, by taking simpler cases of the type, we can determine some initial moves of the required TM both for strings to be accepted and strings to be rejected.

b	b	d	-	-	-	-

Where '-' denotes one of b, d or #

Initially, TM should *mark* left-most b. *The term* ***mark*** *is used here in this sense that the symbol is scanned matching with corresponding b or d as the case may be.* To begin with, the TM should attempt to *match, from the left, the first b to the d which is the first d after all b's have exhausted.* For this purpose, TM **should move right skipping over all b's. And after scanning the corresponding d, it should move left**, until we reach the b, which is the last b that was marked.

Next, TM should mark the b, if it exists, which is immediately on the right of the previously marked b. i.e., should mark the b which is the left-most b which is yet to be marked.

But, in order to recognize the yet-to-be-marked left-most b, we must change each of the b's, immediately on marking, to some other symbol say B. Also, for each b, we attempt to find the left-most yet-to-be-marked d. In order to identify the left-most yet-to-be-marked d, we should change each of the d's immediately on marking it, by some other symbol say D.

Thus we require two additional Tape symbols B and D, i.e, = {b, d, B, D #}.

After one iteration of replacing one b by B and one d by D the tape would be of the form

B	b	D	-	-	-	-

and the tape Head would be scanning left-most b.

In respect of the states of the machine, we observe that in the beginning, in the initial state q_0, the cell under the Head is a b, and then this b is replaced by a B; and at this stage, **if we do not change the state then TM would attempt to change next b also to B** without matching the previous b to the corresponding d. But in order to recognize the form $b^n d^n$ of the string we do not want, in this round, other b's to be changed to B's before we have marked the corresponding d. Therefore

$$(q_0, b) = (q_1, B, R)$$

Therefore, the state must be changed to some new state **say q_1**. Also in order to locate corresponding d, the movement of the tape Head must be to the right. Also, in state q_1, the TM Head should skip over all b's to move to the right to find out the first d from left. Therefore, even on encountering b, we may still continue in state q_1. Therefore, we should have

$$(q_1, b) = (q_1, b, R)$$

However, on encountering a d, the behaviour of the machine would be different, i.e., now TM would change the first d from left to D and *start leftward journey. Therefore, after a d is changed to D, **the state should be changed to say q_2**. In state q_2 we start leftward journey jumping over D's and b's*. Therefore

$$(q_1, d) = (q_2, D, L) \quad \text{and}$$
$$(q_2, D) = (q_2, D, L) \quad \text{and}$$
$$(q_2, b) = (q_2, b, L)$$

In q_2, when we meet the first B, we know that none of the cells to the left of the current cell contains b and, if there is some b still left on the tape, then it is in the cell just to the right of the current cell. Therefore, we should move to the right and then if it is a b, **it is the left-most b on the tape and therefore the whole process should be repeated, starting in state q_0 again**.

Therefore, before entering b from the left side, TM should enter the initial state q_0. Therefore

$$(q_2, B) = (q_0, B, R)$$

For to-be-accepted type string, when all the b's are converted to B's and when the last d is converted to D in q_2, we move towards left to first B and then move to right in q_0 then we get the following transition:

from configuration

B	B	D	D	#	#

q_2

to configuration

B	B	D	D	#	#

q_0

Now we consider a special subcase of b^n d^m (b d)*, in which initially we have the following input

b	D	b	……..

Which after some moves changes to

B	D	b	

q_0

The above string is to be rejected. But if we take (q_0, D) as q_0 then whole process of matching b's and d's will be again repeated and then even the (initial) input of the form

b	d	b	#	#

will be incorrectly accepted. In general, in state q_0, we encounter D, if all b's have already been converted to B's and corresponding d's to D's. Therefore, the next state of (q_0, D) cannot be q_0.

Let

 (q_0, D) = (q_3, D, R)

As explained just above, for a string of the to-be-accepted type, i.e., of the form b^n d^n, in q_3 we do not expect symbols b, B or even another d because then there will be more d's than b's in the string, which should be rejected.

In all these cases, strings are to be rejected. One of the ways of rejecting a string say s by a TM is first giving the string as (initial) input to the TM and then by not providing a value of in some state q h, while making some move of the TM.

Thus the TM, not finding next move, stops in a state q h. Therefore, the string is rejected by the TM.

Thus, each of (q_3, b), (q_3, B) and (q_3, D) is undefined

Further, in q_3, we skip over D's, therefore

 (q_3, D) = (q_3, D, R)

Finally when in q_3, if we meet #, this should lead to accepting of the string of the form b^n d^n, i.e, we should enter the state h. Thus

 (q_3, #) = (h, #, N)

Next, we consider the cases *not* covered by b^n d^m (b d)* with $n \geq 1$, $m \geq 1$ are. Such

Case II when n = 0 but m 0, i.e. when input string is of the form d^m (b d)* for m 0.

Case III when the input is of the form b^n #, n 0

Case IV when the input is of the form # …………..

Now we consider the cases in detail.

Case II:

d	………

q_0

The above string is to be rejected, therefore, **we take (q$_0$, d) as undefined**

Case III. When the input is of the form b^n # #…#…, say

b	#	

q_0

After one round we have

B	#	

q_1

As the string is to be rejected, therefore,

(q$_1$, #) is undefined

Case IV: When # is the left-most symbol in the input

#	….	#	…	#	….

q_0

As the string is to be rejected, therefore, we take **(q$_0$, #) as undefined**

We have considered all possible cases of input strings over = {b,d} and in which, while scanning from left, occurrence of the first # indicates termination of strings over .

After the above discussion, the design of the TM that accepts strings of the form b^nd^n and rejects all other strings over {b, d}***, may be summarized as follows:***

The TM is given by (Q, , , , q_0, h) where
Q = {q_0, q_1, q_2, q_3, h}
= { b, d}
= {b, d, B, D, #}

The next-move partial function is given by

	b	d	B	D	#
q_0	{q_1, B, R)	*	*	(q_3, D, R)	*
q_1	{q_1, b, R)	{q_2, D, L)	*	{q_1, D, R)	*
q_2	{q_2, b, L)	*	(q_0, B, R)	{q_2, D, L)	*
q_3	*	*	*	(q_3, D, R)	(h, #, N)
h	*	*	*	*	Accept

'*' Indicates the move is not defined.

Remark 1.3.4

In general, such lengthy textual *explanation* as provided in the above case of design of a TM, *is not given*. We have included such lengthy explanation, as the *purpose is to explain the very process of design.* In general, table of the type given above along with some supporting textual statements are sufficient as solutions to such problems. In stead of tables, we may give Transition Diagrams (*to be defined*).

Ex. 1) Design a TM that recognizes the language of all strings of even lengths over the alphabet {a, b}.

Ex. 2) Design a TM that accepts the language of all strings which contain aba as a sub-string.

1.4 INSTANTANEOUS DESCRIPTION AND TRANSITION DIAGRAMS

1.4.1 Instantaneous Description

The following **differences** in the roles of tape and tape Head of **Finite** Automaton **(FA) and pushdown** Automaton **(PDA)** on one hand and in the roles of tape and tape head of **Tuning Machine** on other hand need to be noticed:

(i) The cells of the tape of an FA or a PDA are only read/scanned but are **never** changed/**written** into, whereas the cells of the tape of a TM **may be written** also.

(ii) The tape head of an FA or a PDA **always** moves from left to right. However, the tape head of a TM **can move in both** directions.
As a consequence of facts mentioned in (i) and (ii) above, we conclude that in the case of FA and PDA **the information in the tape cells already scanned do not play any role** in deciding future moves of the automaton, but in the case of a TM, the information contents of all the cells**, including the ones earlier scanned also play** a role in deciding future moves. This leads to the **slightly different definitions of configuration or Instantaneous Description (ID)** in the case **of a TM**.

The **total configuration** or, for short just, **configuration** of a Turing Machine is the information in respect of:

(i) Contents of all the cells of the tape, starting from the left–most cell up to atleast the last cell containing a non-blank symbol and containing all cells upto the cell being scanned.

(ii) The cell currently being scanned by the machine and

(iii) The state of the machine.

Some authors use the term *Instantaneous Description* instead of *Total Configuration*.

Initial Configuration: The total configuration at the start of the (Turing) Machine is called the initial configuration.

Halted Configuration: is a configuration whose state component is the Halt state

There are various *notations* used for denoting the total configuration of a Turing Machine.

Notation 1: *We use the notations, illustrated below through an example*:

Let the TM be in state q_3 scanning the symbol g with the symbols on the tape as follows:

#	#	b	d	a	f	#	g	h	k	#	#	#	#

Then one of the notations is

#	#	b	d	a	f	#	g	h	k	#	#	#	#

q_3

Notation 2: However, the above being a two-dimensional notation, is sometimes inconvenient. Therefore the following linear notations are frequently used:
(q_3,##bdaf#,g,hk), in which third component of the above 4-component vector, contains the symbol being scanned by the tape head.

Alternatively, the configuration is also denoted by (q_3,## bdaf# $\underline{g}$ hk), where the symbol under the tape head is underscored but two last commas are dropped.

It may be noted that the sequence of blanks after the last non-blank symbol, is not shown in the configuration. The notation may be alternatively written (q_3, w, g, u) where w is the string *to the left* and u the string *to the right* respectively of the symbol that is currently being scanned.

In case g is the left-most symbol then we *use the empty string* e instead of w. Similarly, if g is being currently scanned and there is no non-blank character to the right of g then we use e, the empty string instead of u.

Notation 3: The next notation neither uses parentheses nor commas. Here the state is written just to the left of the symbol currently being scanned by the tape Head. Thus the configuration (q_3, ##bdaf#, g, h, k) is denoted as # # bdaf#**q_3**ghk
Thus if the tape is like

g	w	#	…………

q_5

then we may denote the corresponding configuration as (q_5, e, g, u). And, if the tape is like

a	b	c	g	#	#	…

q_6

Then the configuration is (q_6, abc, g, e) or (q_6, abc $\underline{g}$) or alternatively as abcq_6g by the following notation.

1.4.2 Transition Diagrams

In some situations, **graphical** representation of the next-move (partial) function of a Turing Machine may give better idea of the behaviour of a TM in comparison to the **tabular** representation of .

A **Transition Diagram** of the next-move functions of a TM is a graphical representation consisting of a finite number of nodes and (directed) labelled arcs between the nodes. Each node represents a state of the TM and a label on an arc from one state (say p) to a state (say q) represents the information about the required **input symbol say x** for the transition from p to q to take place **and the action** on the part of the control of the TM. The action part consists of (i) the symbol say y to be written in the current cell and (ii) the movement of the tape Head.

Then the label of an arc is generally written as x/(y, M) where M is L, R or N.

Example 1.4.2.1

Let M ={Q, , , , q_0, h}
Where Q = { q_0, q_1, q_2, h}
= { 0, 1}
= {0, 1, #}
and be given by the following table.

	0	1	#
q_0	-	-	(q_2, #, R)
q_1	(q_2, 0, R)	(q_1, #, R)	(h, #, N)
q_2	(q_2, 0, L)	(q_1, 1, R)	(h, #, N)
h	-	-	-

Then, the above Turing Machine may be denoted by the Transition Diagram shown below, where we assume that q_0 is the initial state and h is a final state.

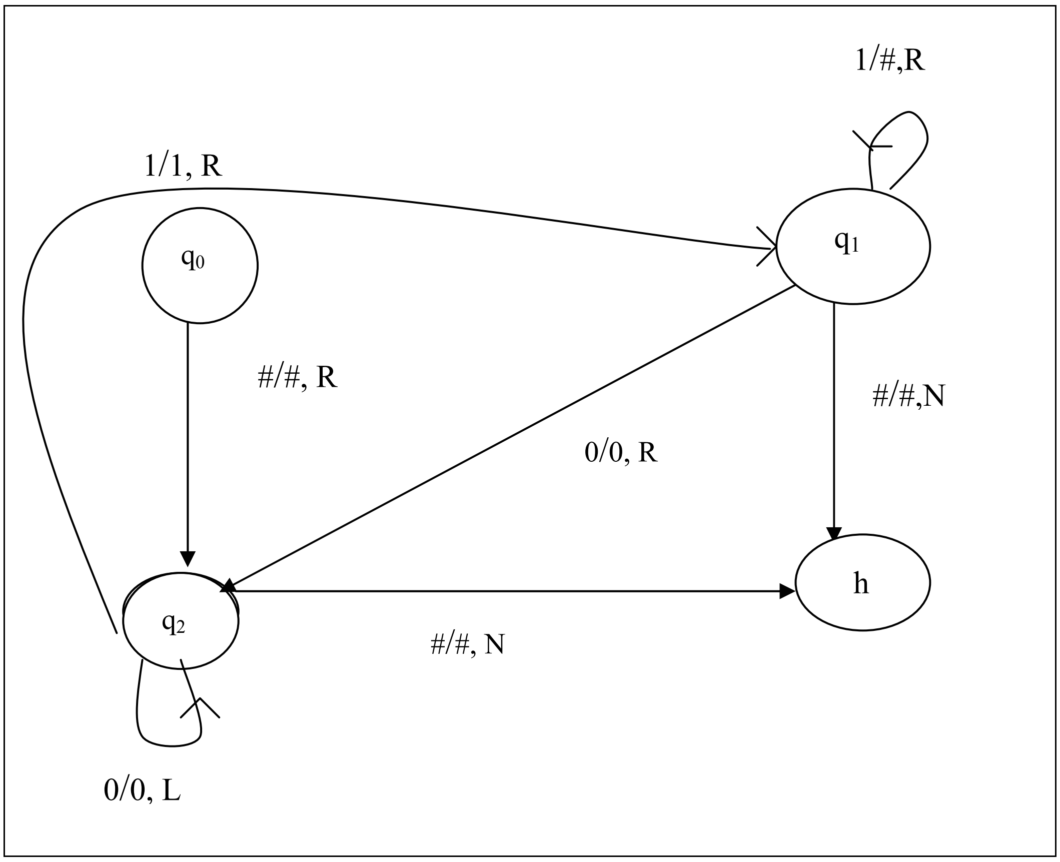


Fig. 1.4.2.1

Ex. 3) Design a TM M that recognizes the language L of all strings over {a, b, c} with
(i) number of a's = Number of b's = Number of c's and

(ii) if (i) is satisfied, the final contents of the tape are the same as the input, i.e., the initial contents of the tape are also the final contents of the tape, else rejects the string.

Ex. 4) Draw the Transition Diagram of the TM that recognizes strings of the form b^n d^n, n   1 and was designed in the previous section.

Ex. 5) Design a TM that accepts all the language of all palindromes over the alphabet {a, b}. A palindrome is a string which equals the string obtained by reversing the order of occurrence of letters in it. Further, find computations for each of the strings (i) babb (ii) bb (iii) bab.

Ex. 6) Construct a TM that copies a given string over {a, b}. Further find a computation of the TM for the string aab.

1.5 SOME FORMAL DEFINITIONS

In the previous sections of the unit, we have used, without formally defining some of the concepte like *move*, *acceptance* and *rejection* of strings by a TM. In this section, we define these concepts formally

In the rest of the section we assume the TM under consideration is

M= (Q, , , , q_0 h)

Definition: Move of a Turing Machine. *We give formal definition of the concept by considering three possible different types of moves,* viz.

'*move to the left',*

'move to the right', and

'*Do not Move'.*

For the definition and notation for Move, *assume the TM is in the configuration*
(q, a_1 a_2 … a_{i-1}, a_i , a_{i+1} … a_n)

Case (i) (a_i, q) = (b, p, L), for motion to left
Consider the following three subcases:

Case i(a) *if i > 1*, then the move is the activity of TM of going from the configuration

(q, a_1 a_2 … a_{i-1}, $\mathbf{a_i}$, a_{i+1} … a_n) to the configuration
(p, a_1 … a_{i-2}, $\mathbf{a_{i-1}}$, a_i a_{i+1} … a_n) and is denoted as
q, a_1 a_2 … a_{i-1}, $\mathbf{a_i}$, a_{i+1} … a_n) $\vdash_m$ (p, a_1…,a_{i-2}, $\mathbf{a_{i-1}}$, b, a_{i+1} … a_n).

The suffix M, denoting the TM under consideration, may be dropped, if the machine under consideration is known from the context.

Case i(b) *if i = 1*, the move leads to hanging configuration, as TM is already scanning left-most symbol and attempts to move to the left, which is not possible. *Hence move is not defined.*

Case i(c) *when i = n and b is the blank symbol #*, then the move is denoted as
(q, a_1 a_2 … a_{n-1}, $\mathbf{a_n}$, e) $\vdash$ (q, a_1 a_2 … a_{n-2}, a_{n-1}, ϵ,e).

Case (ii) (a_i, q) = (b, p, R), for motion to the right

Consider the following two subcases:

Case ii(a) if i < n then the move is denoted as

$(q, a_1 \ldots a_{i-1}, \mathbf{a_i}, a_{i+1} \ldots a_n) \vdash (p, a_1, \ldots a_{i-1}\ b\ \mathbf{a_{i+1}}, a_{i+2} \ldots a_n)$

Case ii(b) *if i = n* the move is denoted as

$(q, a_1 \ldots a_{n-1}, \mathbf{a_n}, e) \vdash (p, a_1 \ldots, \#, e)$

Case (iii) (a_i, q) = (b, p, 'No Move') when Head does not move.

then the move is denoted as

$(q, a_1 \ldots a_{i-1}, \mathbf{a_i}, a_{i+1} \ldots a_n) \vdash (p, a_1 \ldots a_{i-1}, \mathbf{b}, a_{i+1} \ldots a_n)$

Definition: A configuration results (or is derived) from another configuration.

We illustrate the concept through an example based **on say Case (iii)** above of the *definition of 'move'*. In this case, we say the configuration $(p, a_1 \ldots a_{i-1}, \mathbf{b}, a_{i+1} \ldots a_n)$ **results in a single move or is derived in a single move** from the configuration $(q, a_1 \ldots a_{i-1}, \mathbf{a_i}, a_{i+1} \ldots a_n)$. **Also**, we may say that the ***move yields*** the configuration $(p, a_1 \ldots a_{i-1}, b, a_{i+1} \ldots a_n)$ **or *the configuration*** $(q, a_1 \ldots a_{i-1}, a_i, a_{i+1} \ldots a_n)$ yields the configuration $(p, a_1 \ldots a_{i-1}, b, a_{i+1} \ldots a_n)$ **in a single move**.

Definition: Configuration results in n Moves or finite number of moves.

If, for some positive integer n, the configurations $c_1, c_2 \ldots c_n$ are such that c_i results from c_{i-1} **in a single move, i.e.,**

$c_{i-1} \vdash c_i$ for $i = 2, \ldots n$

then, we may say that *c_n results from c_1* ***in n moves or a finite number of moves***. The fact is generally denoted as

$$\mathbf{c_1 \vdash^n c_n} \quad \textbf{or} \quad \mathbf{c_1 \vdash^* c_n}$$

The latter notation is the preferred one, because generally n does not play significant role in most of the relevant discussions.

The notation $c_1 \vdash^* c_n$ also **equivalently stands** for the statement that **c_1 yields c_n in finite number of steps.**

Definition: Computation

If c_0 is an *initial* configuration and for some n, the configurations $c_1, c_2, \ldots, c_n$ are such that $c_0, \vdash c_1 \vdash \ldots \vdash c_n$, then, the sequence of configurations $c_0, c_1 \ldots c_n$ constitutes a *computation*

Definition: A string $\in$ *acceptable by a TM

is said to be acceptable by TM M if $(q_0, \ ) \vdash^* (h, r)$ for $r \in {}^*$

Informally, is acceptable by M, **if** when *the machine M is started in the initial state q_0 after writing on the leftmost part of the tape,* ***then****, if after finite number of moves, the machine M halts (i.e., reaches state h and of course, does not hang and does not continue moving for ever) with some string of tape symbols,* ***the original string is said to be accepted by the machine M***

Definition: Length of computation

If C_0 is initial configuration of a TM M and $C_0, C_1 \ldots, C_n$ is a computation, then n is called the length of the computation $C_0, C_1, \ldots C_n$.

Definition: Input to a computation

In the initial configuration, the string, which is on that portion of the tape beginning with the first non-blank square and ending with the last non-blank square, is called input to the computation.

Definition: Language accepted by a TM

M = (, , , , q_0, h), ***denoted by L(M),*** *and is defined as*

L(M) = { | * and if = $a_1 \dots a_n$ then

(q_0, e, a_1, a,…a_n) $|\!-^*$

(h, $b_1 \dots b_{j-1}, b_j, \dots b_{j,+1} \dots$ bn)

for some $b_1\, b_2 \dots b_n \in {}^*$

L(M), the language accepted by the TM M is the set of all finite strings over which are accepted by M.

Definition: Turing Acceptable Language

A language L over some alphabet is said to be *Turing Acceptable Language*, if there exists a Turing Machine M such that L = L (M)

Definition: Turing Decidable Language

There are at least two alternate, but of course, equivalent ways of defining a Turing Decidable Language as given below

Definition: A language L over , i.e, L * is said to be Turing Decidable, if both the languages L and its complement $^*\sim$ L are Turing acceptable.

Definition: A language L over , i.e, L * is said to be Turing Decidable, if there is a function

f_L: $^*\rightarrow$ { **$\underline{Y}$**, **$\underline{N}$**}
such that for each $\in {}^*$

$$f_L(\ ) = \begin{array}{ll} \underline{Y} & \text{if} \quad L \\ \underline{N} & \text{if} \quad L \end{array}$$

Remark 1.5.1

A very important fact in respect of Turing acceptability of a string (or a language) needs our attention. The fact has been discussed in details in a later unit about undecidability. However, we briefly mention it below.

For a TM M and an input string $\in {}^*$, even after a large number of moves we may not reach the halt state. However, from this we can neither conclude that 'Halt state will be reached in a finite number of moves' nor can we conclude that Halt state will not be reached in a finite number moves.

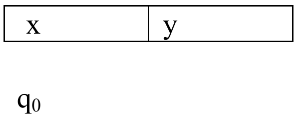
This raises the question of how to decide that an input string w is *not* accepted by a TM M.

An input string w is said to be '*not accepted*' by a TM M = (Q, , , , q_0, h) if any of the following three cases arise:

(i) There is a configuration of M for which there is no next move i.e., there may be a state and a symbol under the tape head, for which does not have a value.

(ii) The tape Head is scanning the left-most cell containing the symbol x and the state of M is say q and (x, q) suggests a move to the 'left' of the current cell. However, there is no cell to the left of the left-most cell. Therefore, move is not possible. The potentially resulting situation (can't say exactly configuration) is called **Hanging configuration**.

(iii) The TM on the given input w enters an infinite loop. For example if configuration is as



and we are given

$(q_0, x) = (q_1, x, R)$

and $(q_1, y) = (q_0, y, L)$

Then we are in an **infinite loop.**

1.6 OBSERVATIONS

The concept of TM is one of the most important concepts in the theory of Computation. In view of its significance, we discuss a number of issues in respect of TMs through the following remarks.

Remark 1.6.1

Turing Machine is not just another computational model, which may be further extended by another still more powerful computational model. It is *not only the most powerful computational model* known so far *but also is conjectured* to be *the ultimate computational model*. In this regard, we state below the

Turing Thesis: ***The power of any computational process is captured within the class of Turing Machines.***

It may be noted that Turing thesis is just a ***conjecture and not a theorem, hence, Turing Thesis*** can not be logically deduced from more elementary facts. *However*, the conjecture can be shown to be *false, if* a more powerful computational model is proposed that can recognize all the languages which are recognized by the TM model *and also* recognizes at least one more language that is not recognized by any TM.
In view of the unsuccessful efforts made in this direction since 1936, when Turing suggested his model, at least at present, it seems to be unlikely to have a more powerful computational model than TM Model.

Remark 1.6.2

The *Finite Automata* and *Push-Down Automata models* were used only as **accepting devices** for languages in the sense that the automata, when given an input string from a language, tells whether the string is *acceptable* or not. ***The Turing Machines are designed to play at least the following three different roles:***

(i) **As accepting devices for languages**, similar to the role played by FAs and PDAs.

(ii) **As a computer of functions.** In this role, a TM represents a particular function (*say the SQUARE function which gives as output the square of the integer given as input*). *Initial input* is treated as representing an **argument** of the function. And the (*final) string* on the tape when the TM enters the *Halt State* is treated as representative of the **value** obtained by an application of the function to the argument represented by the initial string.

(iii) **As an enumerator of strings of a language** that outputs the strings of a language, one at a time, in *some systematic order*, i.e, as a list.

Remark 1.6.3

Halt State of TM vs. set of Final States of FA/PDA

We have already briefly discussed the differences in the behaviour of TM on entering the Halt State and the behaviour of Finite Automata or Push Down Automata on entering a Final State.

A TM on entering the Halt State stops making moves and whatever string is there on the tape, is taken as output irrespective of whether the position of Head is at the end or in the middle of the string on the tape. However, an FA/PDA, while scanning a symbol of the input tape, if enters a final state, can still go ahead (*as it can do on entering a non-final state*) with the *repeated activities of moving to the right, of scanning the symbol under the head and of entering a new state etc. In the case of FA PDA, the portion of string from left to the symbol under tape Head is accepted if the state is a final state and is not accepted if the state is not a final state of the machine.*

To be more clear we repeat: the only *difference* in the two situations when an FA/PDA enters a final state *and* when it enters a non-final state is that in the case of the *first situation*, the part of the input scanned so far is said to be **accepted/recognized**, *whereas* in the *second situation* the input scanned so far is said to be **unaccepted**.

Of course, in the Final State version of TM (discussed below), the Head is allowed movements even after entering a Final State. Some definite statement like 'Accepted/Recognized' can be made if, in this version, the TM is in Final State.

Remark 1.6.4

Final State Version of Turing Machine

Instead of the version discussed above, in which a particular state is designated as *Halt* State, some authors define TM in which a subset of the set of states Q is designated as *Set of Final States*, which may be denoted by F. *This version is extension of Finite automata with the following changes, which are minimum required changes to get a Turing Machine from an FA.*

(i) The Head can move in both Left and Right directions whereas in PDA/FA the head moves only to the Right.

(ii) The TM, while scanning a cell, can both read the cell and also, if required, change the value of the cell, i.e., can *write in the cell*. In Finite Automata, the Head *only* can read the cell. It can be shown that the *Halt State* version of TM is *equivalent* to the *Final State* version of Turing Machine.

(iii) In this version, the TM machine halts only if in a given state and a given symbol under the head, no next move is possible. Then the (initial) input on the tape of TM, is unacceptable.

Definition: Acceptability of $\in$ * in Final State Version

Let $M_1 = (Q, \ , \ , \ , q_0, F)$

be a TM in final state version. Then w is said to be acceptable if C_0 is the initial configuration with w as input string to M_1 and

$$C_0 \vdash^* C_n$$

is such that

$C_n = (p, \alpha, a, \ )$

with p in F, set of final states, and a ∈ , the set of tape symbols, and α, ∈ *

Equivalence of the Two Versions

We discuss the equivalence only informally. If in the Halt state version of a TM in stead of the halt state h, we take F= {h} then it is the Final state version of the TM. *Conversely*, if F= { f_1, f_2,……f_r} is the set of final states then we should note the fact that in the case of acceptance of a string, a TM in final state version enters a final state *only once* and then halts with acceptance. Therefore if we rename each of the final state as h, it will not make any difference to the computation of an acceptable or unacceptable string over . Thus F may be treated as {h}, which further may be treated as just h.

1.7 TURING MACHINES AS COMPUTER OF FUNCTIONS

In the previous section of this unit, we mentioned that a Turing Machine may be used as

(i) A language Recognizer/acceptor

(ii) A computer of Functions

(iii) An Enumerator of Strings of a language.

We have already discussed the Turing Machine in the role of language accepting device. *Next, we discuss how a TM can be used as a computer of functions*

Remark 1.7.1

For the purpose of discussing TMs as computers of functions, we make the following assumptions:

A string over some alphabet say will be written on the tape as # #, where # is the blank symbol.

Also initially, the TM will be scanning the *right-most* # of the string # #.

Thus, the initial configuration, (q_0, # $\underline{\#}$) represents the starting point for the computation of the function with as input.

The assumption facilitates computation of composition of functions.

Though, most of the time, we require functions of one or more arguments having only integer values with values of arguments under the functions again as integers, yet, we consider functions with domain and codomain over *arbitrary* alphabet sets say $_0$ and $_1$ respectively, neither of which contains the blank symbol #.

Next we define what is meant by computation, using Turing Machine, of a function

$$f\!: \; {}_0^* \quad {}_1^*$$

Definition: A function f: f: $_0^*$ $_1^*$ is said to be ***Turing-Computable, or simply computable*, if** there is a Turing Machine M = (Q, , , , q_0, h), where contains the following holds:

(q_0, # $\underline{\#}$,) $|-^*_m$ (h, # $\underline{\#}$,) whenever $_0^*$ and $_1^*$ satisfying f( ) = .

Remark 1.7.2

It may be noted that, **if** the string contains some symbols from the set
- $_0$, i.e, symbols not belonging to the domain of f, **then** the TM may hang or may not halt at all.

Remark 1.7.3

Next, we discuss the case of **functions which require *k arguments***, where k may be any finite integer, greater than or equal to zero. For example,
the operation PLUS takes **two arguments** m and n and returns m + n.

The function f with the rule
f (x, y, z) = (2x + y) * z
takes **three arguments**.

The function C with rule
C ( ) = 17
takes **zero** number of arguments

Let us now discuss how to represent k distinct arguments of a function f on the tape. Suppose k = 3 and x_1 x_2, y_1 y_2 y_3 and z_1 z_2 are the three strings as three arguments of function f. **If** these three arguments are written on the tape as

#	x_1	x_2	y_1	Y_2	y_3	z_1	z_2	#

then the above tape contents may even be *interpreted as a single argument* viz.
x_1 x_2, y_1 y_2 y_3 z_1 z_2. Therefore, in order, **to avoid such an incorrect interpretation**, the arguments are separated by #. Thus, the above *three arguments* will be written on the tape as

#	x_1	x_2	#	y_1	Y_2	y_3	#	z_1	z_2	#

In general, if a function *f* takes k 1 arguments say $_1$, $_2$, …, $_k$ where each of these arguments is a string over $_0$ (i.e., each $_i$ belongs to $_0^*$) and if *f* ($_1$, $_2$, …, $_k$) = for some $_1^*$; then **we say *f* is Turing Computable** if there is a Turing Machine M such that

$$(q_0 , e, \#\ _1\#\ _2 \ldots \#\ _k\#, e) \quad |\!\!-^*_M (h, e, \#\ \#, e)$$

Also, **when f takes zero number of arguments and f( ) = then, we say f is computable**, if there is a Turing Machine M such that

$$(q_0 , e, \# \underline{\#}, e) \quad |\!\!-^*_M (h, e, \# \ \underline{\#}, e)$$

Remark 1.7.4

Instead of functions with countable, but otherwise arbitrary sets as domains and ranges, we consider only those functions, for each of which the domain and range is the *set of natural numbers*. This is not a serious restriction in the sense that any countable set can, through proper encoding, be considered as a set of natural numbers.

For natural numbers, there are various representations; some of the well-known representations are *Roman Numerals* (e.g. VI for six), *Decimal Numerals* (6 for six), *Binary Numerals* (110 for six). Decimal number system uses 10 symbols vis. 0, 1, 2, 3,4, 5, 6, 7, 8 and 9. Binary number system uses two symbols denoted by 0 and 1. **In the discussion of Turing Computable Functions, the *unary* representation described below is found useful.** *The unary number system uses one symbol only:*

Let the symbol be denoted by I then the number with *name six is represented as* I I I I I I. In this notation, *zero is represented by empty/null string*. Any other number say twenty is represented in unary systems by writing the symbol I, twenty times. In order to facilitate the discussion, **the number n, in unary notation will be denoted by I^n in stead of writing the symbol I, n times**.

The *advantage of the unary representation* is that, in view of the fact that most of the symbols on the tape are input symbols and if the input symbol is just one, then the *next state* will generally be determined by **only the current state**, because the other determinant of the next state viz tape symbol is most of the time the unary symbol.

We recall that for the set X, the notation X* represents the set of all finite strings of symbols from the set X. Thus, any function *f* from the set of natural number to the set of natural numbers, *in the unary notation*, is a function of the form *f* : {I}* {I}*

Definition: The function f: N N with f(n) = m for each n ∈ N and considered as f: {I}* {I}*, with {I} a unary number system, **will be called Turing Computable function**, if a TM M can be designed such that M **starting** in *initial tape configuration*

# I I ……. I $\underline{\#}$

with *n consective I's* between the two #'s of the above string, **halts** in the following configuration

# I I …… I #

containing f(n) = m I's between the two #'s

The above idea may be further generalized to the functions of more than one integer arguments. For example, SUM of two natural numbers n and m takes two integer arguments and returns the integer (n + m). The initial configuration with the tape containing the representation of the two arguments say n and m respectively, is of the form

# I I … I # I I ……I $\underline{\#}$

where the string contains respectively n and m I's between respective pairs of #'s and Head scans the last #. **The function SUM will be Turing computable if** we can design a TM which when started with the initial tape configuration as given above, *halts* in the Tape configuration as given below:

# I I … I I ….. I $\underline{\#}$

where the above string contains n + m consecutive I's between pair of #'s.

Example 1.7.5

Show that the SUM function is Turing Computable

The problem under the above-mentioned example may also be stated as: ***Construct a TM that finds the sum of two natural numbers.***

The following design of the required TM, *is not efficient* yet explains a number of issues about which a student should be aware while designing a TM for computing a function.

Legal and Illegal Configurations for SUM function:

In order to understand the design process of any TM for a (computable) function in general and that of SUM in particular, let us consider the possible *legal as well as illegal* initial configuration types as follows.

*Note: in the following, the sequence '…' denotes any sequence of I's possibly empty and the sequences ' ***' denotes any sequence of Tape symbols possibly empty and possibly including #. Underscore denotes the cell being scanned.*

Legal initial configuration types:

Configuration (i)

#	#	#	***

q_0

representing n = 0, m =0

Configuration (ii)

#	#	I	…	#	***

q_0

n = 0, m 0

Configuration (iii)

#	I	…	#	#	***

q_0

n 0, m = 0

Configuration (iv)

#	I	…	#	I	…	#	***

q_0

n 0, m 0

We treat the following configuration

#	…	#	…	#	…	#

q_0

containing two or more than two #'s to the left of # being scanned in initial configuration, as ***valid***, where '…' denotes sequence of I's only.

Some illegal initial configurations:

Configuration (v)

***	I …	***

↑

Where at least one of *** does not contain # and initially the Head is scanning an I or any symbol other than # . The configuration is invalid as it does not contain required number of #'s.

Configuration (vi), though is a special case of the above-mentioned configuration, yet it needs to be mentioned separately.

I	***	#	***

↑

Left most symbol is I or any other non-# symbol
Where *** does not contain any #,
Configuration (vii)

#	…	#	***

↑

Where *** does not contain # then the configuration represents only one of the natural numbers.

Also, in case of legal initial configurations, the final configuration that represents the result m + n should be of the firm.

		#	…..	#	

halt

with '…' representing exactly m + n I's.

Also in case of illegal initial configurations, the TM to be designed, should be in one of the following three situations indicating non-computability of the function with an illegal initial input, as explained at the end of Section 1.5:

(i) the TM has an infinite loop of moves;

(ii) the TM Head attempts to fall off the left edge (i.e. the TM has Hanging configuration); or

(iii) the TM does not have a move in a non-Halt state.

We use the above-mentioned description of initial configurations and the corresponding final configurations, in helping us to decide about the various components of the TM to be designed:

At this stage, we plan how to reach from an initial configuration to a final configuration. In the case of this problem of designing TM for SUM function, it is easily seen that for a legal initial configuration, we need to remove the middle # to get a final configuration.

(a) **Summing up** initially the machine is supposed to be in the initial state (say) q_0

(b) In this case of legal moves for TM for SUM function, first move of the Head should be to the *Left* only

(c) In this case, initially there are at least two more #'s on the left of the # being scanned. Therefore, to keep count of the #'s, *we must change state after* *scanning each #* . Let q_1, q_2 and q_3 be the states in which the required TM enters **after** scanning the three #'s

(d) In this case the movement of the Head, after scanning the initial # and also after scanning one more # on the left, should continue to move to the Left only, so as to be able to ensure the presence of third # also. Also, in states q_1 and q_2, the TM need not change state on scanning I.

Thus we have

$(q_0, \#) = (q_1, \#, L)$,

$(q_1, \#) = (q_2, \#, L)$

and

$(q_1, I) = (q_1, I, L)$, $(q_2, I) = (q_2, I, L)$.

However, from this point onward, the Head should *start moving to the Right.*

$(q_2, \#) = (q_3, \#, R)$.

Thus, at this stage we are in a configuration of the form

#			#			

q_3

For further guidance in the matter of the design of the required TM, we again look back on the legal configurations.

(e) In the configuration just shown above in q_3, if the symbol being scanned is # (as in case of *configuration (i) and configuration (ii))*, then the only action required is to skip over I's, if any, and halt at the next # on the right.

However, if the symbol being scanned in q_3 of the above configuration, happens to be an I (*as in case of configuration (iii) and configuration (iv))* then the actions to be taken, that are to be discussed after a while, have to be different.

But in both cases, movement of the Head has to be to the Right. Therefore, we need two new states say q_4 and q_5 such that

$(q_3, \#) \quad = \quad (q_4, \#, R)$

(***the processing scanning argument on the left, is completed***).

$(q_3, I) \quad = \quad (q_5, I, R)$

(*the scanning of the argument on the left, is initiated*).

Taking into consideration the cases of the initial configuration (i) and configuration (ii) we can further say that

$(q_4, I) \quad = \quad (q_4, I, R)$

$(q_4, \#) \quad = \quad (\text{halt}, \#, N)$

Next, taking into consideration the cases of initial configuration (iii) and configuration (iv) cases, we decide about next moves including the states etc in the current state q_5.

We are in the following general configuration
(that subsumes the initial configuration (iii) and configuration (iv) cases)

#	I		#		#	

q_5

Where the blank spaces between #'s may be empty or non-empty sequence of I's. Next landmark symbol is the next # on the right. Therefore, we may skip over the I's without changing the state i.e

$(q_5,\ I) \quad = \quad (q_5, I, R)$

But we must change the state when # is encountered in q_5, **otherwise, the** next sequence of I's will again be skipped over and we will not be able to distinguish between configuration (iii) and configuration (iv) for further necessary action. Therefore

$(q_5, \#) \quad = \quad (q_6, \#,\ R)$

(*notice that, though at this stage, scanning of the argument on the left is completed, yet we can not enter in state q_4, as was done earlier, because in this case, the sequence of subsequent actions have to be different. In this case, the# in the middle has to be deleted, which is not done in state q_4*)

Thus, at this stage we have the general configuration as

#		#			#	

q_6

Next, in q_6, if the current symbol is a #, as is the case in configuration (iii), then we must halt after moving to the left i.e.

$(q_6,\ \#) = (\text{halt}, \#,\ L)$

we reach the final configuration

0#	I	#	#	

halt

However, if we are in the configuration (iv) then we have

#	I	#	I		#	

q_6

Then the following sequence of actions is required for deleting the middle #:

Action (i): To remove the # in the middle so that we get a continuous sequence of I's to represent the final result. For this purposes, we move to the left and replace the # by I. But then it will give one I more than number of I's required in the final result.

Therefore

Action (ii): We must find out the rightmost I and replace the rightmost I by # and stop, i.e, enter halt state. In order to accomplish Action (ii) we reach the next # on the right, skipping over all I's and then on reaching the desired #, and then move left to an I over there. Next, we replace that I by # and halt.

Translating the above actions in terms of formal moves, we get

For Action (i)

$(q_6,\ I) \quad = \quad (q_7,\ I,\ L)$
$(q_7,\ \#) \quad = \quad (q_8,\ I,\ R)$

(*at this stage we have replaced the # in the middle of two sequences of I's by an I*)

For Action (ii)

(q_8, I) = (q_8, I, R)
(q_8, #) = (q_9, #, L)
(q_9, I) = (halt, #, N)

It can be verified that through above-mentioned moves, the designed TM does not have a next-move at some stage in the case of each of the illegal configurations.

Formally, the SUM TM can be defined as:

SUM = (Q, , , , q_0, h)
where Q = { q_0, q_1,….q_{10}, halt}
  = { I }
  = { I, # }
and

the next-move (partial) function is given by the Table

	I	#
q_0	-	(q_1, #, L)
q_1	(q_1, I, L)	(q_2, #, L)
q_2	(q_2, I, L)	(q_3, #, R)
q_3	(q_5, I, R)	(q_4, #, R)
q_4	(q_4, I, R)	(halt, #, N)
q_5	(q_5, I, R)	(q_6, #, R)
q_6	(q_7, I, L)	(halt, #, L)
q_7	-	(q_8, I, R)
q_8	(q_8, I, R)	(q_9, #, L)
q_9	(halt, #, N)	
halt	-	-

'–' indicates that *is not defined*

Remark 1.7.6

As mentioned earlier also in the case of design of TM for recognizing the language of strings of the form $b^n d^n$, the design given above **contains too detailed explanation** of the various steps. The purpose is to explain the involved design **process** in fine details for better understanding of the students. However, **the students need not supply such details** while solving a problem of designing TM for computing a function. While giving the values of Q, , explicitly and representing either by a table or a transition diagram, we need to give **only** some supporting statements to help understanding of the ideas involved in the definitions of Q, , and .

Example 1.7.7

Construct a TM that multiplies two integers, each integer greater than or equal to zero (*Problem may also be posed as: show that multiplication of two natural numbers is Turing Computable*)

Informal Description of the solution:

The legal and illegal configurations for this problem are the same as those of the problem of designing TM for SUM function. Also, the moves required to check the validity of input given for SUM function are the same and are repeated below:

(q_0, #) = (q_1, #, L)
(q_1, #) = (q_2, #, L)

$(q_1, I) = (q_1, I, L)$
$(q_2, \#) = (q_3, \#, R)$
$(q_2, I) = (q_2, I, L)$

Next, we determine the rest of the behaviour of the proposed TM.

Case I

When n = 0 covering configuration (i) and configuration (ii) The general configuration is of the form

#	#		#	

q_3

To get representation of zero, as, one of the multiplier and multiplic and is zero, the result must be zero. We should enter state say q_4 which skips all I's and meets the next # on the right.

Once the Head meets the required #, Head should move to the left replacing all I's by #'s and halt on the # it encounters so that we have the configuration

#	#		#	

Halt

The moves suggested by the above explanation covering configuration (i) and configuration (ii) are:

$(q_3, \#) = (q_4, \#, R)$
$(q_4, I) = (q_4, I, R)$
$(q_4, \#) = (q_5, \#, L)$
$(q_5, I) = (q_5, \#, L)$
$(q_5, \#) = (\text{Halt}, \#, R)$

Case II

Covering configuration (iii), we have at one stage

#	I		#	#	

q_3

If we take $(q_3, I) = (q_4, \#, R)$, then we get the following desired configuration in finite number of moves:

#	#	#		#	#	#	

Halt

Case III

While covering the configuration (iv), At one stage, we are in the configuration

		n I's		m I's			
#	I	…	#	I	#		

q_3

In this case, the final configuration is of the form

m n I's

#	#	…	#	I	I …	I	#	

Halt

The strategy to get the representation for n m I's consists of the following steps

(i) replace the left-most I in the representation of n by # and then copy the m I's in the cells which are on the right of the # which was being scanned in the initial configuration. In the subsequent moves, copying of I's is initiated in the cells which are in the left-most cells on the right hand of last I's on the tape, containing continuous infinite sequence of #'s.
Repeat the process till all I's of the initial representation of n, are replaced by #. At this stage, as shown in the following figure, the tape contains m I's of the initial representation of the integer m and additionally n.m I's. Thus the tape contains m extra #'s than are required in the representation of final result. Hence, we replace all I's of m by #'s and finally skipping over all I's of the representation of (n . m) we reach the # which is on the right of all the (n . m) I's on the tape as required.

Alternatively: In stead of copying n times of the m I's, we copy only (n-1) times to get the configuration

#		#	I	#	I ….. I	#	I	………	I	#	

m I's ((n-1).m) I's

Then we replace the # between two sequences of I's by I and replace the right-most I by # and halt.

The case of illegal initial configurations may be handled on similar lines as were handed for SUM Turing machine

Remark 1.7.8

The informal details given above for the design of TM for multiplication function are acceptable as complete answer/solution for any problem about design of a Turing Machine. However, if more detailed formal design is required, the examiner should explicitly mention about the required details.

Details of case (iii) are not being provided for the following reasons

(i) Details are left as an exercise for the students

(ii) After some time we will learn how to construct more complex machines out of already constructed machines, starting with the construction of very simple machines. One of the simple machines discussed later is a ***copying machine*** which copies symbols on a part of the tape, in other locations on the tape.

Ex. 7) Design a TM to compute the binary function MONUS (or also called PROPER SUBTRACTION) defined as follows:

Monus : N x N $\rightarrow$ N

(*Note 0 also belongs to N*)

such that

$$\text{monus (m, n)} = \begin{array}{ll} m \quad n & if\ m \quad n \\ 0 & else \end{array}$$

Ex.8) To compute the function n (mod 2)

Let if f denotes the function, then

f: N → {0, 1}
is such that

$$f(n) = \begin{array}{ll} 0 & if\ n\ is\ even, \\ 1 & if\ n\ is\ odd \end{array}$$

1.8 MODULAR CONSTRUCTION OF COMPLEX TURING MACHINES

In the previous example of constructing a Turing Machine even for a simple task of multiplying two numbers, we saw construction was quite complex. The handling of complexity can be attempted by looking at the total machine in terms of sub-machines.

In this section, we look at the task of constructing complex Turing Machines by suitably combining already constructed simplerTuring Machines. For this purpose, we discuss some *Basic* Machines and *Rules* for combining already constructed machines into more complex machines. *Also, we develop notation for expressing the involved rules and denoting the process for combining.*

We **begin by giving below rules** of combining Turing Machines to get more complex TMs from the already constructed Turing Machines. Let M be the TM which is to be constructed by combining the already constructed machines viz. M_1, M_2, …., M_k, where M_i = {Q_i, $_i$, $_i$, $_i$, q_{0i}, h_i} and M = {Q, , , , q_0, h} and M will start its actions in the machine M_1.

Then the rules for constructing M out of M_i are:

Rule 1: Assume all the sets Q_1, Q_2 … Q_k, are all mutually disjoint sets. If there is an overlap, then rename the elements of some sets so that all the sets are mutually disjoint.

Rule 2: The state q_{0i}, the initial state of M_1 will be the initial state of M i.e. $q_0 = q_{01}$; however, the initial state status of q_{02}, ….. q_{0k} is removed. Also, the set of states for M will contain as its subset each of Q_i for i = 1, 2, …, k.

Rule 3: The halt-state status of each h_i = i = 1, 2, …, k is removed and a new state h is included in Q which will serve as the halt state of M. However, each h_i remains astate of M, but its status as halt state is removed.

Thus
(***In the following*** $\bigcup$ ***denotes set union)***

$$Q = \bigcup_{i\ 1}^{k} Q_i \bigcup \{h\},$$

where h $\bigcup_{i\ 1}^{k} Q_i$

Rule 4: contains $\bigcup_{i\ 1}^{k} {}_{i}$ and

contains $\bigcup_{i\ 1}^{k} {}_{i}$.

It may be noted that may contain some more symbols, in addition to the symbols in $\bigcup_{i\ 1}^{k} {}_{i}$ and similarly may contain some more symbols, in addition to the symbols in $\bigcup_{i\ 1}^{k} {}_{i}$.

Rule 5(i): If the composite machine M is to halt on reaching h_i with symbol currently being scanned as x, then introduce a move $(h_i, x) = (h, x, N)$.

Rule 5(ii) If, in stead of halting in the state h_i, of machine M_i, while scanning the symbol x, the composite machine M is required to transfer the control to some machine say $M_b = \{Q_b, \ _b, \ _b, \ _i, q_{0b}, h_b\}$ in some state say p and the symbol x is required to be replaced by z then introduce the move $(h_i, x) = (p, z, N)$.
Diagrammatically we have

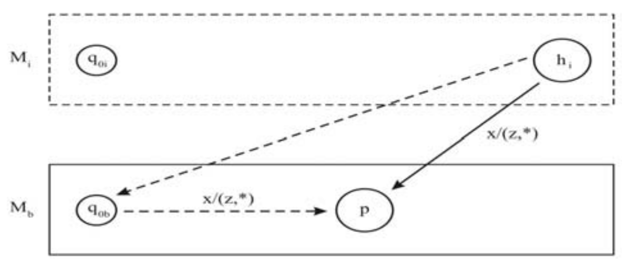


Fig. 1.8.1

This completes the details of the general rules for obtaining a composite machine out of already designed machines as components. However, there may be some special rules for design of each particular composite machine.

Example 1.8.1

Let $M_i = \{Q_i, \ _i, \ _i, \ _i, q_i, h_i\}$ for i = 1, 2
be two given TMs. We are required to construct a TM which first simulates M_1 and then M_2 and halts.

Then M is obtained by taking

M = (Q, , , , q, h)

Where

$Q = Q_1 \quad Q_2 \quad \{h\}$

$= \ _1 \quad \ _2, \quad = \ _1 \quad \ _2,\ q = q_1$ and $h = h_2$

and consists of

(i) all the moves defined by $_1$

(ii) all the moves defined by $_2$

(iii) $(h_1, x) = (q_2, x, N)$ for all x (*where q_2 is the initial state of M_2*)

In words: M is obtained by

(i) taking initial state q_1 of M_1 as initial state of M

(ii) removing halt state status of h_1 of M_1 and initial state status of q_2 of M_2

(iii) Introducing moves from the (old) halt state h_1 of M_1 to be (old) initial state q_2 of M for each symbol x of the tape s.t.

$(h_1, x) = (q_2, x, N)$

Diagrammatically M is given by

M

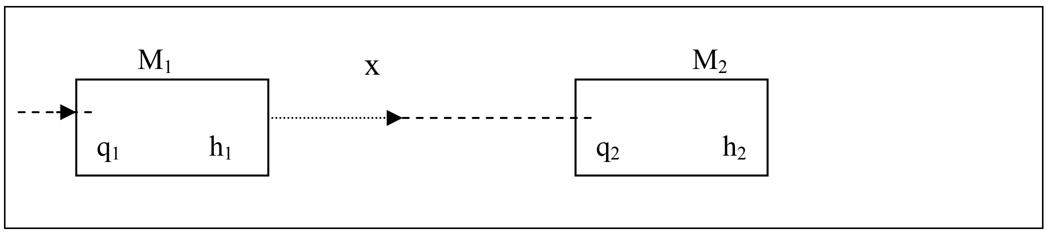


Fig. 1.8.2

Example 1.8.2

Let us consider one way of *combining the following three machines*. (*There are many possible ways of combining these three machines*). For all the three machines, the input symbol set = {0, 1} and = {0, 1, #} are the same. Further,

$M_1 = (Q_1, {}_1, {}_1, {}_1, q_{10}, h_1)$, which finds the first 1 after the current symbol and halts, and is given by $Q_1 = (q_{10}, q_{11}, h_1)$ with

$_1$ $(q_{10}, x) = (q_{11}, x, R)$ for each of x = 0, 1 and #
$_1$ $(q_{11}, 0\} = (q_{11}, 0, R)$,
$_1$ $(q_{11}, \#) = (q_{11}, \#, R)$; and
$_1$ $(q_{11}, 1) = (h_1, 1, N)$

$M_0 = (Q_0, , , {}_0, q_{00}, h_0)$, which finds the first 0 after the current symbol and halts, is given by $Q_0 = \{q_{00}, q_{01}, h_0\}$ with

$_0$ $(q_{00}, x) = (q_{01}, x, R)$ for each of x = 0, 1 and #,
$_0$ $(q_{01}, 1) = (q_{01}, 1, R)$,
$(q_{01}, \#) = (q_{01}, \#, R)$; and
$_0$ $(q_{01}, 0) = (h, 0, N)$

$M_3 = (Q_3, , , {}_3, q_3, h_3)$, which moves the tape Head one cell to the right and Halts where

$Q_3 = \{q_{30}, h_3\}$
$_3$ $(q_{30}, x) = (h_3, x, R)$ for each of x = 0, 1 or #.

*Now we combine the above three Turing Machines M_1, M_2 and M_3 as building blocks, so that the constructed composite Machine M finds the first occurrence of a **non-blank** symbol (i.e., symbol which is a 0 or a 1) after skipping two symbols, viz, currently being scanned symbol and the immediately next symbol. For example, the composite machine returns*

(i) 1 for each of the following input strings
0###10# or
00##10# or
001#00

(ii) 0 for each of the strings
11#0 or
11###0 or
110##1

The Turing Machine M is given by

$M = (Q, \ , \ , \ , q_{03}, h)$, where
$Q_1 = \{q_{00}, q_{01}, h_0, q_{10}, q_{11}, h_1, q_{30}, h_3, h\}$

In the machine M, q_{00} and q_{10} *are not* initial states. Also h_1, h_2, h_3, *are no more* halt states and h is the *new* Halt State.

In addition to simulating moves of M_0, M_1 and M_3, the following moves are added:

$(h_3, *) = (q_{00}, *, N)$
$(h_3,$ **$) = (q_{10}, , N)$,** where '*' is any symbol from ,
so that from M_3, we may go to M_0 or M_1on scanning any symbol.
Further in order to halt in the new machine, we introduce

$(h_0, *) = (h, *, N)$
$(h_1, *) = (h, *, N)$, where '*' is any symbol from ,

Note: The constructed machine is of Non-Deterministic (to be defined) type.
After appropriate shortcut Notations, the combined TM is graphically as shown below:

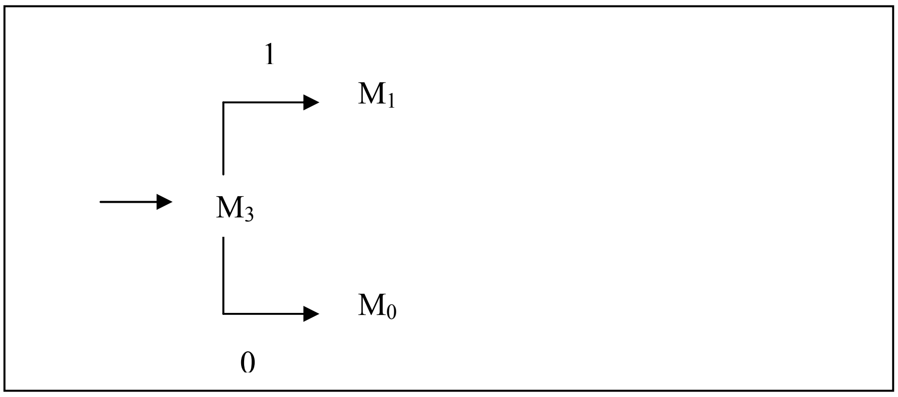


Fig. 1.8.3

Some Short cut Notations:

(i) If there is the same output and same next state for more than one inputs in a particular state, then single labeled arrow may be used instead of more than one arrow, e.g., The part of the transition diagram

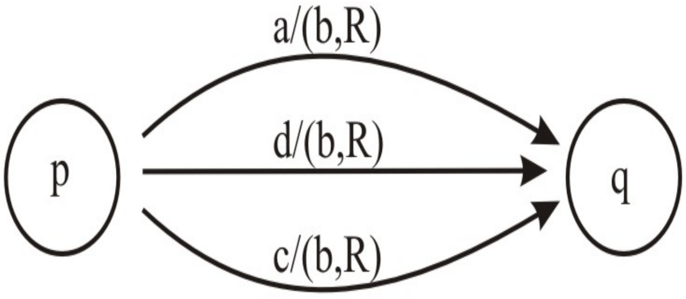


Fig. 1.8.4

may be replaced by

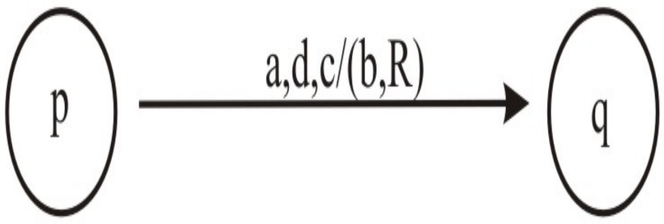


Fig. 1.8.5

(ii) Further, in the case discussed above, if   = {a, b, c, d}, is the set of tape symbols, then the diagram may be further modified as

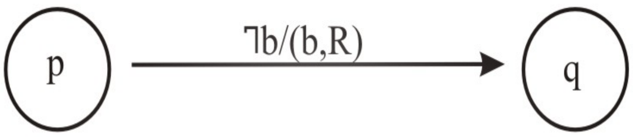


Fig. 1.8.6

where   denotes 'except for b, on all other tape symbols'.

The same shorthand is used when instead of **states** p and q in the two figures above, we have component **machines** M_1 and M_2.

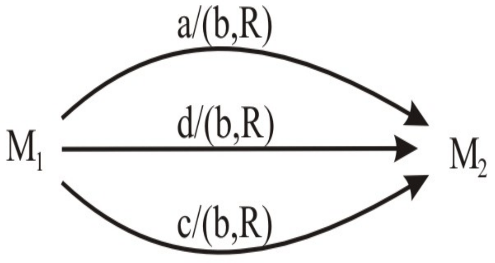


Fig. 1.8.7

Further, if on all inputs the composite machine operates as machine M_1 until M_1 halts, and then M_2 and then operates as M_2 would operate, then the following notation may be used.

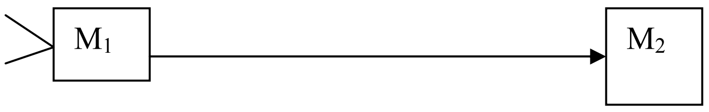


Fig. 1.8.8

Where there are no labels on the arrow.

(iii) If the composite machine M is such that first it operates as machine M_1 until it halts and then operates as say M_2 or M_3 depending on the symbol being scanned at the time of halting of M_1, say out of a or b respectively, then the following notation is used.

$$\begin{tikzcd}
> M_1 \arrow[r, "a"] \arrow[d, "b"'] & M_2 \\
M_3 &
\end{tikzcd}$$

Fig. 1.8.9

In the case of above composition of machines, in addition to all moves defined by $_1$ and $_2$ for machines M_1 and M_2 we have the additional moves:

(i) $(h_1, a) = (q_2, a, N)$ and

(ii) $(h_1, b) = (q_3, b, N)$,

where h_1 is the halt state of M_1 and q_2 and q_3 are the initial states of M_2 and M_3 respectively.

Some Basic Machines and Notations:

As the purpose is to explain how complex machines are obtained combining basic machines, the basic machines do not necessarily start scanning the left-most symbol.

There are two types of basic machines viz.

(i) **Symbol Writing Machines:** Let M = (Q, , , , q_0, h) where and let a be a particular symbol such that for some $(q_0, x) = (h, a, N)$ for all x (*where x is used in the sense of a variable, which actually is not a member of*).

This machine after starting in the initial state q_0 and reading any symbol, writes 'a' in place of the current symbol and halts.

We denote such machines by W_a or sometimes just by **a**

Where a may denote the symbol a as well as the machine that writes a.

However, context will resolve whether a particular occurrence denotes the symbol or the machine.

(ii) **Right/Left head Moving Machines:**

(a) ***Right Head Moving Machine***
Let M = (Q, , , , q_0, h) and is given by
$(q_0, x) = (h, x, R)$ for all x
(*where x is used in the sense of a variable, which actually is not a member of*).
This machine in the initial state q_0 scans the current symbol and whatever may be the current symbol, moves Head one square to the Right and halts.
Such a machine is denoted by R.

(b) ***Left Head Moving Machine***
Similarly, if a machine in the initial state q_0, scans the current symbol, and whatever may be the current symbol, it moves Head are square to the left and then halt;

such a machine is denoted by L.

(c) A machine which goes on moving to the Right except when it meets a specific symbol say a , and on meeting a, the machine halts. Such a machine is denoted by

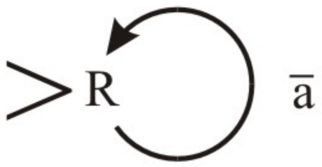


Fig. 1.8.10

Where $\bar{a}$ denotes any symbol from $\sim \{a\}$
Or is denoted by

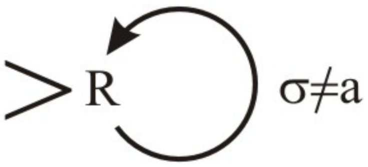


Fig. 1.8.11

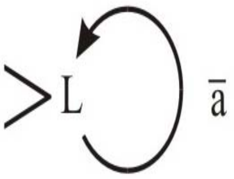
Or is denoted by just
R_a
(*note in R_a there is no bar on a*).

Therefore R_a finds the occurrence of first a to the right and halts.

(d) Thus $R_{\bar{\#}}$ denotes the machine, which finds the first non-blank symbol on the right.

In general, $R_{\bar{a}}$ denotes the machine which while moving to the right skips all a's and halts on finding a symbol different from a

(e) L_a finds the occurrence of the first a to the left and halts. $L_{\bar{\#}}$ denotes the machine, which finds the first non-blank symbol on the left and halts.
In general $L_{\bar{a}}$ denotes the machine which while moving to the left skips all a's and halts on finding a symbol different from a. On scanning the symbol a, the machine halts. Such a machine may be denoted by either of the following three notations:



OR

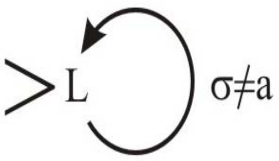


Fig. 1.8.12 **1.8.13**

Using the above notation and basic machines we provide notation for more complex machines.

Example 1.8.3

$> R \quad ^{a} \quad R$ denotes a machine, which in the initial state moves the Head one square to the Right and halts if the new symbol being scanned *is not a*. However, if the new symbol being scanned is *a* then, the Head moves Right once more.

Remark 1.8.4

(i) We should **note the difference between R_a and Ra (and similarly L_a and La)** R_a denotes the machine that finds the first a on the Right. But **Ra** denotes a machine which first moves to the right and then writes 'a' in place of the new symbol being scanned. Further,

(ii) the sequence like **R a R b L** denotes a combination of **five** machines, the first of which moves the Head to the Right and halts; then second machine writes an *a* in the current cell and halts; then the third machine again moves the Head to the Right and halts; then the fourth machine writes 'b' in the current cell and halts; and then finally the last machine moves the Head to the Left and halts. **Thus, if initially the Tape configuration is as follows:**

……… c b $\underline{a}$ b d # a c b # …….

Then after all the actions of the above-mentioned combined machine, the Tape configuration will be

c b *a* $\underline{a}$ b # a c b #

However, the combined machine R_aR_b L when starts in the same Tape configuration, viz.,

c b $\underline{a}$ b d # a c b #

will yield

c b a b d # a $\underline{c}$ b #.

R_aR_b L first searches for the next a to the right on the Tape through the machine R_a. Then R_a machine halts but R_b machine initiates and moves to the first b on the right and halts. Then the machine L initiates and moves the Head one cell to the Left.
Another Short-Hand: We use the notation

$$x,y,z \quad \} \quad r$$

to denote that when the current symbol is any one of x, y or z then the machine should proceed in the direction of the arrow with r representing the symbol which is actually present

For example, M is the composite machine

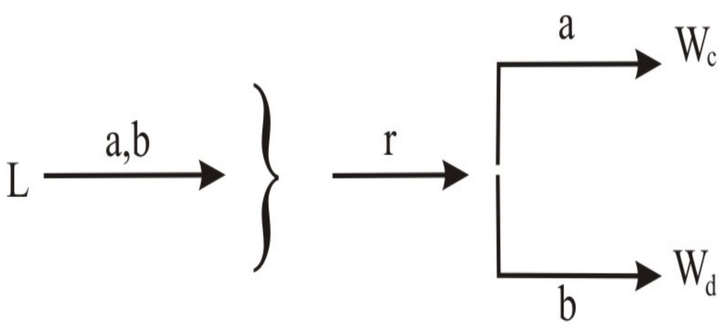


Fig. 1.8.14

and Tape configuration is

c b <u>a</u> d e …

Then the machine M, first moves the Head to the Left, and it finds ‘b’ there and hence activates the machine wd, which writes d in place of b and halts. Thus the configuration after M has executed and halted will be

c <u>d</u> a d e …

Using the shorthand notation introduced above, we describe a number of Turing Machines. Some of these machines would be quite useful in the construction of more complex machines and hence will be given standard names.
Example 1.8.5

S_R The right-shifting machine. The machine takes an input of the form

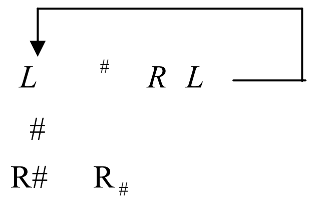
a <u>b</u> c b a #
and returns
a b c b a <u>#</u>
(*with one extra, # on the left hand side*)

First, we explain the strategy behind the construction of the machine S_R.

From the current position, we move to the cell on left and note the symbol over there. And if it is not # then copying it in the cell to the right of the cell of the noted element; i.e., we apply $L \quad {}^{\#} \quad R \ L$. The process is repeated unless the noted symbol is #. The process terminates on encountering #, followed by moving to the Right and writing # over there and then moving from there to the # on the right of given sequence of non-blank symbols. We may further explain the meaning of the expression

$$L \qquad {}^{\#} \qquad R \ \ L$$

In the above expression L means, first move to the Left. Then $\quad {}^{\#} \quad$ means note the symbol over there and call that symbol . If the noted symbol which we call is not # then execute R L. Otherwise take some other action denoted by a different arrow, if any Else stop. Next R L denotes that first move to the right, write down the symbol which was noted down earlier which we call and then move left again.
Therefore S_R is of the form



Let us call execution of the following loop, starting with left-most L as one **iteration.**

$$\begin{tikzcd} L \arrow[r, phantom, "^{\#}"] & R\,L \arrow[l, bend right] \end{tikzcd}$$

Then we explain the effect of each iteration as follows:
Let us start in the configuration.

$\# \; a \; b \; c \; b \; a \; \underline{\#} \; \#$

Then after one iteration we reach (just before the beginning of the left-most L)

$\# \; a \; b \; c \; \underline{b} \; a \; a \quad \#$

and after next iteration we reach the configuration

$\# \; a \; b \; \underline{c} \; b \; b \; a \quad \#$

Then we have

$\# \; a \; \underline{b} \; c \; c \; b \; a \quad \#$

Next, we have

$\# \; \underline{a} \; a \; b \; c \; b \; a \quad \#$

At this stage when S_R applies L the tape is of the configuration.

$\underline{\#} \; a \; a \; b \; c \; b \; a \quad \#$

Therefore, the branch R# is taken up.
i.e. we get

$\# \; \underline{\#} \; a \; b \; c \; b \; a \quad \#$

And, finally, when $R_{\#}$ is executed, then we get the configuration # #abcba$\underline{\#}$.

Example 1.8.6

To construct the *copy machine C* which takes a string of the form #   # in the initial state and gives, in the halt state, the configuration #   # $\underline{\#}$ where   is a string of tape symbols but not containing the blank symbol #.

First we explain how the proposed machine should work *through an example* and side by side, be as given below give the construction of C. Let initially, we be in the configuration

$\# \; b \; a \;\; c \; b \; c \; c \; a \; \underline{\#} \; \# \ldots$

Step I Move to the # which is on the left of the sequence of non-blank symbols. In other words we apply $L_{\#}$.
After this step we would be in the configuration

$\underline{\#} \; b \; a \; c \; b \; c \; c \; a \; \# \; \# \ldots$
(*i.e. first component machine would be $L_{\#}$*)

Step II Next we move right and note the symbol (*in this case b*) and replace it by # and **cross over** all non-blank symbols and first # on the Right to reach the second # on the right of non-blank symbols i.e. we have the configurations $\# \; \# \; a \; c \; b \; c \; c \; a \; \# \; \underline{\#}$ and we remember b also through   .

We write this b in place of # being scanned, to get the configuration
$\# \; \# \; a \; c \; b \; c \; c \; a \; \# \; \underline{b}$. This step may be summarized as

$$R \quad ^{\#} \quad \# R_{\#} R_{\#}$$

Step III. Then we should come back to the original position of b through $L_{\#}$ $L_{\#}$ and write back b. Thus, we reach the configuration

$\underline{b}$ a c b c c a # b # …

The machine component of Step III is given by

$$L_\# \; L_\#$$

Iterative Steps

Now copying of next symbol (*which is 'a' in this case*) can be carried out by applying the Step II followed by Step III once again.

Final Steps

The copying process should stop when we encounter #, after a finite number of repetitions of 'Step II and Step III. At this stage we should move to the # which is first on the right of the given string.

Thus the copying machine C is as given below:

$$\begin{tikzcd}
L_\# \arrow[r, no head] & R \arrow[r, no head, "\#"] \arrow[d, no head, "\#"'] & \# R_\# R_\# \; L_\# L_\# \arrow[l, no head, bend right] \\
& R_\# &
\end{tikzcd}$$

To have better understanding, we consider the traces of some more iterations.

After **second** iteration of Step II and Step III, the tape configuration is

b a $\underline{c}$ b c c a # b a # # . ..

After 7 iterations we get

# b a c b c c a $\underline{\#}$ b a c b c c a #

As the copying machine finally scans the # following the copied part through the last component $R_\#$ of the copying machine, is justified, in view of keeping the Head on the #, which is to the right of all non-blanks.

Finally, we get the configuration

# b a c b c c a # b a c b c c a $\underline{\#}$

Example 1.8.7

Design a Turing Machine that decrements one from a positive intger, using binary representation for integers.

Solution: In order to construct the desired machine, we consider some cases of Tape configurations representing the binary numbers before and after subtraction of 1.

Case (i) When the given binary number is represented on the tape in the form

$$\# \; x_1 \ldots x_k \, 1 \, \underline{\#}$$

where $x_1 = 1$ and x_i may be 0 or 1 i = 2, 3, … k,then after subtraction of 1, the representation of the number becomes

$$\# \; x_1 \ldots x_k \, 0 \, \underline{\#}$$

requiring the change to only the right-most bit.

Case (ii) When the given binary number is represented on the tape in the form

$$\# x_1, ..., x_k\ 1\ 0\ \underline{\#}$$

then after subtraction the binary number representation becomes

$$\# x_1 ... x_k\ \ 0\ \ 1\ \ \underline{\#}$$

requiring the two least significant bits to be reversed.

Case (iii) When the given binary number is represented on the tape in the form
$\# x_1 ... x_k\ 1\ \ \underbrace{\underline{0\ 0\ ...\ 0}}_{\text{i zeros}}\ \underline{\#}$

The number after subtraction of 1 is given by
$\# x_1 ... x_k\ 0\ \ \underbrace{\underline{1\ 1\ 1\ \ 1\ 1\ 1}}_{\text{i ones}}\ \ \underline{\#}$

Thus $1\ \underbrace{\underline{0\ 0\ \ ...\ 0}}_{\text{i zeros}}$ is replaced by $0\ \underbrace{\underline{1\ 1\ 1\ 1\ 1}}_{\text{i ones}}$

Thus in case (iii), which is a generalization of case (ii), each of all the continuous zeros from right to left, is replaced by a 1 and the 1, on the left of these 0's is replaced by a 0.

Case (iv) is again a special case of case (iii), in which the given binary number is represented in the form

$$\# 1\ \ 0\ \ 0 ...\ \ 0\ \#$$

then after subtraction of 1 we get the binary number representation of the form

$$\# \ 0\ 1\ 1\ 1\ 1\ 1\ 1\ \underline{\#}$$

However, in our binary representations, leading bit, i.e., left-most bit is always 1. Therefore, we need to delete the leading 0, by shifting the string '#0 1 1 1 1 1 1#'to the left so that we get # 1 1 1 1 1 1 $\underline{\#}$

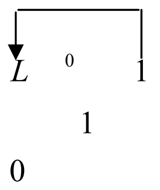
The process of subtraction of 1 from a binary number may now be summarized as follows:

Step I: The machine starts in the following configuration # # x_1 x_2 …. x_n $\underline{\#}$ where x_1 =l and x_i = 0 or 1 for i = 2, 3, ...

Step II: In view of the above case analysis, we attempt to find first 1 while moving from right to left and changing each of the 0 on the way to a 1. And when the Head scans the first 1, we change this 1 to a 0. This part of the machine may be represented by

$$\begin{tikzcd}
L \arrow[r, "0"] \arrow[d, "1"] & w_1 \arrow[l, bend right] \\
w_0
\end{tikzcd}$$

Where W_i denotes 'write i' which can also be denoted by just **i**, i.e., the above diagram may be denoted as:



Step III: Next step is to remove the leading zero, if any, by shifting the rest of the binary string to the Left. This situation may occur if the initial tape configuration is

# 1 0 0 ... 0 $\underline{\#}$

resulting in the configuration

# 0 1 1 ... 1 $\underline{\#}$.

In order that the representations is appropriate, having finally the most significant bit as 1, we shift the rest of the string to the left so that finally the tape configuration is of the form

1 1 ... 1 $\underline{\#}$

In order to execute Step III, we use $L_\#$ so that configuration becomes

$\underline{\#}$ 0 1 1 ... 1 #

Then move right and check the bit. *If the bit is a 1* then we move to right to the next # through $R_\#$. *If the bit is a 0, then* we execute the following steps:

Case III (i) Write a # over 0 to get

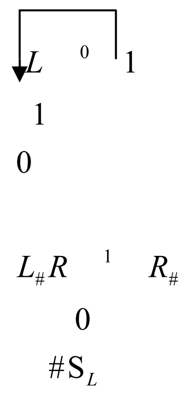
$\underline{\#}$ 1 1 1... 1

Then we use S_L so that we get # 1 1 ... 1 $\underline{\#}$.

Step III may be summarized as the machine

$$\begin{tikzcd} L_\# R \arrow[r, "1"] \arrow[d, "0"] & R_\# \\ \# S_L \end{tikzcd}$$

Combing the machines of Step I, Step II and Step III



Ex.9) Construct the machine S_L which transforms a string **#   #** to $\underline{\#}$, i.e., shifts each element of   one position to the Left.

Ex.10) To construct a Turing Machine which simulates a function

f: * * s.t.
if * i.e. (i.e., is of the form = a_1 a_2 ... a_k where a_i)
Then f( ) =
i.e. if = a_1, a_2, … a_k then f maps the configuration
#a_1 a_2 ... a_k $\underline{\#}$ to the configuration
a_1 a_2 ... a_k a_1 a_2 ... a_k $\underline{\#}$

Ex. 11) Design a TM that checks for palindromes over an alphabet {c, d}. In other words, if = {c, d} and w ∈ *, then the TM returns y for 'Yes' if w = w,^{R} and returns N for 'No' if w w^R.

1.9 SUMMARY

In this unit, after giving informal idea of what a Turing machine is, the concept is formally defined and illustrated through a number of examples. Further, it is explained how TM can be used to compute mathematical functions. Finally, a technique is explained for designing more and more complex TMs out of already designed TMs, starting with some very simple TMs.

1.10 SOLUTIONS/ANSWERS

Exercise 1: The transition diagram of the required TM is as shown below:

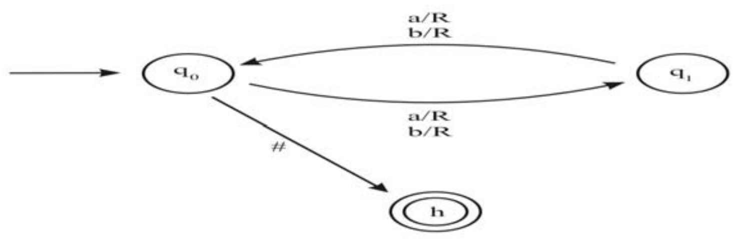


Fig.1.10.1

The required TM M = (Q, , , , q_0, h) with
Q = {q_0, q_1, h}
= {a, b} and = {a, b, #}.
The next move function is given by the transition diagram above. If the input string is of even length the TM reaches the halt state h. However, if the input string is of odd length, then TM does not find any next move in state q_1 indicating rejection of the string.

Exercise 2: The transition diagram of the required TM is as shown below,

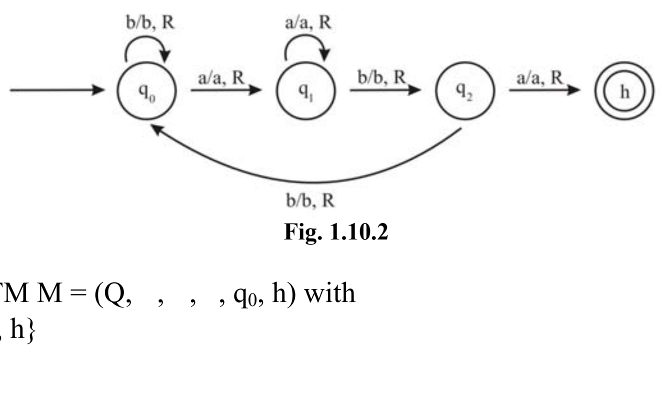


Fig. 1.10.2

The required TM M = (Q, , , , q_0, h) with
Q = {q_0, q_1, q_2, h}

= {a, b}   and   = {a, b, #}.
The next move function is given by the transition diagram above.
The transition diagram almost explains the complete functioning of the required TM. However, it may be pointed out that, if a string is not of the required type, then the blank symbol # is encountered either in state q_0 or in state q_1 or in state q_2. As there is no next move for (q_0, #), (q_1, #) or Q(q_2, #), therefore, the string is rejected.

Exercise 3: The transition diagram of the required TM is as shown below:

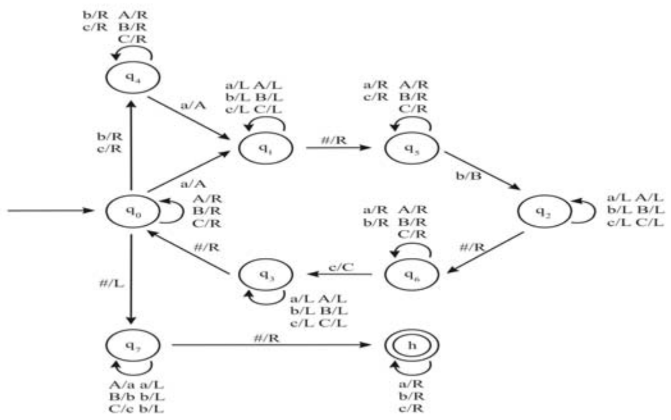


Fig. 1.10.3

The required TM M = (Q, , , , q_0, h) with

Q = {q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, h}
= {a, b, c}   and   = {a, b, c, A, B, C, #}   is shown by the diagram.

The design strategy is as follows:

Step I While moving from left to right, we find the first occurrence of **a** if it exists. If such an **a** exists, then we replace it by A and enter state q_1 either directly or after skipping b's and c's through state q_4.

In state q_1, we move towards left skipping over all symbols to reach the leftmost symbol of the tape and enter state q_5.

In q_5, we start searching for b by moving to the right skipping over all non-blank symbols except b and if such b exists, reach state q_2.

In state q_2, we move towards left skipping over all symbols to reach the leftmost symbol of the tape and enter q_6.

In q_6, we start searching for c by moving to the right skipping over all non-blank symbols except c and if such c exists, reach state q_3.

In state q_2, we move towards left skipping all symbols to reach the leftmost symbol of the tape and enter state q_0.

If in any one of the states q_4, q_5 or q_6 no next move is possible, then reject the string. **Else** repeat the above process till all a's are converted to A's, all b's to B's and all c's to C's.

Step II is concerned with the restoring of a's from A's, b's from B's and c's from C's, while moving from right to left in state q_7 and then after successfully completing the work move to halt state h.

Exercise 4: The Transition Diagram of the TM that recognizes strings of the form b^n d^n, n   1 and designed in the previous section is given by the following diagram.

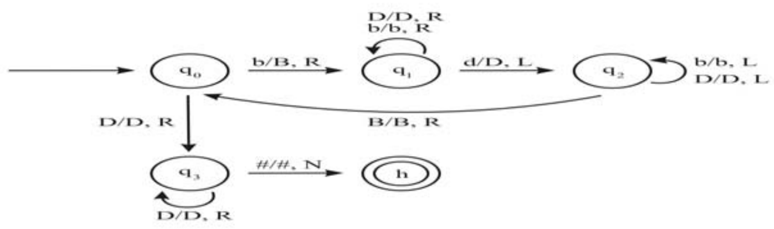


Fig. 1.10.4

Exercise 5: The transition diagram of the required TM is as shown below.

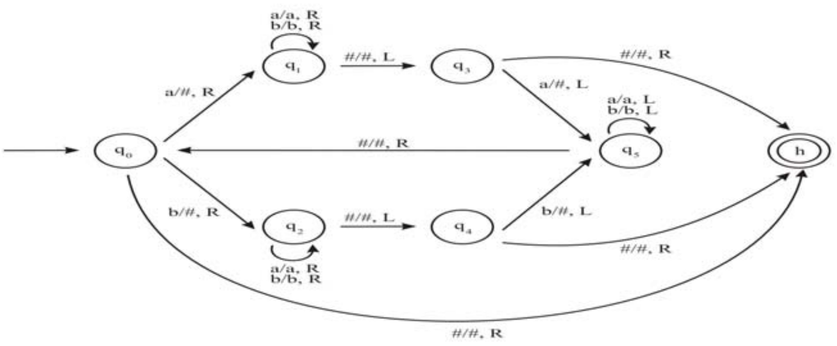


Fig. 1.10.5

The required TM M = (Q,   ,   ,   , q_0, h) with
Q = {q_0, q_1, q_2, q_3, q_4, q_5, h}
  = {a, b}   and   = {a, b, #}.
The next move function is given by the transition diagram above.

The proposed TM functions as follows:

(i) In state q_0, at any stage if TM finds the blank symbol then TM **has found a palindrome of even length**. Otherwise, it notes the symbol being read and attempts to match it with last non-blank symbol on the tape. **If the symbol is a, the TM replaces it by # goes to state q_1,** in which it skips all a's and b's and on #, the TM from q_1 will go to q_3 to find a matching a in last non-blank symbol position. **If a is found**, TM goes to q_5 replace a by #. However, if b is found then TM has no more indicating the string is not a palindrome. However, if in state q_2 only #'s are found, then it indicates that the previous 'a' was the middle most symbol of the **given string indicating palindrome of odd length.**

Similar is the case when b is found in state q_0, except that the next state is q_2 in this case and roles of a's and b's are interchanged in the above argument.

(ii) The fact of a string not being a palindrome is indicated by the TM when in state q_3 the symbol b is found or in state q_4 the symbol a is found.
The initial configuration is $q_0$babb.

The required computations are:

(i) q_0babb # q_2babb $\vdash$ #a$q_2$bb $\vdash$ #abbq_2# $\vdash$ #abq_4b $\vdash$ #aq_5b# $\vdash$ #$q_5$ab $\vdash$ q_5#ab
$\vdash$ #$q_0$ab $\vdash$ ##q_1b $\vdash$ #bq_1# $\vdash$ ##q_3b,
As there is no move in state q_3 on b, therefore, string is not accepted.

(ii) The initial configuration is $q_0$bb. Consider the computation:
$q_0$bb $\vdash$ #q_2b $\vdash$ #bq_2 $\vdash$ #q_4b # $\vdash$ q_5 ### $\vdash$ q_0 # $\vdash$ h#
(*We may drop #'s in the rightmost positions*).

(iii) The initial configuration is $q_0$bab. Consider the computation:

$q_0bab \vdash \#q_2ab \vdash^* \#aq_4b \vdash \#q_5a\# \vdash q_5\#\# \vdash \#q_0\# \vdash h\#$
(Note $\vdash^*$ denotes sequence of any finite number of $\vdash$).

Exercise6: The transition diagram of the required TM is as shown below.

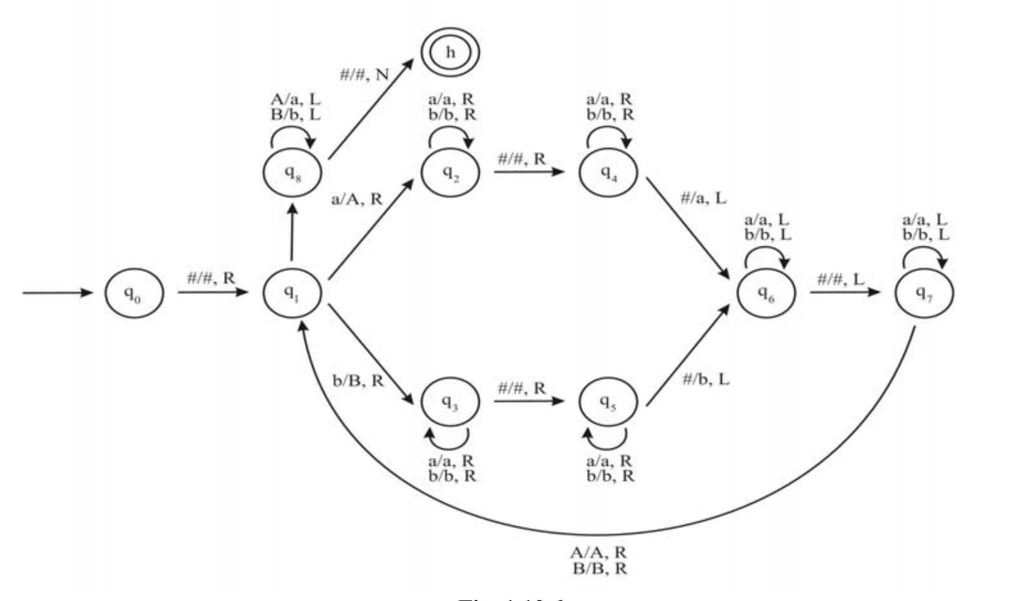


Fig. 1.10.6

The required TM M = (Q,   ,   ,   , q_0, h) with
Q = {q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, h}
  = {a, b}   and   = {a, b, #}.

The next move function is given by the transition diagram above.
In the solution of the problem, we can deviate slightly from our convention of placing the input string on the left-most part of the tape. **In this case, we place # in the leftmost cell of the tape followed by the** input string. Therefore, in the beginning in the initial state q_0, the TM is scanning # in stead of the first symbol of the input.
Before we outline the functioning of the proposed TM let us know that for the input string aab is placed on the tape as

#	a	A	b	#	#	***

and for the input, output on the tape is of the form

#	a	a	b	#	a	a	b	#	#	***

Outline of the functioning of the proposed TM

The TM in state q_1 notes the leftmost a or b, replaces it by A or B respectively and copies it in the next available # (the first # on the right is left as marker and is not taken as available). If the symbol in the state q_1 is a, then TM while skipping symbols passes through state q_2 and reaches q_4. However, if the symbol in state q_1 is b, then TM while skipping symbols passes through state q_3 and reaches state q_5. Then TM copies the symbol and reaches the state q_6. Next, TM starts its leftward journey skipping over a's, b's, A's, B's and # and meets A or B in q_7. At this stage, TM goes to state q_1. Then repeats the whole process until the whole string is copied in the second part of the tape.

But, in this process original string of a's and b's is converted to a string of A's and B's. At this stage TM goes from q_1 to state q_8 to replace each A by a and each B by b. This completes the task.

The Computation of the TM on input aab

The initial configuration is q_0#abb. Therefore, the computation is

$q_0\#abb \vdash \#q_1abb \vdash \#Aq_2ab$
$\vdash \#Aaq_2b \vdash \#Aabq_2\#$
$\vdash \#Aab\#q_4 \vdash \#Aabq_5\#a$
$\vdash \#Aabq_6b\#a$
$\vdash \#Aq_6ab\#a \vdash \#q_6Aab\#a$
$\vdash \#Aqab\#a$

(*At this point whole process is repeat and, therefore, we use* $\vdash^*$, *representing a finite number of* $\vdash$)

$\vdash^* \#AAq_0b\#aa$
$\vdash^* \#AABq_0b\#aab$

At this stage TM enters state q_7.

$\vdash \#AAq_7B\#aab$
$\vdash \#Aq_7Ab\#aab$
$\vdash \#q_7Aab\#aab$
$\vdash q_7\#Aab\#aab$
$\vdash h\#aab\#aab$

Exercise7 : In respect of the design of the TM (Q,   ,   ,   , q_0, h), where   = { I }   = { I, #} where we made the following observations:

Observation1: General form of the tape is

	#	I	….	I	#	I	..	I	#	

There are three significant positions of #, which need to be distinguished viz right-most # on left of I's, middle #, middle # and left-most # on the right of I's. Therefore, there should be change of state on visiting each of these positions of #.

Observation2: Initial configration is

	#	I	….	I	#	I	….	I	#	

q_0

and as observed above

(q₀, #) = (q₁, #, L)

The following forms of the tape

	I	I	#	

q_1

and

	#	#	

q_1

guide us to moves

$(q_1, I) = (q_2, \#, L)$

change of state is essential else other I's will also be converted to #'s,

$(q_1, \#) = (\text{ halt}, \#, N)$

Observations3: The moves are guided by principle that convert the left-most I to # on the right side the corresponding right-most I to # on the left-side

$(q_2, I) = (q_2, I, L)$
$(q_2, \#) = (q_3, \#, L)$
$(q_3, I) = (q_3, I, L)$
$(q_3, \#) = (q_4, \#, R)$

(We have reached the right-most # on the left of all I's as shown below)

#			#		#	

q_4

If we have configration of the form

#	#		#	

q_4

then it must have resulted from initial configuration in which m < n represented by say

#	I	I	#	I	I	I	#

q_4

Therefore, we must now enter a state say q_7 which skips all I's on the right and then halts
Therefore

$(q_4, \#) = (q_7, \#, R)$
$(q_7, I) = (q_7, I, R)$
$(q_7, \#) = (\text{halt}, \#, N)$

Next, we consider (q_4, I)
$(q_4, I) = (q_5, \#, R)$
(*state must be changed otherwise, all I's will be changed to #'s*)

$(q_5, I) = (q_5, I, R)$
$(q_5, \#) = (q_6, \#, R)$
(*the middle # is being crossed while moving from left to right*)

$(q_6, I) = (q_6, I, R)$
$(q^6, \#) = (q_0, \#, N)$
(the left-most # on right side is scanned in q_6 to reach q_0 so that whole process may be repeated again.)

Summarizing the above moves the transition table for function is given by

	I	#
q_0		$(q_1, \#, L)$
q_1	$(q_2, \#, L)$	(halt, #,L)
q_2	(q_2, I, L)	$(q_3, \#, L)$
q_3	(q_3, I, L)	$(q_4, \#, L)$
q_4	$(q_5, \#, R)$	$(q_7, \#, R)$

q_5	(q_5, I, R)	$(q_6, \#, R)$
q_6	(q_6, I, R)	$(q_6, \# R)$
q_7	(q_7, I, R)	(halt, #, N)
Halt	-	-

Exercise8: By our representation conventions, the initial configuration is as follows

#	I	. . .	I	#	#	…

q_0

$\underbrace{\qquad\qquad}_{n\,I's}$

If **n is even,** then f (n) = 0 which further is represented by final configuration

#	#	

halt

If n is odd, then f(x) = 1 which is represented by f (n) = 1 which is represented by a final configuration of the form

#	I	#	

halt

The strategy of reaching from initial configuration to a final configuration is that after scanning even number of I's we enter state q_2 and after scanning odd number of I's, we enter state q_1 and then take appropriate action, leading to the following (partial) definition of transition function :

$(q_0, \#) = (q_2, \#, L)$
$(q_2, I) = (q_1, \#, L)$
$(q_2, \#) = (\text{halt}, \#, N)$
$(q_1, I) = (q_2, \#, L)$
$(q_1, \#) = (q_3, \#, R)$
$(q_3, \#) = (\text{halt}, I, R)$

For the transition

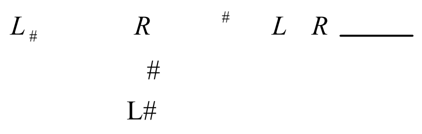
$(q_i, a_k) = (q_j, a_l, m)$, the sequence of actions is as follows: First a_l is written in the current cell so far containing a_k. Then movement of tape head is made to left, to right or 'no move' respectively according as the value of m is L, R or N. Finally the state of the control changes to q_j.

The transition function for the above computation is

	#	I
q_0	$(q_2, \#,L)$	$(q_1, \#, L)$
q_1	$(q_3, \#, R)$	$(q_2, \#, L)$
q_2	(halt, #, N)	$(q_1, \#, L)$
q_3	(halt, I, R)	-
halt	-	-

The students are advised to make transition diagram of the (partial) function defined by the above table.

Exercise 9: The desired machine S_L is given by



Exercise 10: Hint: The machine CS_L obtained by composing the earlier designed two machines C and S_L is the required machine.

Exercise 11: The proposed design is broken up into a number of the following steps:

Step I: is to mark the left end of the tape by writing a non-blank character say d in the left-most cell after shifting the given string to the Right.
Thus we apply S_R which transforms the tape configuration.
# <u>#</u> with *
to the configuration
<u>#</u> # #

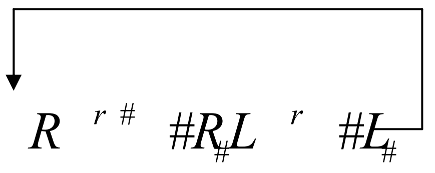
And then **we write d in the left most cell so** that tape configuration becomes
<u>d</u> # #

And the component TM for Step 1 is given by S_R $L_\#$ L d.

Step II: In order to move to the left-most non-blank symbol of the original string, **apply R** to reach the # which is to the left of the left-most non-blank symbol.

Step III: The following moves are repeatedly applied:
(i) Apply R to move to the left-most non-blank. The current symbol is read as r and then replaced by #. Then this r is attempted to be matched with the right-most non-blank symbol. The right-most non-blank symbol is reached by applying $R_\#L$. **Thus total machine component of Step III (i) is given by # $R_\#L$.**

(ii) *At this stage one of the possibilities is that* **the symbol currently being scanned is same as r**.
In this case the symbol is replaced by # and then the #, if any, to the left of non-blank part is reached through $L_\#$.
Whole process is repeated.
The TM component of the part discussed so far, Step III is of the form



At some stage, either of the following three cases happen.

(a) the tape contains string of #'s only.
(b) only one non-# symbol is left on the tape.
(c) right-most non-# symbol does not match r.

Out of these three cases, in the first two cases, the given string is a palindrome and hence the tape configuration.

	d	#…	#	

Or

	d	# … # r # …	

needs to be replaced finally by

	#	Y	<u>#</u>

In the third case (c) above, at some stage the tape may be of the form
d # # # ... # # ………… <u>b</u> # # …. #
In this case first all non-blanks need to be replaced by blanks and then final Tape configuration should be

N <u>#</u>

First we discuss **the cases when the string is a palindrome** and hence we need to replace the configuration

d # ... <u>#</u>   OR   d # …# <u>r</u> # …#

(In the configurations Head is scanning # or) by the configuration

y

In such cases at some stage, the current symbol is # or r, some non-blank symbol .

Let us first consider the case when head scans # (*i.e, case of even length palinidrome*)
We move to left-most symbol d through $L_{\overline{\#}}$, replace d by # then, move to the Right to write Y in the cell under the Head and finally move to the Right. Thus the TM component to handle this part is given by:

$$\overset{\#}{L_{\overline{\#}}} \ \#RYR$$

after R component of the component TM of Stage II.
Next we discuss the case when the initially given string is a palindrome of the form.
a b c b a <u>#</u>
for which, after a number of moves, the following configuration is reached:
d # # # <u>c</u> # # #

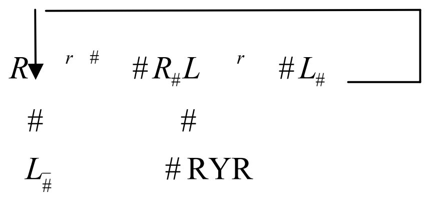
Then c is replaced by #. While executing $R \quad {}^{r \ \#} \quad \# R_{\#}$ part of the following component of the TM of Step III

$$R \quad {}^{r \ \#} \quad \# R_{\#} \quad L \quad {}^{r} \quad \# L_{\#}$$

leads to the configuration d # # # # <u>#</u> # # by replacing c by # and moving to the next # on the right. Next executing L of $L \quad {}^{r} \quad \# L_{\#}$ takes us back to the configurations d # ## <u>#</u> # # #, where in the case of even palindromes or for all states except last for odd palindromes, we expect under the Head at this stage, the previously noted symbol. But in this case it does not happen, because # is present in stead of the expect symbol c. Therefore the part ${}^{r} \quad \# L_{\#}$ is not executed.

Therefore, there is an accepting branch # from L. Afterward, actions are similar as in the case of even palindrome discussed above.

Combining the two cases we get the following component of the TM which correspond to the two cases of acceptance as Palindrome of the given string:



Case (iii) When the **given string is not a palindrome** and we have already reached a stage where the corresponding positions do not have the same letter e.g.

$$d\#\overset{\#}{a}\overset{\#}{b}ccc\overset{\#}{a}\#$$

in which after having executed

$$R \quad {}^{r\ \#} \quad \#R_{\#} \quad L \quad {}^{r\ \#} \quad \#L_{\#}$$

once completely and only upto ${}^{r\ \#}$ # $R_{\#}$ L in the second round we find a ‘c’ (instead of expected ‘b’)
at the stage to the component R ${}^{r\ \#}$ $R_{\#}$ L we add

another arc t r or #

(*in addition to the arc* ^{r} *when the pair of letters in corresponding positions match*) as shown in the **lower right** part of the next diagram.

Action-to-be defined once a non-palindrome is recognized: **Replace** all non-blanks by blanks so that the tape assumes the configuration.
d # …. <u>#</u> #
which finally through a series of actions assumes the form
N <u>#</u>

Coming back to the latest non-accepting configuration, the Head is scanning a non-blank (*in our example ‘c’*), we replace it by # and move to the next non-blank (if any) on the left-side, i.e., apply $L_{\overline{\#}}$. Thus application of $L_{\overline{\#}}$ # is repeated as long as there are non-blanks available on the tape. Also, as all non-blanks are continuous, therefore, in stead of # $L_{\overline{\#}}$, we may take only # L. As all non-blanks are continuous, therefore, we reach # only when the configuration is of the form d <u>#</u> #. This configuration is to be replaced by
N <u>#</u>

The sequence of actions required for the final configuration is $L_{\overline{\#}}$ # R N R.
After combining all the above sub-machines, we get

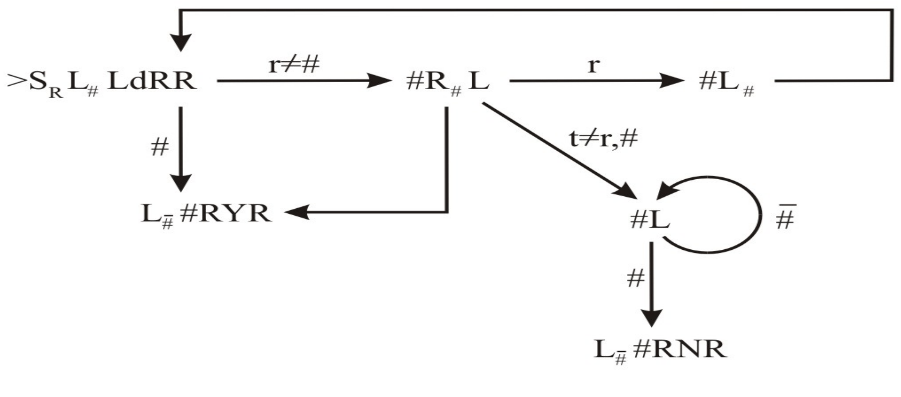


Fig. 1.10.7

1.11 FURTHER READINGS

1. H.R. Lewis & C.H.Papadimitriou: Elements of the Theory of computation, PHI, (1981)
2. J.E. Hopcroft, R.Motwani & J.D.Ullman: Introduction to Automata Theory, Languages, and Computation (II Ed.) Pearson Education Asia (2001)
3. J.E. Hopcroft and J.D. Ullman: Introduction to Automata Theory, Language, and Computation, Narosa Publishing House (1987)
4. J.C. Martin: Introduction to Languages and Theory of Computation, Tata-Mc Graw-Hill (1997)