8. TURING MACHINES

CHAPTER - 8
TURING MACHINES

In the previous chapters, we have studied the concepts of regular language and context-free languages, and their association with finite automata and pushdown automata respectively.

It reveals that regular languages is proper subset of context free language. Therefore pushdown automata is more powerful than finite automata.

If we compare finite automata and pushdown automata, we see the nature of temporary storage creates the difference. If we extrapolate this observation, we will discover even more powerful languages families, if we give the automata more flexible storage.

8.1 TURING MACHINE (TM) - INTRODUCTION

Turing machine is a simple mathematical model of a computer. It was proposed by the mathematician "*Alan Turing*" in 1936.

It is similar to a finite automaton but with an unlimited and unrestricted memory. Turing machine, is a much more accurate model of a general purpose computer.

Turing machine can do everything that a real computer do. If turing machine cannot solve certain problems, then these problems are beyond the theoretical limits of computation.

8.1.1 Turing Machine Model

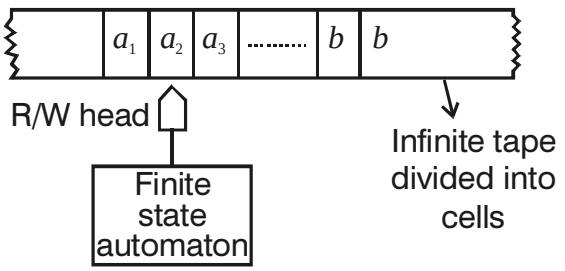


Figure.8.1 Turing Machine model

The turing machine can be thought of as a finite state automaton connected to a R/W (Read/Write) head. It has an infinite tape which is divided into number of cells.

Each cell stores one symbol. The input to and the output from the finite state automata (or) control unit are affected by R/W head which can examine one cell at a time.

In one move, the machine examines the present symbol under the R/W head on the tape and the present state of an automata to determine.

(i) a new symbol to be written on the tape in the cell under the R/W head.

(ii) a motion of the R/W head along the tape; either the head moves one cell left (L), or one cell right (R).

(iii) the next state of automaton

(iv) whether to halt or not.

8.1.2 Definition of TM

In one move, depending upon the symbol scanned by the tape head and the state of finite control, the turing machine:

(i) Changes state

(ii) Prints a symbol on the tape cell scanned, replacing what was written there,

(iii) Moves its head left (or) right one cell.

Turing machine T(M) is defined as

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

Q is the finite set of states

Γ is the finite set of allowable tape symbols.

B is a symbol of Γ, is the blank

Σ a subset of Γ not including B, is the set of input symbols.

δ is the next move function, a kind mapping of function, defined as

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

$q_0 \in Q$ is the initial state

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

8.2 REPRESENTATION OF TURING MACHINES

We can describe turing machine with three difference ways. They are:

(i) Instantaneous Descriptions

(ii) Transition table

(iii) Transition diagram

8.2.1 Representation by Instantaneous Descriptions

An Instantaneous Description (ID) of the turing machine M is denoted by $\alpha_1 q \alpha_2$. Here $q \in Q$ is the current state of M. $\alpha_1\alpha_2$ is the string in Γ^* that is the contents of the tape upto the rightmost non blank symbol or symbol to the left of the head, whichever is rightmost.

We define a *move* of M as follows. Let $x_1x_2..... x_{i-1}qx_i.....x_n$ be an ID. Let $\delta(q_i,x_i) = (p,y,L)$. If $i-1=n$, then take x_i as B(blank). If $i=1$, then there is no next ID, as the tape head is not allowed to fall off the left end of the tape. If $i>1$, then we write

$$x_1x_2..... x_{i-1}qx_i.....x_n \vdash_M x_1x_2.....x_{i-2}px_{i-1}yx_{i+1}.....x_n \quad .....(8.1)$$

If $\delta(q, x_i) = (p, y, R)$, then change of ID is

$$x_1, x_2....x_{i-1}qx_i....x_n \vdash_M x_1x_2.....x_{i-1}ypx_{i+1}.....x_n \quad .....(8.2)$$

If $i-1 = n$, the string $x_i..... x_n$ is empty and the right side of (8.2) is longer than left side.

Note :

If one ID results from the another by one move, they are related by the symbol $\vdash_M$. If one ID results from another in finite number of moves (including zero moves), they are related by the symbol $\vdash_M^*$.

Example 8.1

A snapshot of turing machine is shown in Figure 8.2, obtain the instantaneous description.

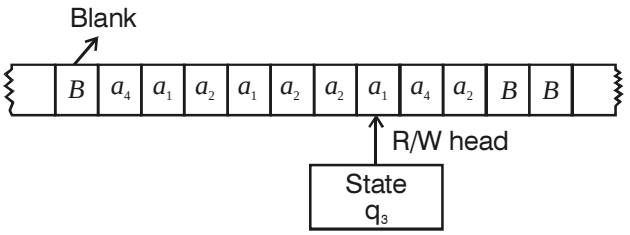
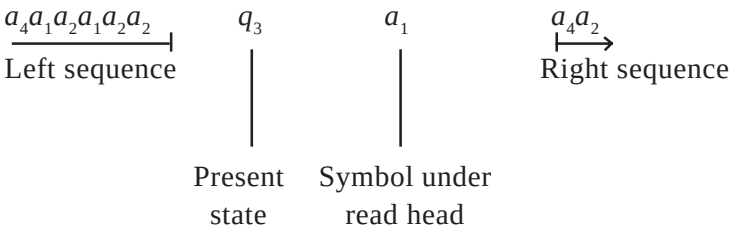


Figure 8.2

Solution :

The present symbol under R/W head is a_1. The present state is q_3. So a_1 is written to the right of q_3. The nonblank symbols to the left of a_1 form the string $a_4a_1a_2a_1a_2a_2$ which is written to the left of q_3. The sequence of nonblank symbols to the right of a_1 is a_4a_2.

The ID is



Note :

For constructing the ID, simply insert the current state in the input string to the left of the symbol under R/W head.

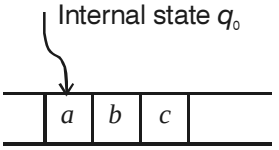
Example 8.2

Show the situation before and after the move caused by the transition.

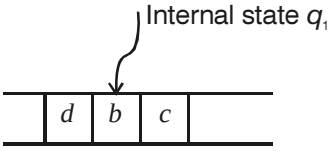
$\delta(q_0,a) = (q_1, d, R)$

Solution :

Situation before the move :



After the move :



$\delta(q_0,a)$ denotes the current state of control unit q_0 and the current tape symbol being processed or read is a.

(q_1, d, R) denotes the result of new state q_1 in control unit, a new tape symbol which replace the old one, and a move to the right.

Example 8.3

Consider a Turing Machine defined as

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

$F = \{q_1\}$

and $\delta(q_0,a) = (q_0, b, R)$

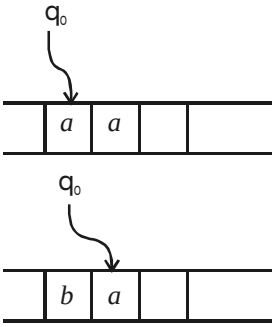
$\delta(q_0, b) = (q_0, b, R)$

$\delta(q_0,B) = (q_1, B, L)$

Process the string $w = aa$, and the initial state of turing machine is q_0.

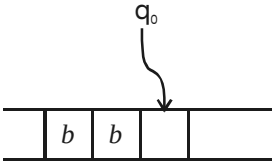
Step 1 :

TM is in state q_0, with the symbol a on the read-write head, the applicable transition is $\delta(q_0, a) = (q_0, b, R)$. So the read-write head will replace a by b and then move right on the tape.



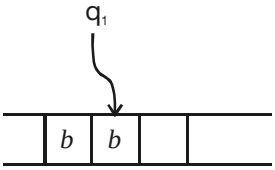
Step 2 :

The machine is still in state q_0. The subsequent a will also be replaced by a b, but b's will not be modified.



Step 3:

The machine halts on encountering first blank in state q_1 (final state).



Example 8.4

Consider the TM defined as :

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

F = $\{q_1\}$ *a*nd

$\delta(q_0,a) = (q_1, a, R)$

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

$\delta(q_0,B) = (q_1,B,R)$

$\delta(q_1,a) = (q_0, a, L)$

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

$\delta(q_1, B) = (q_0, B, L)$

Check the validity for the string *abab*

Solution :

The control unit is in q_0, and read-write head is in *a*. It reads *a* but does not change it.

Its next state is q_1 and read-write moves right, so that it is now over *b* (second symbol).

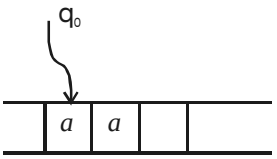
This symbol is also left unchanged. The machine goes back into state q_0 and the R/W head moves left. We are now back to the original state, and the sequence of moves starts again. The machine is in an *infinite loop*.

Example 8.5

Write the instantaneous descriptions of Example 8.3.

Solution :

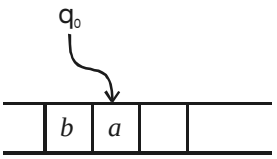
1.



The ID is *left sequence - state - tape symbol - right sequence*

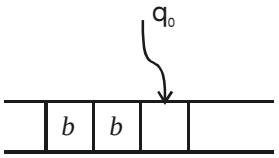
$\therefore$ ID is $Bq_0aa = q_0aa$

2.



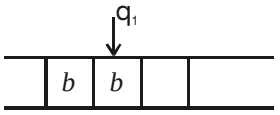
The symbol has been replaced. Now the ID = bq_0a

3.



ID is bbq_0B

4.



It has traversed from right to left

Hence ID is bq_1b

8.2.2 Representation by Transition Table

We give the definition of δ in the form of a table called the transition table. If $\delta(q,a) = (\gamma,\alpha,\beta)$, we write αβγ under *a*-column and *q*-row.

So if we have αβγ in the table, it means that α is written in the current cell, β gives the movement of the R/W head (L or R) and γ denotes the new state into which the Turing machine enters.

Example 8.6

Present States	Tape Symbols		
	B	0	1
Start → q_1	$1Lq_2$	$0Rq_1$	
q_2	BRq_3	$0Lq_2$	$1Lq_2$
q_3		BRq_4	BRq_5
q_4	$0Rq_5$	$0Rq_4$	$1Rq_4$
◎ q_5	$0Lq_2$		

Figure 8.3 Turing machine transition table

Turing machine has 5 states q_1..... q_5, where q_1 is initial state and q_5 is the only final state.

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

$\Gamma = \{0, 1, B\}$ B-blank

δ is given in the above Figure

Draw the computation sequence of input string 00.

Solution :

The computation sequence is described interms of the contents of the tape and current state.

If the string is $a_1a_2.... a_ja_{j+1}....a_m$ and the TM (Turing machine) is in state q is to read a_{j+1}, then we write $a_1a_2..... a_jqa_{j+1}..... a_m$. Likewise for the input string 00B, we get the following sequence.

$q_1$00B $\vdash$ 0$q_1$0B$\vdash$ 00q_1B $\vdash$ 0$q_2$01

$\vdash$ $q_2$001 $\vdash$ q_2B001 $\vdash$ B$q_3$001

$\vdash$ BB$q_4$01 $\vdash$ BB0$q_4$1 $\vdash$ BB01q_4B

$\vdash$ BB010q_5 $\vdash$ BB01$q_2$00 $\vdash$ BB0$q_2$100

$\vdash$ BBB1$q_4$00 $\vdash$ BBB10$q_4$0 $\vdash$ BBB100q_4B

$\vdash$ BBB1000q_5B$\vdash$ BBB100$q_2$00$\vdash$ BBB10$q_2$000

$\vdash$ BBB1$q_2$0000$\vdash$ BBB$q_2$10000$\vdash$ BBq_2B10000

$\vdash$ BBB$q_3$10000 $\vdash$ BBBB$q_5$0000

Since the machine halts in q_5 (final state) it is accepted.

Example 8.7

Consider the TM described by the transition table. Describe the processing of (a) 011, (b) 0011, (c) 001. Which of the above strings are accepted by M?

	Present State	Tape Symbols				
		0	1	*x*	*y*	*B*
Start →	q_1	xRq_2				BRq_5
	q_2	$0Rq_2$	yLq_3		yRq_2	
	q_3	$0Lq_4$		xRq_5	yLq_3	
	q_4	$0Lq_4$		xRq_1		
	q_5				yRq_5	BRq_6
	◎ q_6					

Solution :

(a) $q_1$011 $\vdash$ $xq_2$11 $\vdash$ q_3xy1

$\vdash$ xq_5y1 $\vdash$ $xyq_5$1

As $\delta(q_5,1)$ is not defined, the input string 011 is rejected, (or) not accepted.

(b) $q_1$0011 $\vdash$ $xq_2$011 $\vdash$ $x0q_2$11 $\vdash$ $xq_3$0y1

$$\vdash q_4x0y1 \vdash xq_1 0y1 \vdash xxq_2y1$$
$$\vdash xxyq_2 1 \vdash xxq_3yy \vdash xq_3xyy$$
$$\vdash xxq_5yy \vdash xxyq_5y \vdash xxyyq_5\text{B}$$
$$\vdash xxyy\text{B}q_6$$

M halts. As q_6 is an accepting state the input string 0011 is accepted.

(c) $q_1001 \vdash xq_201 \vdash x0q_21 \vdash xq_30y$
$\vdash q_4x0y \vdash xq_10y \vdash xxq_2y$
$\vdash xxyq_2$

M halts. As q_2 is not an accepting state, 001 is not accepted in M.

8.2.3 Representation by Transition Diagram

We can represent the transitions of a Turing machine pictorially, much as we did for the *PDA*. A *transition diagram* consists of a set of nodes corresponding to the states of the TM. An arc from state q to state p is labeled by one or more items of the form X/YD, where X and Y are tape symbols, and D is a direction, either L or R. that is, whenever $\delta(q,X) = (p, Y, D)$, we find the label X/YD on the arc from q to p. However, in our diagrams, the direction D is represented pictorially by $\leftarrow$ for "left" and $\rightarrow$ for "right".

As for other kinds of transition diagrams, we represent the start state by the word "Start" and an arrow entering that state. Accepting states are indicated by double circles. Thus, the only information about the TM one cannot read directly from the diagram is the symbol used for the blank. We shall assume that symbol is B unless we state otherwise.

Example 8.8

Design a turing machine to recognise all strings of odd number of 0's, where $\sum = \{0\}$.

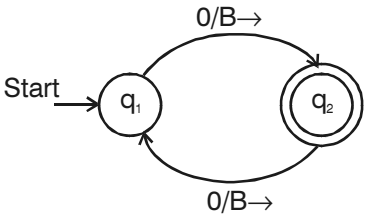
Solution :

(i) q_1 is the initial state.

(ii) On scanning the first zero enters the state q_2, second zero q_1, third zero q_2 and so on.

(iii) If the string ends in q_2 then it holds odd no. of zero's.

Transition diagram is:



$\therefore$ Q = $\{q_1, q_2\}$
Γ = {0, B}
S = q_1
F = $\{q_2\}$

8.3 LANGUAGE ACCEPTANCE OF TM

The language accepted by Turing machine (M) denoted by L(M), is defined as:

$L(M) = \{w : w \in \Sigma^* \text{ and } q_0 w \overset{*}{\vdash} \alpha_1 p \alpha_2, \text{ for some } p \in F, \alpha_1, \alpha_2 \in \Gamma^*\}$

When a turing machine processes an input string *w*, there are three possibilities. They are:

(i) Turing machine can accept the string by entering *accepting state*

(ii) It can reject the string by entering *non-accepting state*.

(iii) It can enter an *infinite loop* so that it never halts.

Example 8.9

Design a turing machine M to accept the language

$L = \{0^n 1^n : n \geq 1\}$ Compute 0011.

Solution :

Initially the tape of M contains $0^n 1^n$ followed by infinity of blanks. Starting at leftmost 0, we check it off by replacing it with some other symbol, say *x*. We then let the read-write head travel right to find the leftmost '1', which in turn is checked off by replacing it with another symbol, say *y*. After that, we go to left again to the leftmost 0, replace it with an *x*, then move to the leftmost '1', and replace it with *y*, and so on.

Travelling back and forth the same way, we match each 0 with a corresponding 1. If after some time no 0's and 1's remain, then the string must be in L.

Working on this, the solution is

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

$F = \{q_4\}$

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

$\Gamma = \{0, 1, x, y, B\}$

The transitions can be broken into several parts. The set

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

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

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

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

replaces the leftmost 0 with an *x*, then causes the read-write head to travel right to the first 1, replacing it with a *y*. When the *y* is written, the machine enters a state q_2, indicating that an 0 has been successfully paired with a 1.

The next set of transitions reverses the direction until an x is encountered, repositions the read-write head over the left most 0, and returns control to initial state.

$\delta(q_2, y) = (q_2, y, L)$

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

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

We are now back in the initial state q_0, ready to deal with the next 0 and 1.

After one pass through this part of the computation, the machine will have carried partial computation.

i.e. $q_0 00....011.....1 \overset{*}{\vdash} xq_0 0.....0y1.....1,$

So that single 0 has been matched with a single 1. After 2 passes, we will have completed the partial computation.

$q_0 00....011.....1 \overset{*}{\vdash} xxq_0 0..... 0yy1.....1$ and so on, indicating that the matching is carried out properly.

When the input is a string 0^n1^n, the rewriting continues this way, stopping only when there are no more 0's to be erased, when looking for the leftmost 0, the read-write head travels with the machine in state q_2. When an x is encountered, the direction is reversed, to get the 0. But now, instead of finding an 0, it will find a 1. To terminate, a final check is made to see if all 0's and 1's is replaced. This is done by

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

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

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

If the input string is not in the language, the computation will halt in non-final state.

If the input string is in the language, it will result in final state.

Computation of 0011.

$q_0 0011 \vdash xq_1 011 \vdash x0q_1 11 \vdash xq_2 0y1$

$\vdash q_2 x0y1 \vdash xq_0 0y1 \vdash xxq_1 y1$

$\vdash xxyq_1 1 \vdash xxq_2 yy \vdash xq_2 xyy$

$\vdash xxq_0 yy \vdash xxyq_3 y \vdash xxyyq_3 B$

$\vdash xxyyBq_4 B.$

8.4. DESIGN OF TURING MACHINES

The guidelines for designing a turing machine is

(i) The fundamental objective in scanning a symbol by R/W head is to 'know' what to do in the future. The machine must remember the past symbols scanned. The turing machine can remember this by going to the next unique state.

(ii) The number of states must be minimised. This can be achieved by changing the states only when there is a change in the written symbol or when there is a change in the movement of R/W head.

Example 8.10

Design a Turing machine to recognise all strings consisting of even number of 1's.

Solution :

The construction is made by defining the following moves

(i) q_1 is in initial state. M enters state q_2 on scanning 1 and writes *b*.

(ii) If M is in state q_2 and scans 1, it enters q_1 and writes *b*.

(iii) q_1 is the only accepting state.

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

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

$\Gamma = \{1, B\}$

$F = \{q_1\}$

and δ is given by

	Present State	Tape Symbol (1)
Start →	q_1	Bq_2R
	q_2	Bq_1R

(a) Computing the sequence of 11

$q_1 11 \vdash B q_2 1 \vdash BB q_1$

As q_1 is accepting state, it is accepted.

(b) Computing the sequence of 111

$q_1111 \vdash B q_2 11 \vdash BB q_1 1 \vdash BBB q_2$

As q_2 is not acceting state, it is not accepted.

Example 8.11

Design a Turing machine M to recognise the language.

$L = \{1^n\ 2^n\ 3^n : n \geq 1\}$

Solution :

Let us evolve the procedure for processing of input string 112233.

Step 1 :

q_1 is the initial state. The R/W head scans the leftmost 1, replaces 1 by *b*, and moves to the right. M enters q_2.

Step 2 :

On scanning the leftmost 2, the R/W head scan replaces 2 by *b*, and moves to the right, M enters q_3.

Step 3 :

On scanning the leftmost 3, the R/W head replaces 3 by *b*, and moves to the right. M enters q_4.

Step 4 :

After scanning the rightmost 3, the R/W heads moves to the left until it finds the leftmost 1. As a result, the leftmost 1, 2 and 3 are replaced by *b*.

Step 5 :

Step 1–4 are repeated until all 1's, 2's and 3's are replaced by blanks.

$q_1112233 \vdash bq_212233 \vdash b1q_22233$

$\vdash b1bq_3233 \vdash b1b2q_333$

$\vdash b1b2bq_43 \vdash b1b2q_5b3$

$\vdash b1bq_52b3 \vdash b1q_5b2b3$

$\vdash bq_51b2b3 \vdash q_6b1b2b3$

$\vdash bq_11b2b3 \vdash bbq_2b2b3$

$\vdash bbbq_22b3 \vdash bbbbq_3b3$

$\vdash bbbbbbq_4b \vdash bbbbbq_7bb$

Thus

$$q_1 112233 \vdash_M^* q_7 bbbbbb$$

Since q_7 is an accepting state. The string is accepted.

	Present States	Tape Symbols			
		1	2	3	b
Start →	q_1	bRq_2			bRq_1
	q_2	$1Rq_2$	bRq_3		bRq_2
	q_3		$2Rq_3$	bRq_4	bRq_3
	q_4			$3Lq_5$	bLq_7
	q_5	$1Lq_6$	$2Lq_5$		bLq_5
	q_6	$1Lq_6$			bRq_1
	((q7))				

8.5 COMPUTABLE LANGUAGES AND FUNCTIONS

A language that is accepted by a turing machine is said to be recursively enumerable. The turing machine may be viewed as a computer of functions from positive integers to positive integers. Integers are represented in unary fashion.

The integers i≥0 is represented by the string 0^i. If a function has k arguments, i_1, i_2, ...i_k, then these integers are initially placed on the tape separated by 1's as $0^{i_1}10^{i_2}10^{i_3}$..... 10^{i_k}. So it is possible that a turing machine can be considered as a computer of functions from integer to integer.

8.5.1 Definition of Recursive Function

A function $f : N \rightarrow N$ is said to be a computable function of k arguments, if there exists a Turing Machine M halts with a tape consisting of 0^m for some m, where

$f(i_1, i_2, ...i_k) = m$.

If $f(i_1, i_2, ... i_k)$ is defined for all $i_1, i_2, ... i_k$, then we say, f is a *total recursive function*. A function $f(i_1, i_2,.....i_k)$ computed by a TM is called *partial recursive function*.

Example 8.12

Construct a TM for successive function ?

$f : N \rightarrow N$, $f(x) = x+1$.

Solution :

Assume that the input is encoded in UNARY form.

Let M = (Q, Σ, Γ, δ, q_0, B, F)

where

Q = $\{q_0, q_1\}$ q_0=start state q_1=final state

$\Sigma = \{0\}$

$\Gamma = \{0, B\}$

F = $\{q_1\}$

The transition function δ can be defined as

States	Tape Symbols	
	0	B
q_0	$(q_0, 0, R)$	$(q_1, 0, R)$
q_1	–	–

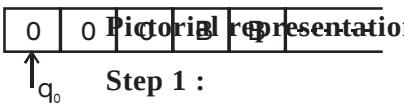
Let us consider the input x=3, This is encoded as 000.

$(q_0, \underline{0}00B) \vdash (q_0, 0\underline{0}0B) \vdash (q_0, 00\underline{0}B)$

$\vdash (q_0, 000\underline{B}) \vdash (q_1, 000\underline{0}B)$

The machine halts in an accepting state q_1 by computing the successive of x.

Pictorial representation :

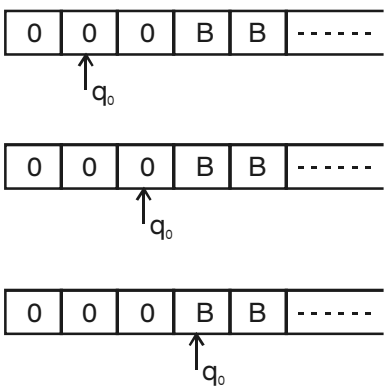


Step 1 :

The input string 000B is in input tape.

Step 2 :

On state q_0 and current input symbol 0, the tape head moves right until, it sees the Blank symbol. $\delta(q_0,0)=(q_0,0,R)$



on $\delta(q_0,B) = (q_1, 0, R)$

The first blank symbol is replaced by 0 and the state is changed to q_1 which is acceptance state

8.6 PROGRAMMING TECHNIQUES OF TURING MACHINES

To describe complicated turing machine constructions we need some "higher-level" conceptual tools. The principal ones are discussed below.

8.6.1 Storage in the Finite Control

(i) The finite control can be used to hold the finite amount of information.

(ii) It is considered as a pair of elements, like (q_0,a), where one exercising control and second component stores a symbol in the finite control.

Example 8.13

Consider a turing machine M that looks at the first input symbol, records it in its finite control, and checks that the symbol does not appear elsewhere on its input. Thus M accepts the language 01* + 10*.

Let $M = (Q, \{0,1\}, \{0,1,B\}, \delta, \{q_0,B\},B,F)$

where

$Q = \{q_0,q_1\} \times \{0,1,B\}$

$= ([q_0,0], [q_0,1], [q_0,B], [q_1,0], [q_1,1], [q_1,B])$

$F = \{[q_1,B]\}$

we define δ as follows

(i) (a) $\delta([q_0,B],0) = ([q_1,0], 0,R)$

(b) $\delta([q_0,B],1) = ([q_1,1], 1, R)$

i.e., q_0 is initial state and M moves right. The first components of M's state becomes q_1, and the first symbol seen is stored in the second component.

(ii) (a) $\delta([q_1,0],1) = ([q_1,0], 1, R)$

(b) $\delta([q_1,1],0) = ([q_1,1], 0, R)$

If M has a 0 stored and sees a 1 or viceversa, then M continues to move to the right.

(iii) (a) $\delta([q_1,0],B) = ([q_1,B],0,L)$

(b) $\delta([q_1,1], B) = ([q_1,B], 1, L)$

M enters the final state $[q_1,B]$ if it reaches a blank symbol without having first encountered a second copy of the leftmost symbol.

For state $[q_1, 0]$ and symbol 0 (or) for state $[q_1,1]$ and symbol 1, δ is not defined.

8.6.2 Multiple Tracks

We can imagine the tape of the turing machine is divided into multiple tracks, for any finite *k*. The symbols on the tape are considered *k*-tuples, one component for each track.

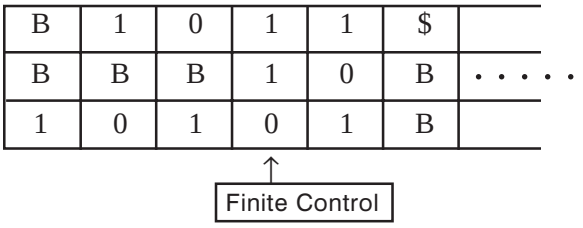


Figure 8.4 A three track turing machine

8.6.3 Checking off Symbols

It is a useful trick for visualing how a TM recognizes languages defined by repeated strings, such as

$\{ww|w \text{ in } \Sigma^*\}$, $\{wcy|w \text{ and } y \text{ in } \Sigma^*, w \neq y\}$ (or) $\{ww^R|w \text{ in } \Sigma^*\}$

It is also useful when lengths of substrings must be compared. For example:

$\{a^ib^i : i \geq 1\}$ (or) $\{a^ib^jc^k : i \neq j, j \neq k\}$

We introduce an extra track on the tape that holds a Blank B (or) ✓ and other track to hold data as usual.

B	B	B	B	✓	✓	.....	extra Track
a	*a*	*b*	*c*	*a*	*b*	.....	

Example 8.14

Consider a turing machine M = (Q, Σ, Γ, δ, q_0, B, F) for the language L = $\{wcw|w \in \{a, b\}^+\}$

a) Q = $\{[q,d] \mid q = q_1\ q_2 \ldots q_9$ and $d = a, b$ or B$\}$

b) $\Sigma = \{[B, d] \mid d = a, b$ or $c\}$

c) $\Gamma = \{[x, d] \mid x =$ B or ✓ and $d = a, b, c$ or B$\}$

d) $q_0 = [q_1, B]$

e) $F = \{[q_9, B]\}$

f) B = [B, B]

g) δ is defined for $d = a$ or b and $e = a$ or b.

δ is defined as

1) $\delta([q_1, B], [B, d]) = ([q_2, d], [\checkmark, d], R)$

   M checks the symbol scanned on the tape, stores the symbol in the finite control and moves right.

2) $\delta(q_2, d], [B, e]) = ([q_2, d], [B, e], R)$

   M continues to move right over unchecked symbols, looking for *e*.

3) $\delta([q_2,d], [B, c]\} = ([q_3, d], [B, c], R)$

   On finding *c*, M enters a state with first component q_3.

4) $\delta([q_3,d], [\checkmark, e]) = ([q_3,d], [\checkmark, e], R)$

   M moves right over checked symbols.

5) $\delta([q_3,d], [B, d]) = ([q_4,B], [\checkmark, d], L)$

   M encounters an unchecked symbol. If the unchecked symbol matches the symbol stored in the finite control, M checks it and begins moving left. If the symbols disagree, M has no next move and so halts without accepting. M also halts if in state q_3 it reaches [B, B] before finding an unchecked symbol.

6) $\delta([q_4,B], [\checkmark, d]) = ([q_4,B], [\checkmark, d], L)$

   M moves left over checked symbols.

7) $\delta([q_4,B], [B, c]) = ([q_5,B], [B, c], L)$

   M encounters the symbol *c*.

8) $\delta([q_5,B], [B, d]) = ([q_6,B], [B, d], L)$

   If the symbol immediately to the left of *c* is unchecked, M proceeds left to find the rightmost checked symbol.

9) $\delta([q_6,B], [B, d]) = ([q_6,B], [B, d], L)$

   M proceeds left.

10) $\delta([q_6,B], [\checkmark, d]) = ([q_1,B], [\checkmark, d], R)$

    M encounters a checked symbol and moves right to pick up another symbol for comparison. The first component of state becomes q_1 again.

11) $\delta([q_5,B], [\checkmark, d]) = ([q_7,B], [\checkmark, d], R)$

M will be in state $[q_5, B]$ immediately after crossing c moving left. (See rule 7). If a checked symbol appears immediately to the left of *c*, all symbols to the left of *c* have been checked. M must test whether all symbols to the right have been checked. If so, they must have compared properly with the symbols to the left of *c*, so M will accept.

12) $\delta([q_7,B], [B, c]) = ([q_8,B], [B, c], R)$

M moves right over *c*.

13) $\delta([q_8,B], [\checkmark, d]) = ([q_8,B], [\checkmark, d], R)$

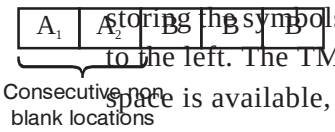
M moves to the right over checked symbols.

14) $\delta([q_8,B], [B, B]) = ([q_9,B], [\checkmark, B], L)$

If M finds [B, B], the blank, it halts and accepts. If M finds an unchecked symbol when its first component of state is q_8, it halts without accepting.

8.6.4 Shifting Over

The turing machine can make space on its tape by shifting all nonblank symbols a finite number of cells to the right. To do so, the tape head makes an excursion to the right, repeatedly storing the symbols read in its finite control and replacing them with symbols read from cells to the left. The TM can then return to the vacated cells and print symbols of its choosing. If space is available, it can push blocks of symbols left in a similar manner.



Shifting consecutive non blank locations to the right for a finite number is called shifting over. Here finite control is used as the temporary storage.

Example 8.15

We construct part of a Turing machine, M = (Q, Σ, Γ, δ, q_0, B, F), which may occasionally have a need to shift nonblank symbols two cells to the right. We suppose that M's tape does not contain blanks between nonblanks, so when it reaches a blank it knows to stop the shifting process. Let Q contain states of the form $[q, A_1, A_2]$ for $q = q_1$ or q_2, and A_1 and A_2 in Γ. Let X be a special symbol not used by M except in the shifting process. M starts the shifting process in state $[q_1, B, B]$. The relevant portions of the function δ are as follows.

1) $\delta([q_1, B, B], A_1) = ([q_1, B, A_1], X, R)$ for A_1 in $\Gamma - \{B, X\}$

M stores the first symbol read in the third component of its state, X is printed on the cell scanned, and M moves to the right.

2) $\delta([q_1, B, A_1], A_2) = ([q_1, A_1, A_2], X, R)$ for A_1 and A_2 in $\Gamma - \{B, X\}$

M shifts the symbol in the third component to the second component, stores the symbol being read in the third component, prints an X, and moves right.

3) $\delta([q_1, A_1, A_2], A_3) = ([q_1, A_2, A_3], A_1, R)$ for A_1, A_2 and A_3 in $\Gamma - \{B, X\}$

M now repeatedly reads a symbol A_3, stores it in the third component of state, shifts the symbol previously in the third component, A_2, to the second component, deposits the previous second component, A_1, on the cell scanned, and moves right. Thus a symbol will be deposited two cells to the right of its original position.

4) $\delta([q_1, A_1, A_2], B) = ([q_1, A_2, B], A_1, R)$ for A_1 and A_2 in $\Gamma - \{B, X\}$

When a blank is seen on the tape, the stored symbols are deposited on the tape.

5) $\delta([q_1, A_1, B], B) = ([q_2, B, B], A_1, L)$

After all symbols have been deposited, M sets the first component of state to q_2 and moves left to find an X, which marks the rightmost vacated cell.

6) $\delta([q_2, B, B], A) = ([q_2, B, B], A, L)$ for A in $\Gamma - \{B, X\}$

M moves left until an X is found. When X is found, M transfers to a state that we have assumed exists in Q and resumes its other functions.

8.6.5 Subroutines

As with programs, a "modular" or "top-down" design is faciliated if we use subroutines to define elementary processes. A Turing machine can simulate any type of subroutine found in programming languages, including recursive producers and any of the known parameter-passing mechanisms. We shall here describe only the use of parameterless, nonrecursive subroutines, but even are quite powerful tools.

The general idea is to write part of a TM program to serve as a subroutine; it will have a designated initial state and a designated return state which temporarily has no move and which will be used to effect a return to the calling routine. To design a TM that "calls" the subroutine, a new set of states for the subroutine is made, and a move from the return state is specified. The call is effected by entering the initial state for the subroutine, and the return is effected by the move from the return state.

Example 8.16

Design a TM that can compute multiplication with subroutine "copy".

Solution : **(Apr/May 2004)**

The design of a TM M to implement the total recursive function "multiplication" is given below. M starts with 0^m10^n on its tape and ends with 0^{mn} surrounded by blanks. The general idea is to place a 1 after 0^m10^n and then copy the block of n 0's onto the right end m times, each time erasing one of the m 0's. The result is 10^n10^{mn}. Finally the prefix 10^n1 is erased, leaving 0^{mn}.

Step 1 :

The initial ID $q_0 0^m 10^n$ to $B0^{m-1}1q_1 0^n 1$. The corresponding set of rules are:

$\delta(q_0,0) = (q_6, B, R)$

$\delta(q_6, 0) = (q_6, 0, R)$

$\delta(q_6, 1) = (q_1, 1, R)$

Step 2 :

The heart of the algorithm is a subroutine COPY, which begins in an ID $0^m1q_1 0^n 10^i$ and eventually enters an ID $0^m1q_5 0^n 10^{i+n}$. COPY is defined in Figure 8.5. In state q_1, on seeing a 0, *M* changes it to a 2 and enters state q_2. In state q_2, M moves right, to the next blank, deposits the 0, and starts left in state q_3. In state q_3, M moves left to a 2. On reaching a 2, state q_1 is entered and the process repeats until the l is encountered, signaling that the copying process is complete. State q_4 is used to convert the 2's back to 0's, and the subroutine halts in q_5.

States	Inputs			
	0	1	2	B
q_1	$(q_2, 2, R)$	$(q_4, 1, L)$		
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$		$(q_3, 0, L)$
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	$(q_1, 2, R)$	
q_4		$(q_5, 1, R)$	$(q_4, 0, L)$	

Figure 8.5 δ for subroutine COPY.

Step 3 : Termination condition

Additional states are needed to convert an ID $B^i0^{m-i}1q_5 0^n 10^{ni}$ to $B^{i+1}0^{m-i-1}1q_1 0^n 10^{ni}$, which restarts COPY, and to check whether $i = m$, that is, all m 0's have been erased. In the case that $i = m$, the leading 10^n1 is erased and the computation halts in state q_{12}. These moves are shown in Figure 8.6.

States	Inputs			
	0	1	2	B
q_5	$(q_7,0,L)$			
q_7		$(q_8,1,L)$		
q_8	$(q_9,0,L)$			(q_{10},B,R)
q_9	$(q_9,0,L)$			$(q_0,B, R$
q_{10}		(q_{11},B,R)		
q_{11}	(q_{11},B,R)	(q_{12},B,R)		

Figure 8.6 Additional moves for TM performing multiplication.

For input string $0^2 10^2 1$ using ID,

$\delta(q_0, 001001)$ $\vdash$ $Bq_6 01001B$ (rule 1)

$\vdash$ $B0q_6 1001B$ (rule 2)

$\vdash$ $B01q_1 001B$ (rule 3)

$\vdash$ $B012q_2 01B$ (rule 4)

$\vdash$ $B0120q_2 1B$ (rule 6)

$\vdash$ $B01201q_2 B$ (rule 7)

$\vdash$ $B0120q_3 10$ (rule 8)

$\vdash$ $B012q_3 010$ (rule 10)

$\vdash$ $B01q_3 2010$ (rule 9)

$\vdash$ $B012q_1 010$ (rule 11)

$\vdash$ $B0122q_2 10$ (rule 4)

$\vdash$ $B01221q_2 0$ (rule 7)

$\vdash$ $B012210q_2 B$ (rule 6)

$\vdash$ $B01221q_3 00$ (rule 8)

$\vdash$ $B0122q_3 100$ (rule 9)

$\vdash$ $B012q_3 2100$ (rule 10)

$\vdash$ $B0122q_1 100$ (rule 11)

$\vdash$ $B012q_4 2100$ (rule 5)

$\vdash$ $B01q_4 20100$ (rule 13)

$\vdash$ $B0q_4 100100$ (rule 13)

$\vdash$ $B01q_5 00100$ (rule 12)

$\vdash$ $B0q_7 100100$ (rule 14)

$\vdash$ B$q_8$0100100 (rule 15)
$\vdash$ q_9B0100100 (rule 16)
$\vdash$ B$q_0$0100100 (rule 18)
$\vdash$ BB$q_6$100100 (rule 1)
$\vdash$ BB1$q_1$00100 (rule 3)
$\vdash$ BB12$q_2$0100 (rule 4)
$\vdash$ BB120$q_2$100 (rule 6)
$\vdash$ BB120$q_2$100 (rule 6)
$\vdash$ BB1201$q_2$00 (rule 7)
$\vdash$ BB12010$q_2$0 (rule 6)
$\vdash$ BB120100q_2B (rule 6)
$\vdash$ BB12010$q_3$00 (rule 8)
$\vdash$ BB1201$q_3$000 (rule 9)
$\vdash$ BB120$q_3$1000 (rule 9)
$\vdash$ BB12$q_3$01000 (rule 10)
$\vdash$ BB1$q_3$201000 (rule 9)
$\vdash$ BB12$q_1$01000 (rule 11)
$\vdash$ BB122$q_2$1000 (rule 4)
$\vdash$ BB1221$q_2$000 (rule 7)
$\vdash$ BB12210$q_2$00 (rule 6)
$\vdash$ BB122100$q_2$0 (rule 6)
$\vdash$ BB1221000q_2B (rule 6)
$\vdash$ BB122100$q_3$00 (rule 8)
$\vdash$ BB12210$q_3$000 (rule 9)
$\vdash$ BB1221$q_3$0000 (rule 9)
$\vdash$ BB122$q_3$10000 (rule 9)
$\vdash$ BB12$q_3$210000 (rule 10)
$\vdash$ BB122$q_1$10000 (rule 11)
$\vdash$ BB12$q_4$210000 (rule 5)
$\vdash$ BB1$q_4$2010000 (rule 13)
$\vdash$ BB$q_4$10010000 (rule 13)
$\vdash$ BB1$q_5$0010000 (rule 12)
$\vdash$ BB$q_7$10010000 (rule 14)

$\vdash$ Bq_8B10010000 (rule 15)

$\vdash$ BBq_{10}10010000 (rule 17)

$\vdash$ BBBq_{11}0010000 (rule 20)

$\vdash$ BBBBq_{11}010000 (rule 21)

$\vdash$ BBBBBq_{11}10000 (rule 21)

$\vdash$ BBBBBBq_{12}0000 (rule 22)

8.7 MODIFICATIONS OF TURING MACHINES

One reason for the acceptance of the Turing machine as a general model of a computation is that the model with which we have been dealing is equivalent to many modified versions that would seem off-hand to have increased computing power. In this section we give informal proofs of some of these equivalence theorems.

The different types of modifications performed on the TM is :

(i) Two-way infinite tape

(ii) Multitape turing machine

(iii) Nondeterministic turing machines

(iv) Multidimensional turing machines

(v) Multihead turing machines

(vi) Off-line turing machines

8.7.1 Two-way Infinite Tape

A Turing machine with a two-way infinite tape is denoted by M = $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$, as in the original model. As its name implies, the tape is infinite to the left as well as to the right. We denote an ID of such a device as for the one-way infinite TM. We imagine, however, that there is an infinity of blank cells both to the left. and right of the current nonblank portion of the tape.

The relation $\vdash_M$, which relates two ID's if the ID on the right is obtained from the one on the left by a single move, is defined as for the original model with the exception that if $\delta(q,X) = (p, Y, L)$, then $qX\alpha \vdash_M pBY\alpha$ (in the original model, no move could be made), and if $\delta(q, X) = (p, B, R)$, then $qX\alpha \vdash p\alpha$ (in the original, the; B would appear to the left of p).

The initial ID is q_0w. While there was a left end to the tape in the original model, there is no left end of the tape for the Turing machine to "fall off," so it can proceed left as far as it wishes. The relation $\vdash_M$, as usual, relates two ID's if the one on the right can be obtained from the one on the left by some number of moves.

Theorem

L is recognized by a Turing machine with a two-way infinite tape if and only if it is recognized by a TM with a one-way infinite tape.

Proof

The proof that a TM with a two-way infinite tape can simulate a TM with a one-way infinite tape is easy. The former marks the cell to the left of its initial head position and then simulates the latter. If during the simulation the marked cell is reached, the simulation terminates without acceptance.

Conversely, let $M_2 = (Q_2, \Sigma_2, \delta_2, B, F_2)$ be a TM with a two-way infinite tape. We construct M_1, a Turing machine simulating M_2 and having a tape that is infinite to the right only. M_1 will have two tracks, one to represent the cells of M_2's tape to the right of, and including, the tape cell initially scanned, the other to represent, in reverse order, the cells to the left of the initial cell. The relationship between the tapes of M_2 and M_1 is shown in below Figure 8.7 with the initial cell of M_2 numbered 0, the cells to the right 1,2,..., and the cells to the left –1, –2,.....

The first cell of M_1's tape holds the symbol ϕ in the lower track, indicating that it is the leftmost cell. The finite control of M_1 tells whether M_2 would be scanning a symbol appearing on the upper or on the lower track of M_1.

(a)

....	A_{-5}	A_{-4}	A_{-3}	A_{-2}	A_{-1}	A_0	A_1	A_2	A_3	A_4	A_5	....

(b)

A_0	A_1	A_2	A_3	A_4	A_5
ϕ	A_{-1}	A_{-2}	A_{-3}	A_{-4}	A_{-5}

Figure 8.7. (a) Tape of M_2. (b) Tape,of M_1.

It should be fairly evident that M_1 can be constructed to simulate M_2, in the sense that while M_2 is to the right of the initial position of its input head, M_1 works on the upper track. While M_2 is to the left of its initial tape head position, M_2 works on its lower track, moving in the direction opposite to the direction in which M_2 moves. The input symbols of M_1 are symbols with a blank on the lower track and in input symbol of M_2 on the upper track. Such a symbol can be rectified with the corresponding input symbol of M_2. B is identified with [*B, B*].

We now give a formal construction of $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, B, F_1)$. The states Q_1, of M_1 are all objects of the form [*q*, U] or [*q, D*], where *q* is in Q_2, plus the symbol q_1. Note that the second component will indicate whether M_1 is working on the upper (U for up) or lower (D for down) track. The tape symbols in Γ_1 are all objects of the form [*X, Y*], where *X* and *Y* are in Γ_2 . In addition, *Y* may be ϕ, a symbol not in Γ_2. Σ_1 consists of all symbols [*a*, B], where *a* is in Σ_2, F_1 is $\{[q, U]\ [q, D] \mid q \text{ is in } F_2\}$. We define δ_1, as follows:

1) For each *a* in $\Sigma_2 \cup \{B\}$

$\delta_1(q_1,[a,B]) = ([q,U], [X, \phi], R)$ if $\delta_2(q_2,a)=(q,X,R)$

If M_2 moves right on its first move, M_1 prints ϕ in the lower track to mark the end of tape, sets its second component of state to U, and moves right. The first component of M_1's state holds the state of M_2. On the upper track, M_1 prints the symbol X that is printed by M_2.

2) For each a in $\sum_2 \cup \{B\}$,

$\delta_1(q_1, [a, B]) = ([q, D], [X, \phi], R)$ if $\delta_2(q_2, a) = (q, X, L)$.

If M_2 moves left on its first move, M_1 records the next state of M_2 and the symbol printed by M_2 as in (1) but sets the second component of its state to D and moves right. Again, ϕ is printed in the lower track to mark the left end of the tape.

3) For each $[X, Y]$ in Γ_1, with $Y \neq \phi$, and $A = L$ or R,

$\delta_1([q, U], [X, Y]) = ([p, U], [Z, Y], A)$ if $\delta_2(q, X)=(p, Z, A)$.

M_1 simulates M_2 on the upper track.

4) For each $[X, Y]$ in Γ_1, with $Y \neq \phi$,

$\delta_1([q, D], [X, Y], =([p,D], [X, Z], A)$ if $\delta_2(q, Y)=(p, Z, \overline{A})$.

Here A is L if is R, and A is R if is L. M_1 simulates M_2 on the lowertrack of M_1. The direction of head motion of M_1 is opposite to that of M_2.

5) $\delta_1([q, U], [X, \phi]) = \delta_1([q, D], [X,\phi])$

$= ([p, C], [Y, \phi], R)$ if $\delta_2(q, X) = (p, Y, A)$

Here $C=U$ if $A=R$, and $C=D$ if $A = L$. M_1 simulates a move of M_2 on the cell initially scanned by M_2. M_1 next works on the upper or lower track, depending on the direction in which M_2 moves. M_1 will always move right in this situation.

8.7.2 Multitape Turing Machines

A multitape Turing machine is shown in the Figure 8.8. It consists of a finite control with k tape heads and k tapes; each tape is infinite in both directions. On a single move, depending on the state of the finite control and the symbol scanned by each of the tape heads, the machine can:

(i) change state;

(ii) print a new symbol on each of the cells scanned by its tape heads;

(iii) move each of its tape heads, independently, one cell to the left or right, or keep it stationary.

Initially, the input appears on the first tape, and the other tapes are blank. We shall not define the device more formally, as the formalism is cumbersome and a straightforward generalization of the notation for single-tape TM's.

Theorem

If a language *L* is accepted by a multitape Turing machine, it is accepted by a single-tape Turing machine.

Figure 8.8. Multitape turing machine

Proof

Let *L* be accepted *by* M_1, a TM with *k* tapes. We can construct M_2, a one-tape TM with 2*k* tracks, two tracks for each of M_1's tapes. One track records the contents of the corresponding tape of M_1, and the other is blank, except for a marker in the cell that holds the symbol scanned by the corresponding head of M_1. The arrangement is illustrated in figure 8.9. The finite control of M_2 stores the state *of* M_1, along with a count of the number of head markers to the right of M_2's tape head.

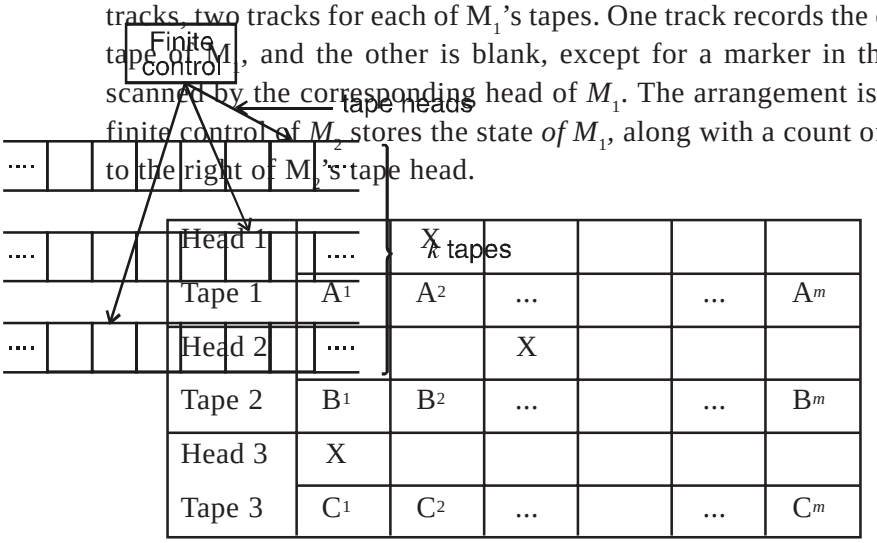


Figure 8.9. Simulation of three tapes by one

Each move of M^1 is simulated by a sweep from left to right and then from right to left by the tape head of M^2. Initially, M^2's head is at the leftmost cell containing a head marker. To simulate a move of M^1, M^2 sweeps right, visiting each of the celis with head markers and recording the symbol scanned by each head of M^1. When M^2 crosses a head marker, it must update the count of head markers to its right. When no more head markers are to the right, M^2

has seen the symbols scanned by each of M^1's heads, so M^2 has enough information to determine the move of M^1. Now M^2 makes a pass left, until it reaches the leftmost head marker. The count of markers to the right enables M^2 to tell when it has gone for enough. As M^2 passes each head marker on the leftward pass, it updates the tape symbol of M^1 "scanned" by that head marker, and it moves the head marker one symbol left or right to simulate the move of M^1. Finally, M^2 changes the state of M^1 recorded in M^2's control to complete the simulation of one move of M^1. If the new state of M^1 is accepting, then M^2 accepts.

Example 8.17

The language $L = \{ww_R \mid w \text{ in } (0 + 1)^*\}$ can be recognized on a single-tape TM by moving the tape head back and forth on the input, checking symbols from both ends, and comparing them.

To recognize L with a two-tape TM, the input is copied onto the second tape. The input on one tape is compared with the reversal on the other tape by moving 'the heads in opposite directions, and the length of the input checked to make sure it even.

Note that the number of moves used to recognize L by the one-tape machine is approximately the square of the input length, while with a two-tape machine, time proportional to the input length is sufficient.

8.8 SOLVED PROBLEMS

1. Construct a TM to compute the function $f(w) = w^R$, where $w \in \{a,b\}^+$

Solution :

Let M = {Q, Σ, Γ, δ, q_0, B, F} be a TM

where Q = {q_0, q_1,..... q_{10}}

Σ = {a, b}

Γ = {a, b, B, X, Y}

F = {q_{10}}

q_0 = Initial state

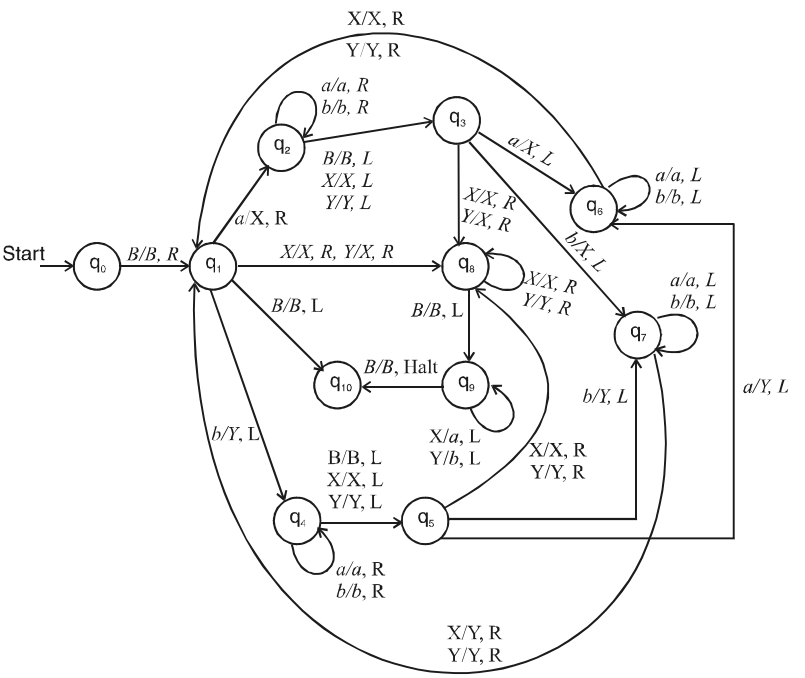
The state q_1 changes a and b in to X and Y respectively and remembers it (by going to state q_2 in case of an 'a', and q_4 in the case of a 'b') as it moves the tape head to the right. When the TM arrives at q_3 or q_5, if there is a lower case symbol on the right corresponding to the one on the left, the TM sees it, remembers it (by going to either q_6 (or) q_7) and changes it to the symbol it remembers from the left. For even length strings, the TM will return to q_1 and if there is no a (or) b change, then the machine move to q_8 and the final phase of processing.

In the odd length string the machine will move to either q_3 or q_5. Finally it moves back to the left to change all upper case symbols of X, Y to a, b respectively.

Transition table

States	a	b	B	X	Y
q_0	–	–	(q_1, B, R)	–	–
q_1	(q_2, X, R)	(q_4, Y, R)	(q_{10}, B, L)	(q_8, X, R)	(q_8, X, R)
q_2	(q_2, a, R)	(q_2, b, R)	(q_3, B, L)	(q_3, X, L)	(q_3, Y, L)
q_3	(q_6, X, L)	(q_7, X, L)	–	(q_8, X, R)	(q_8, X, R)
q_4	(q_4, a, R)	(q_4, b, R)	(q_5, B, L)	(q_5, X, L)	(q_5, Y, L)
q_5	(q_6, Y, L)	(q_7, Y, L)	–	(q_8, X, R)	(q_8, Y, R)
q_6	(q_6, a, L)	(q_6, b, L)	–	(q_1, X, R)	(q_1, X, R)
q_7	(q_7, a, L)	(q_7, b, L)	–	(q_1, Y, R)	(q_1, Y, R)
q_8	–	–	(q_9, B, L)	(q_8, X, R)	(q_8, Y, R)
q_9	–	–	(q_{10}, B, H)	(q_9, a, L)	(q_9, b, L)

Transition diagram



Let w = *abb*, computation is

q_0B*abb* $\vdash$ Bq_1*abb* $\vdash$ BXq_2*bb* $\vdash$ BX*bq*$_2$*b* $\vdash$ BX*bbq*$_2$B

$\vdash$ BX*bq*$_3$*b* $\vdash$ B*X*q$_7$*b*X $\vdash$ *Bq*$_7$X*b*X $\vdash$ *Byq*$_1$*b*X

$\vdash$ BYYq_4X $\vdash$ BYq_5YX $\vdash$ BYYq_8X $\vdash$ BYYXq_8B

$\vdash$ BYYq_9X $\vdash$ BYq_9B*a* $\vdash$ Bq_9Y*ba* $\vdash$ q_9B*bba*

$\vdash$ Bq_{10}*bba*

For string of equal length, let *w* = *baba*, the computation is,

q_0B*baba* $\vdash$ Bq_1*baba* $\vdash$ BYq_4*aba* $\vdash$ BY*aq*$_4$*ba*

$\vdash$ BY*abq*$_4$*a* $\vdash$ BY*abaq*$_4$B $\vdash$ BY*abq*$_5$*a*

$\vdash$ BY*aq*$_6$*b*Y $\vdash$ BYq_6*ab*Y $\vdash$ Bq_6Y*ab*Y

$\vdash$ BXXq_3*b*Y $\vdash$ BXq_7XXY $\vdash$ BXYq_1XY

$\vdash$ BXYXq_8Y $\vdash$ BXYXYq_8B $\vdash$ BXYXq_9Y

$\vdash^*$ Bq_9*abab* $\vdash$ q_{10}B*abab*

2. The addition function $f(m, n) = m + n$ is computable by a Turing Machine.

Solution :

Let the input $w = 0^m0^n$

ie. $m = 0^m$ and $n = 0^n$

$\therefore$ the result $w = 0^m\ 0^n = 0^{m+n}$

Let M = (Q, Σ, Γ, δ, q_1, B, F) be a TM

with Q = {q_1, q_2, q_3, q_4}

Σ = {0} = Input alphabet

Γ = {0, 1, B} = Tape alphabet

F = {q_4}

and δ is defined as follows

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

i.e. on seeing the 1st zero, it is made as Blank and proceed further.

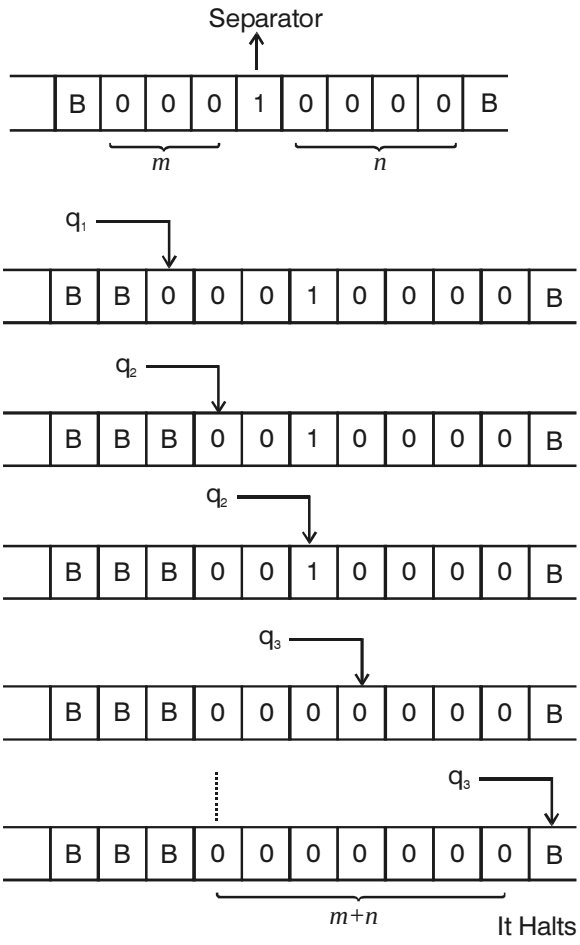
$\delta(q_2, 0) = \{q_2, 0, R\}$

$\delta(q_2, 1) = \{q_3, 0, R\}$

$\delta(q_3, 0) = \{q_3, 0, R\}$

$\delta(q_3, B) = \{q_4, B, H\}$ H = Halt

Example : To compute $w = 0^3\ 1\ 0^4$, the corresponding pictorial representation is:



ie. computation is as follows:

$q_1 0^3 10^4 \vdash B q_2 0^2 10^4 \vdash^* B00 q_2 10^4 \vdash B000 q_3 0^4$

$\vdash^* B0^3 0^4 q_3 B \vdash 0^7 q_4$

3. Design a TM that can compute proper subtraction. That is $m–n$.

$m–n$ if $m > n$

$m–n = 0$ if $m < n$ **(Nov/Dec 2003)**

Solution :

Let M = (Q, Σ, Γ, δ, q_0, B, F) be the required TM where

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

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

$\Gamma = \{0, 1, B\}$

$F = \{q_6\}$

and δ is defined in the transition table

Let $w = 0^m\ 1\ 0^n$ be the input string. The output of the machine should halt 0^{m-n} on this tape.

The construction is as follows:

M repeatedly replaces its leading 0 by B, then searches right for a 1 followed by a 0 and changes the 0 to 1. Next M moves left until it finds a blank and repeats the cycle. The process comes to an end if :

(a) Searching right for a 0, M finds a Blank. In that case n 0's in $0^m\ 1\ 0^n$ have all been changed to 1's and n+1 of the m 0's have been changed to B. The n 1's are changed to Blank and one of the blank in the m is changed to zero, leaving $m–n$ 0's on the tape.

(b) M cannot find a 0 to change to B, because the first m 0's already have been changed. Then $n \geq m$, so $m–n$=0, M replaces all remaining 1's and 0's by B.

q_0 is an initial state.

q_1 is used to search for 1 in the right

q_2 is used to search for '0' on right. If '0' is found, then change 0 to 1.

q_3 is used to move left to find a blank and then enters q_0 to repeat the cycle.

q_4 is used to move left, changing all 1's to B's until encountering a blank B.

When q_6 is entered, B is changed back to 0 and M halts.

The transition function δ is as follows :

States	Inputs		
	0	1	B
q_0	(q_1, B, R)	(q_5, B, R)	–
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	–
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)

Example :

(1) $m = 2, n = 1$

String $0^2 10^1$ computation is as follows :

$q_0 0010 \vdash B q_1 010 \vdash B0 q_1 10 \vdash B01 q_2 0 \vdash B0 q_3 11$

$\vdash B q_3 011 \vdash q_3 B011 \vdash B q_0 011$

$\vdash BB q_1 11 \vdash BB1 q_2 1 \vdash BB11 q_2 \vdash BB1 q_4 1$

$\vdash BB q_4 1 \vdash B q_4 \vdash B0 q_6$ (q_6 is accepting state)

(2) $m = 1, n = 2$

String 010^2 computation is as follows:

$q_0 0100 \vdash B q_1 100 \vdash B1 q_2 00 \vdash B q_3 110 \vdash q_3 B110$

$\vdash B q_0 110 \vdash BB q_5 10 \vdash BBB q_5 0 \vdash BBBB q_5$

$\vdash BBBBB q_6$ (Since $n > m$, the result is blank)

4. Is it possible that a Turing Machine could be considered as a computer of functions from integers to integer? If yes, justify your answer. **(Nov/Dec 2003)**

Solution :

Yes. The turing machine may be viewed as a computer of functions from positive integers to positive integers. Each integer will be represented in an unary fashion. That is, the integer $i \geq 0$ is represented by the string 0^i. If a function has k arguments $i_1, i_2, ... i_k$ then these integers are initially placed on the tape separated by 1 has $0^{i_1} 1\ 0^{i_2} 1\ 0^{i_2} 1$ ..... $1\ 0^{i_k}$

Definition :

A function $f : N \rightarrow N$ is said to be computable function of k arguments if there exists a Turing machine M halts with a tape consisting of 0^m for some m, where

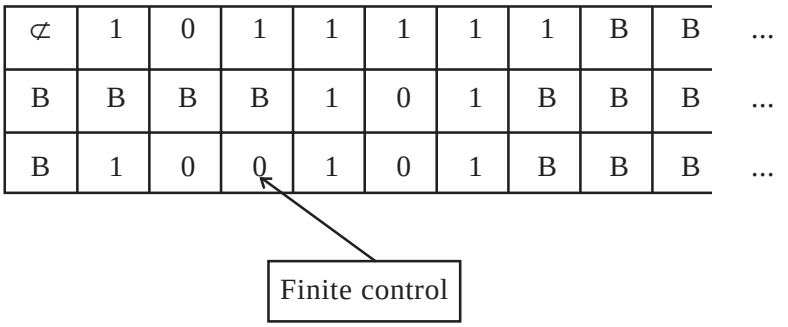
$f(i_1, i_2, ... i_k) = m$.

If $f(i_1, i_2, ... i_k)$ is defined for all $(i_1, i_2, ... i_k)$ then we say that f is *total recursive function.* A function $f(i_1, i_2, ... i_k)$ computed by a TM is called *partial recursive function.*

5. Explain how a Turing machine with the multiple tracks of the tape can be used to determine the given number is prime or not?

Solution : **(Apr/May 2004)**

Consider a TM with a 3 track tape as shown in figure:



It takes a binary input greater than 2, written on the first track, and determines whether it is a prime. The input is surrounded by the symbols and $ on the first track. Thus the allowable input symbols are [⊄, B, B], [0, B, B], [1, B, B] and [$, B, B]. These symbols can be identified with ⊄, 0, 1, & $ respectively when viewed as input symbols. The blank symbol can be identified with [B, B, B].

To test if the input is a prime, the TM first writes the number 2 in binary on the second track and copies the first track on to the third. Then the second track is subtracted as many times as possible, from the third track effectively dividing the third track by the second and leaving the remainder.

If the remainder is zero, the number on the first track is not a prime. If the remainder is non-zero, the number on the second track is increased by one. If the second track equals the first, the number on the first track is a prime, because it can not be divided by any number lying properly between 1 and itself. If the second is less than first, the whole operation is repeated for the new number on the second track.