Turing Machines

LEARNING OBJECTIVES

After completing this chapter, the reader will be able to understand the following:

- Computational problems that can be solved using a Turing machine (TM)
- Formal model of a TM
- Comparison of finite automata (FA) and TM on the basis of their relative computational powers
- Recursively enumerable languages and recursive languages
- Concept of composite TM and iterative TM
- Multi-tape TM, multi-stack TM, and multi-track TM
- Concept of universal Turing machine (UTM)
- Halting problem and limits of computability
- Church's Turing hypothesis
- Solvable, semi-solvable, and unsolvable problems
- Total and partial recursive functions
- Post's correspondence problem (PCP)

4.1 INTRODUCTION

The limitations of finite state machines (FSMs) necessitate the search for a more powerful machine. We have seen that an FSM cannot remember an arbitrarily long sequence of symbols. It has a read head that can move only in one direction—to the right always. It cannot move backwards to retrieve previous information stored onto the tape. Similarly, though the two-way finite automaton (2FA) has a read head that can move in any direction, it cannot store (write) anything onto the tape; hence, it cannot multiply two numbers, as the process requires intermediate results to be stored onto the tape. Furthermore, an FSM cannot check if a set of parentheses are well-formed, or for palindrome sequences. All these limitations arise because the FSMs do not have memory, and hence, they cannot solve problems that need to store data to be used for later computation.

We have already discussed the limitations of FSMs in detail in Chapter 2 (refer to Section 2.14). To overcome these limitations, we require a more powerful machine, which has the following properties:

1. Finite state control similar to that of the FSM—since any program needs a finite number of functions/states.
2. External memory capable of remembering arbitrarily long sequences of inputs—to achieve this, the machine should have unbounded one-dimensional memory from which, it can choose any required part, by moving to the left, right, or staying in one position (i.e., no movement)—the entire unbounded tape acts as an infinite memory.
3. Ability to store (write onto the tape)—the machine head should be a read/write head.
4. Ability to consume its own output as input during execution at a later time (like a complete feedback control system)—for this, all the intermediate information must be stored in the memory. For example, while multiplying numbers (multiplication is a process of repetitive addition), it is required to store the intermediate or partial sums, which are later added to get the final answer. Hence, the distinction between the input and output symbols needs to be dropped. This means that external communication as well as communication within the machine should rest on a common alphabet set.

All the aforementioned characteristics are satisfied by the *Turing machine* (TM), which was proposed by Alan Turing, a decade prior to the designing of the first shared-program computer. This concept of the Turing machine led to the concept of algorithms and finite procedures, since the basis of the TM is to first divide the process into primitive operations (functions/procedures/states) such that they cannot be further divisible, and then execute each operation in sequence.

4.2 ELEMENTS OF A TURING MACHINE

A TM consists of the following:

1. A head, which can read or write a symbol at a time, and move either to the left, right, or remain in the same position, depending on the symbol read from the tape cell.
2. An infinite tape, extending on either side of the head, marked-off into square cells, on which the symbols from an alphabet set can be written. The tape represents the unbounded one-dimensional memory (refer to Fig. 4.1), which is considered to be filled with blank characters, $\flat$'s, unless specified otherwise.
3. A finite set of symbols, called the external alphabet set I, which consists of lower-case English letters, digits, usual punctuation marks, and the blank character $\flat$.
4. A finite set of states denoted by S; the machine resides in one of these states.

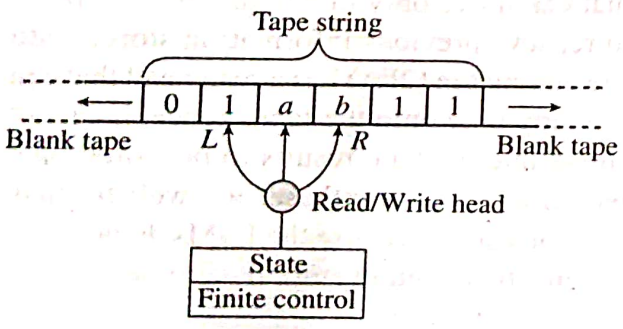


Figure 4.1 Turing machine

The TM thus overcomes all the limitations of the FSM. It has infinite memory in the form of a tape that is unbounded at both the ends; a read/write head that can move in any direction, and helps the machine retrieve information from the tape that is stored earlier—in a way, this can be considered as converting the dumb storage into memory; it can consume its own output as input for use at a later stage in the algorithm execution. All these features make it a completely powerful machine. The TM and FSM are thus two extremes—an FSM is a minimal machine, whereas a TM is the most powerful machine.

4.3 TURING MACHINE FORMALISM

As we have already seen, a TM has a finite set of symbols I, and a finite set of states S (one of which is the halt/final state).

A TM is denoted with the help of three functions, namely, the machine function (MAF), state transition function (STF), and the direction function (DIF), which are defined as:

$$\text{MAF: } I \times S \rightarrow I$$
$$\text{STF: } I \times S \rightarrow S$$
$$\text{DIF: } I \times S \rightarrow D = \{L, R, N\}$$

where, L stands for 'move towards left by one tape cell position', R stands for 'move towards right by one tape cell position', and N stands for 'no movement, that is, stay in the same position or remain at the same tape cell, which has been read last'.

Note: Observe that the MAF for FSM is defined as: $I \times S \rightarrow O$, as there is a distinction between input and output; on the other hand, in case of a TM, it is defined as: $I \times S \rightarrow I$.

Based on the machine's state—which is an element of S—at any given time, and the input symbol read—which is an element of I—at any given time, the machine either takes no action (i.e., halt state) or performs the following three actions:

1. The input symbol read is erased and another symbol (possibly a blank character or the same symbol again) is printed on the tape cell.
2. The internal state of the machine is changed (possibly to the same state as it was initially in).
3. The head either moves one cell towards the right or left, or remains in the same position.

These actions can be described collectively by an ordered set of five elements, that is, a quintuple. For example, (a, α, b, β, L) represents such a quintuple, which exists if the head reads $a \in I$, while the machine is in state $\alpha \in S$, writes b onto the tape cell after erasing a, changes the machine state to $\beta \in S$, and moves the head position to the left by one cell.

To represent these quintuples together, a table called a transition matrix or functional matrix (FM) is used. Such a matrix is a combination of the aforementioned three individual functions—the MAF, STF, and DIF. In this functional matrix, the rows and columns are labelled by symbols forms S and I, respectively (as in FSMs), while the remaining triples, that is, (b, β, L) are placed as the entries of the matrix.

Table 4.1 Example functional matrix

S \ I	0	1	$\not b$
α	$0\alpha R$	$0\beta R$	$\not b\alpha R$
β	—	—	$\not b\gamma N$
γ	—	—	—

If the number of symbols in S, that is, the number of states = $|S| = m$, and the number of symbols in I, that is, number of tape symbols = $|I| = n$, then the size of the functional matrix is $(m \times n)$ number of output triples. For an example, refer to Table 4.1.

We see that in Table 4.1, S has three states and I has three tape symbols; hence, the functional matrix is of size $3 \times 3 = 9$.

If δ is a functional matrix, then an example transition can be expressed as:

$$\delta\ (a, \alpha) \rightarrow (a, \alpha, L)$$

This means, on reading symbol a while in α state, the machine erases a, writes a again, changes the state from α to itself, and moves the head position to the left by one tape cell.

In the aforementioned case, the significant change after transition is just the position of the head, because the symbol on the tape a and the current state of the machine, that is, α, remain the same even after the transition. Therefore, instead of writing the whole triple in the functional matrix, we can save space by writing only the significant changes. Hence, the aforementioned triple can be simplified as:

Table 4.2 Simplified functional matrix

S \ I	0	1	$\not b$
α	R	$0\beta R$	R
β	—	—	γN
γ	—	—	—

$$\delta\ (a, \alpha) \rightarrow (L)$$

The modified functional matrix, in which we only record the significant changes after deleting all other insignificant entries to save space, is called a *simplified functional matrix* (SFM). The SFM equivalent to the functional matrix in Table 4.1 is depicted in Table 4.2.

Note: It is also possible to convert a quintuple representation to a quadruple representation. For this, additional states may have to be introduced in S, so that the given TM is deterministic. For example, the quintuple (a, α, b, β, R) may be written as two quadruples: (a, α, b, β') and (b, β', R, β), where, β' is a new state. This quadruple notation essentially breaks two of the three operations into different transitions. The first transition (a, α, b, β') is about replacing the input symbol a by b and the second transition (b, β', R, β) is about moving the head position one cell to the right. However, this quadruple notation is very rarely used; the quintuple notation discussed here is more popular.

Formal Definition

A Turing machine (TM) is denoted here:

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

where,

Q: Finite set of states

Γ: Finite set of allowable tape symbols, including blank character $\not b$

Σ: The set of input symbols, which is a subset of Γ, excluding blank character $\not b$

B: A symbol for blank character $\not b$

q_0: Start (or initial) state $\in Q$
F: Set of final states (or halt states) $\subseteq Q$
δ: Functional matrix, such that δ: $(Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R, N\})$

In order to have a deterministic TM, there should be a unique triple (b, β, d) for every state α on some symbol a, for $\delta(\alpha, a)$ in the functional matrix.

Notes:

1. δ may be undefined for some arguments, which are represented in the table as '—', (similar to the transition table of an FSM). Refer to Tables 4.1 and 4.2.
2. FSM is a special case of a TM. If we make the tape length finite and restrict the head movement only to one direction, then the functional matrix corresponding to this FSM will specify: $I \times S \rightarrow S$, which is, nothing but the state transition function (STF).

4.4 INSTANTANEOUS DESCRIPTION

Instantaneous description (ID) of a TM M is denoted by '$\alpha_1\ q\ \alpha_2$', where q is the current state of the TM, that is, $q \in Q$, and '$\alpha_1\ \alpha_2$' is the string in Γ^*—that is, the contents of the tape bounded on both the ends by blank characters ($\not b$'s). Note that $\not b$ may occur within '$\alpha_1\ \alpha_2$' as well.

We define a move (or transition) of M as follows:

Let '$a_0\ a_1 \ldots a_{i-1}\ q\ a_i \ldots a_n$' be an ID of M. This ID indicates that the current state of the machine is q and the machine head is about to read symbol a_i from the tape.

Suppose, $\delta(q, a_i) = (*, p, L)$ is a transition, where

If $i = n + 1$, then a_i is taken to be $\not b$, that is, blank character.

If $i = 0$, then the tape head moves one position to the left of the first symbol that is considered to be blank character $\not b$, in the next ID.

If $i > 1$, then we can write the ID for the aforementioned transition as

$$a_0\ a_1 \ldots a_{i-1}\ q\ a_i \ldots a_n \underset{M}{\vdash} a_0\ a_1 \ldots a_{i-2}\ p\ a_{i-1}\ *\ a_{i+1} \ldots a_n$$

Similarly, for the move $\delta(q, a_i) = (*, p, R)$ we can write

$$a_0\ a_1 \ldots a_{i-1}\ q\ a_i \ldots a_n \underset{M}{\vdash} a_0\ a_1 \ldots a_{i-1}\ *\ p\ a_{i+1} \ldots a_n$$

If the two IDs are related by '$\underset{M}{\vdash}$', we say that the second results from the first by one move, as we have seen here. If one ID results from another by some finite number of moves (including zero number of moves), then they are related by the symbol, '$\overset{*}{\underset{M}{\vdash}}$'.

The language accepted by M is denoted by $L(M)$ and is the set of those words in * that cause M to enter a final state when M is placed in state q_0 and the tape head of M is at the leftmost cell.

Formally, we can define the language accepted by $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ as

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

Let us consider an example of a TM.

Example 4.1 Consider an SFM for a TM with the following:

$$I = \{0, 1, \not b\}$$
$$S = \{\alpha, \beta, \gamma = \text{halt}\}$$
$$D = \{L, R, N\}$$

The SFM is given in Table 4.3.

Find the purpose of the aforementioned TM, provided the initial configuration of the TM is as given in Fig. 4.2.

Table 4.3 SFM for example TM

S \ I	0	1	$\not b$
α	R	$0\beta R$	R
β	—	—	γN
γ	—	—	—

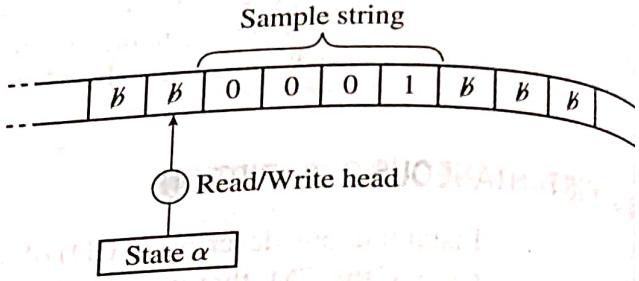


Figure 4.2 Initial configuration of example TM

Solution It is very important to note that the functional matrix gives us only half the information on what the TM does, and unless we know the initial configuration or initial ID of the TM, we cannot determine its purpose. Thus, the interpretation of the TM is highly dependent on its initial configuration, which includes the following: what the TM starts with, how the input string is assumed to be placed onto the tape, and the initial state of the machine. Note that the interpretation may differ for different initial configurations even for the same functional matrix.

Therefore, we can say that the initial configuration or the initial ID is like the assumption that forms the basis for the interpretation of the algorithm described by the functional matrix.

As we can see, initially the machine is the state α, and the head points to the blank character $\not b$ just before the actual string begins; hence, α is the initial or start state of the TM.

We observe that the function of the aforementioned TM is to replace the last (or ending) 1 by 0, whenever a sequence of 0's followed by a 1 is encountered; it then moves to the next available blank, and halts.

Let us simulate the working of the TM using a sample string '0001':

$$\alpha \not b 0\,0\,0\,1 \not b \qquad \text{initial configuration}$$
$$\vdash_M \not b\, \alpha\, 0\,0\,0\,1 \not b \qquad \delta(\alpha, \not b) = (R)$$
$$\vdash_M \not b\, 0\, \alpha\, 0\,0\,1 \not b \qquad \delta(\alpha, 0) = (R)$$
$$\vdash_M \not b\, 0\,0\, \alpha\, 0\,1 \not b \qquad \delta(\alpha, 0) = (R)$$

$$\vdash_M \; \flat\, 0\, 0\, 0\, \alpha\, 1\, \flat \qquad \delta(\alpha, 0) = (R)$$

$$\vdash_M \; \flat\, 0\, 0\, 0\, 0\, \beta\, \flat \qquad \delta(\alpha, 1) = (0\, \beta\, R)$$

$$\vdash_M \; \flat\, 0\, 0\, 0\, 0\, \gamma\, \flat \qquad \delta(\beta, \flat) = (\gamma\, N)$$

The simulation of a TM for a particular input is a recording of the transition from one ID to another. The simulation given here is a formal notation. However, the IDs can also be represented using a slightly different notation as shown in the following simulation. We observe that in the changed notation, the head is clearly shown as a pointer pointing to the tape cell to be read. This change makes it visibly simple for the reader to understand. In this chapter, we are going to follow the same changed notation for representing the IDs for TM in preference to the formal one. Using the simpler ID notation, the aforementioned simulation can be shown as follows:

$\flat\, 0\, 0\, 0\, 1\, \flat$ $\uparrow$ α	initial configuration
$\flat\, 0\, 0\, 0\, 1\, \flat$ $\quad\uparrow$ $\quad\alpha$	moved one position to right using $\delta(\alpha, \flat)$
$\flat\, 0\, 0\, 0\, 1\, \flat$ $\quad\quad\uparrow$ $\quad\quad\alpha$	moved one position to right using $\delta(\alpha, 0)$
$\flat\, 0\, 0\, 0\, 1\, \flat$ $\quad\quad\quad\uparrow$ $\quad\quad\quad\alpha$	moved right position to right using $\delta(\alpha, 0)$
$\flat\, 0\, 0\, 0\, 1\, \flat$ $\quad\quad\quad\quad\uparrow$ $\quad\quad\quad\quad\alpha$	moved one position to right using $\delta(\alpha, 0)$
$\flat\, 0\, 0\, 0\, 0\, \flat$ $\quad\quad\quad\quad\quad\uparrow$ $\quad\quad\quad\quad\quad\beta$	changed 1 to 0, changed state to β, and moved one position to right using $\delta(\alpha, 1)$
$\flat\, 0\, 0\, 0\, 0\, \flat$ $\quad\quad\quad\quad\quad\uparrow$ γ (halt)	changed state to γ with no movement using $\delta(\beta, \flat)$, halt after that

4.5 TRANSITION GRAPH FOR TURING MACHINE

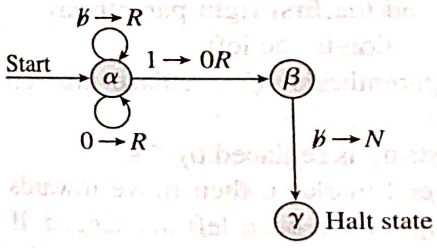


Figure 4.3 TG for example TM

A TM can also be represented pictorially with the help of a transition graph (TG). In such a graph, the nodes denote the internal states from the set S. A pair of nodes is connected by a directed edge (or arc) labelled by the input symbol from the set ⌈, followed by an arrow and the new output symbol again from the set ⌈— this symbol is to be written after erasing the previous symbol; the direction of the move is selected from the set D. The transition graph for the TM in Example 4.1 is shown in Fig. 4.3.

Note: If a TM starts the run from an initial state and eventually reaches a halt state, the computation is stopped and we say that it has *accepted* the input word.

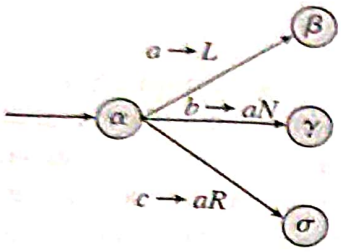


Figure 4.4 Example transition graph

On the other hand, let us consider the transition graph in Fig. 4.4. If the machine is in state α, and if the symbol read is different from a, b, and c, then we cannot proceed with the transition. This, in fact, corresponds to having unspecified entries in the functional matrix. Further, if the machine is in state β and reads a or b, then there is no way to proceed. In such situations, the computation is halted and we say that the machine has *rejected* the input word.

Sometimes, the machine may go into an infinite loop without ever halting. For example, if the triplet corresponding to (b, β) is (b, β, R), that is $\delta(\beta, b) = (b, \beta, R)$, and the remainder of the tape consists of only blank characters, then the machine goes on moving to the right without ever changing the state creating an infinite loop.

4.6 SOLVED PROBLEMS

Let us try to construct some TMs that solve specific problems.

Example 4.2 Construct a TM for checking if a set of parentheses are well-formed.

Solution Before directly creating the functional matrix let us first decide the initial configuration, that is, we first begin with the assumption on which the algorithm is to be fixed and then generate the FM.

The initial configuration of this TM is as shown in Fig. 4.5(a).

It is assumed that the input string is to be written onto the tape delimited by semicolons on both the sides. The initial state of the machine is q_0, and the machine head points to the leftmost symbol of the input string. In case the input string is empty, the head points to the right end-marker semicolon.

Algorithm

1. From the initial state, move to the right until you read the first right parenthesis ')' replace this right parenthesis by '*' and move one position to the left.
2. Continue moving left till you read the first left parenthesis '(' ; replace this left parenthesis by '*' and move to the right.
3. Repeat the aforementioned two steps till the whole string is replaced by *'s.
4. While moving right if you came across ';' (right end-marker), then move towards the left to search for any left parenthesis '(' that might have been left unchanged. If the parentheses are well-formed, there will be no more left parentheses '('; and the machine will only read the left end-marker ';'.

For this TM, we have:

$$I = \{(, *,), ;, R_p, L_p, O\}$$
$$S = \{q_0, q_1, q_2, q_3 = \text{halt}\}$$
$$D = \{L, R, N\}$$

Here, q_0 is considered as the start state while q_3 is the halt state.

The SFM for the TM is as given in Table 4.4.

Table 4.4 SFM of a TM that checks if parentheses are well-formed

S \ I	(	*	)	;	R_p	L_p	O
q_0	R	R	$*q_1L$	q_2L	—	—	—
q_1	$*q_0R$	L	L	R_pq_3N	—	—	—
q_2	L_pq_3N	L	L	Oq_3N	—	—	—
q_3	—	—	—	—	—	—	—

If the TM encounters an extra right parenthesis ')', it ends up in halt giving R_p as output. If there is an extra left parenthesis '(', it moves to state q_3, that is, halt state, giving L_p as output. If the entire sequence of parentheses is balanced, then the machine moves to the halt state q_3 giving the output O, indicating acceptable input.

We observe from Table 4.4 that state q_0 is responsible for searching for the first right parenthesis ')'. It moves towards the right, ignoring any left parenthesis '(' that is encountered. Once the right parenthesis is found, it is replaced with '*'; then the machine moves one position to the left and changes to state q_1. State q_1 is responsible for searching towards the left for the matching left parenthesis '('. Once the first left parenthesis '(' is found, it is replaced with '*', and the machine moves back to q_0. Thus, the states q_0 and q_1 represent the first two steps of the algorithm.

While moving towards the left and searching for the left parenthesis '(' while in state q_1, if the machine cannot find '(', that is, the machine reads ';' (left end-marker) instead, then it is an error condition. In such a case, the machine generates output symbol R_p, indicating an extra right parenthesis, and moves to the halt state q_3.

On the other hand, if while moving towards the right in q_0 and searching for the first right parenthesis, if the machine reads ';' (right end-marker), indicating there are no more right parentheses ')', to be changed to '*', the machine moves to state q_2. In state q_2, one needs to check for any unmatched left parentheses, '(', that are left. In state q_2, while moving to the left, if the machine comes across the symbol ';' (left end-marker), indicating that there are no more unmatched left parentheses '(', it is considered as the acceptance condition. Hence, the machine generates the output O, indicating that the input string is accepted, and moves to the halt state q_3.

However, in case while moving left it finds a left parenthesis '(', then it indicates the string is not well-formed and has at least one unmatched extra left parenthesis '('. This is an error condition and the machine generates output symbol L_p and halts.

The transition graph for this TM is shown in Fig. 4.5(b).

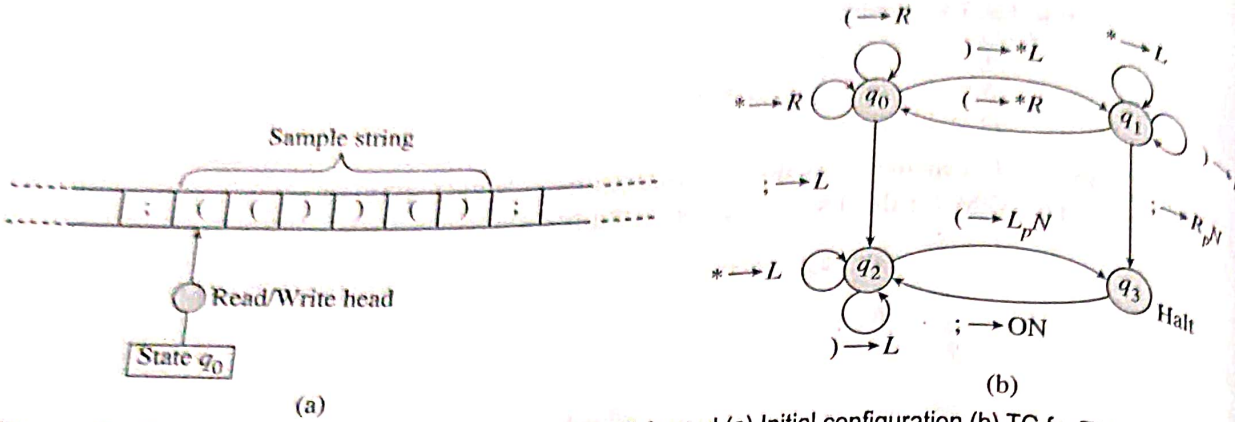
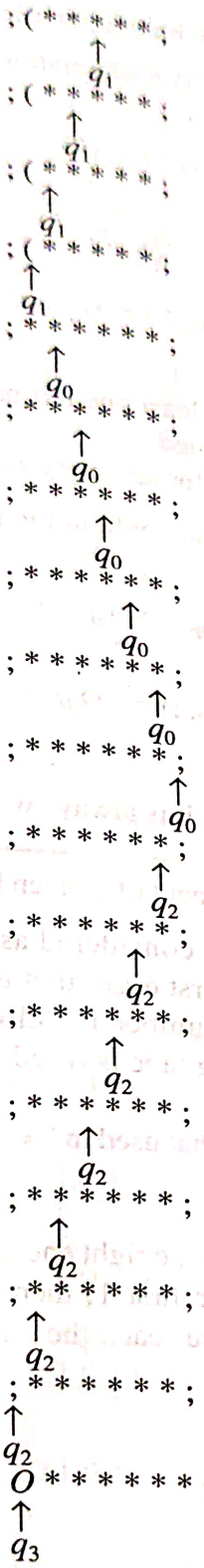


Figure 4.5 TM for checking if a set of parentheses are well-formed (a) Initial configuration (b) TG for TM that checks if parentheses are well-formed

Simulation

1. Let us simulate the working of the aforementioned TM on the sequence '(( ) ( ))'.

$;(()())$; $\uparrow$ q_0	initial configuration
$;(()())$; $\uparrow$ q_0	$\delta(q_0, () = (R)$
$;(()())$; $\uparrow$ q_0	$\delta(q_0, () = (R)$
$;(({}^*())$; $\uparrow$ q_1	$\delta(q_0,)) = ({}^* q_1, L)$
$;({}^*{}^*())$; $\uparrow$ q_0	$\delta(q_1, () = ({}^* q_0 R)$
$;({}^*{}^*())$; $\uparrow$ q_0	$\delta(q_0, {}^*) = (R)$
$;({}^*{}^*())$; $\uparrow$ q_0	$\delta(q_0, () = (R)$
$;({}^*{}^*(){}^*)$; $\uparrow$ q_1	$\delta(q_0,)) = ({}^* q_1 L)$
$;({}^*{}^*{}^*{}^*)$; $\uparrow$ q_0	$\delta(q_1, () = ({}^* q_0 R)$
$;({}^*{}^*{}^*{}^*)$; $\uparrow$ q_0	$\delta(q_0 {}^*) = (R)$
$;({}^*{}^*{}^*{}^*{}^*;$ $\uparrow$ q_1	$\delta(q_0,)) = ({}^* q_1 L)$



$\delta(q_1, *) = (L)$

$\delta(q_1, *) = (L)$

$\delta(q_1, *) = (L)$

$\delta(q_1, *) = (L)$

$\delta(q_1, ( ) = (* \; q_0 \; R)$

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

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

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

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

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

$\delta(q_0, ;) = (q_2 \; L)$

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

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

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

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

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

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

$\delta(q_2, ;) = (O \; q_3 \; N)$

The output symbol O indicates that the string '(( ) ( ))' is accepted by the TM as it is well-formed.

2. Let us now simulate the working of this TM for input sequence '('.

initial configuration

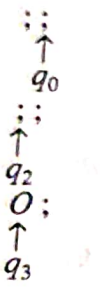
$\delta(q_0, (\,) = (R)$

$\delta(q_0, ;) = (q_2\ L)$

$\delta(q_2, (\,) = (L_p\ q_3\ N)$

The output symbol L_p indicates that there is at least one left parenthesis '(', which is unmatched, and the input string is not well-formed.

3. Finally, let us simulate the working of this TM for an empty sequence:



initial configuration

$\delta(q_0, ;) = (q_2\ L)$

$\delta(q_2, ;) = (O\ q_3\ N)$

We observe that the empty string is accepted, as it is always well-formed.

Example 4.3 Design a TM to find the 2's complement of a given binary number.

Solution The initial configuration of the TM is considered as shown in Fig. 4.6(a). The initial state is q_0, and the head points to the first digit, that is, the most significant bit (MSB), of the binary number. The binary number is delimited by semicolons, which are used as the end-markers. The remaining tape is filled with blank characters, that is, $\not b$'s.

The algorithm we are going to use is the same as that used in Example 2.17 in Chapter 2.

Algorithm

1. Move towards the right till you reach ';', which is the right end-marker of the sequence.
2. Start moving towards the left till you reach the first 1; then move towards the left and replace each 0 by 1, and each 1 by 0, till you reach the left end-marker ';' of the sequence, then halt.

For this TM, we have

$$I = \{0, 1, ;\}$$
$$S = \{q_0, q_1, q_2, q_3 = \text{halt}\}$$
$$D = \{L, R, N\}$$

The SFM for the TM is given in Table 4.5.

Table 4.5 SFM for a TM that finds 2's complement of a binary number

S \ I	0	1	;
q_0	R	R	q_1L
q_1	L	q_2L	q_3N
q_2	$1L$	$0L$	q_3N
q_3	—	—	—

The TG for the TM is as shown in Fig. 4.6(b).

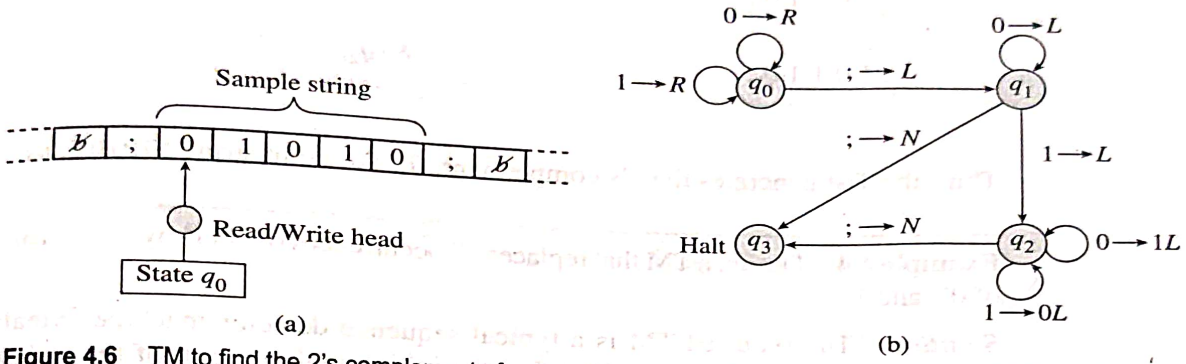
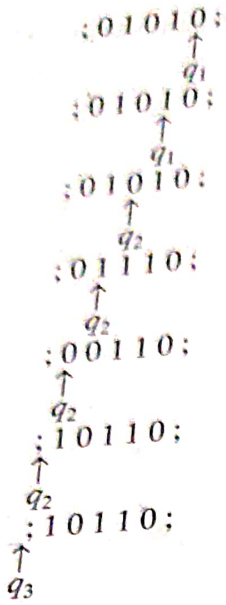


Figure 4.6 TM to find the 2's complement of a given binary number (a) Initial configuration (b) TG for TM that finds 2's complement of a binary number

Simulation

Let us simulate the working of this TM for the string '01010'.

; 0 1 0 1 0 ; ↑ q_0	initial configuration
; 0 1 0 1 0 ; ↑ q_0	$\delta\ (q_0, 0) = (R)$
; 0 1 0 1 0 ; ↑ q_0	$\delta\ (q_0, 1) = (R)$
; 0 1 0 1 0 ; ↑ q_0	$\delta\ (q_0, 0) = (R)$
; 0 1 0 1 0 ; ↑ q_0	$\delta\ (q_0, 1) = (R)$
; 0 1 0 1 0 ; ↑ q_0	$\delta\ (q_0, 0) = (R)$



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

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

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

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

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

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

$\delta(q_2, ;) = (q_3\ N)$

Thus, the TM generates the 2's complement, for the input string '01010', as '10110'.

Example 4.4 Design a TM that replaces all occurrences of '111' by '101' from a sequen of 0's and 1's.

Solution The required TM is a typical sequence detector machine (Mealy machin that we have studied in Chapter 2. Let the initial configuration of the TM be as shov in Fig. 4.7(a).

The initial state is assumed to be q_0. The string is delimited at the rightmost end by semicolon ';', which is used as the end-marker. The TM starts reading from the leftmo symbol in the input string.

The SFM for the required TM, which converts each occurren of '111' by '101' is as given in Table 4.6.

Table 4.6 *SFM for a TM that replaces each occurrence of '111' by '101'*

S \ I	0	1	;
q_0	R	q_1R	q_5N
q_1	q_0R	q_2R	q_5N
q_2	q_0R	q_3L	q_5N
q_3	—	$0q_4R$	—
q_4	—	q_0R	—
q_5	—	—	—

For this TM, we have

$I = \{0, 1, ;\}$
$S = \{q_0, q_1, q_2, q_3, q_4, q_5 = \text{halt}\}$
$D = \{L, R, N\}$

We see from Table 4.6 that states q_0, q_1, and q_2 collectively iden tify the sequence '111'. Once the sequence is identified, state q makes a transition to state q_3 and points to the middle 1, which i replaced by 0 in state q_3; thus, the string is replaced by '101'. Onc the replacement is done, the TM moves to state q_4, which resets t state q_0 for repeating the same process again to search for anothe sequence of '111' that may exist.

The TG for the TM is shown in Fig. 4.7(b).

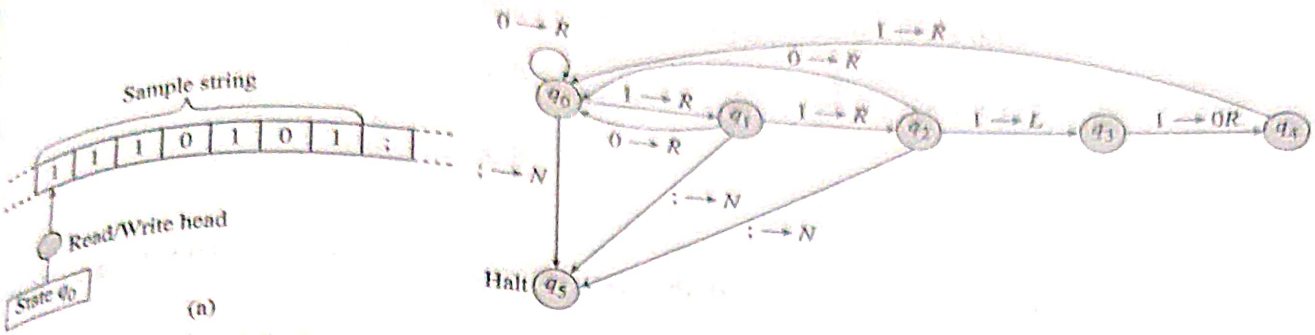


Figure 4.7 TM that replaces all occurrences of '111' by '101' (a) Initial configuration (b) TG for TM that replaces each occurrence of '111' by '101'

Simulation

Let us simulate the working of the TM that we have constructed for the input string '11110111'.

1 1 1 1 0 1 1 1 ; ↑ q_0	initial configuration
1 1 1 1 0 1 1 1 ; ↑ q_1	$\delta\ (q_0, 1) = (q_1\ R)$
1 1 1 1 0 1 1 1 ; ↑ q_2	$\delta\ (q_1, 1) = (q_2\ R)$
1 1 1 1 0 1 1 1 ; ↑ q_3	$\delta\ (q_2, 1) = (q_3\ L)$
1 0 1 1 0 1 1 1 ; ↑ q_4	$\delta\ (q_3, 1) = (0\ q_4\ R)$
1 0 1 1 0 1 1 1 ; ↑ q_0	$\delta\ (q_4, 1) = (q_0\ R)$
1 0 1 1 0 1 1 1 ; ↑ q_1	$\delta\ (q_0, 1) = (q_1\ R)$
1 0 1 1 0 1 1 1 ; ↑ q_0	$\delta\ (q_1, 0) = (q_0\ R)$
1 0 1 1 0 1 1 1 ; ↑ q_1	$\delta\ (q_0, 1) = (q_1\ R)$
1 0 1 1 0 1 1 1 ; ↑ q_2	$\delta\ (q_1, 1) = (q_2\ R)$
1 0 1 1 0 1 1 1 ; ↑ q_3	$\delta\ (q_2, 1) = (q_3\ L)$

$$\delta(q_3, 1) = (0\ q_4\ R)$$

$$\delta(q_4, 1) = (q_0\ R)$$

$$\delta(q_0, :) = (q_5\ N)$$

1 0 1 1 0 1 0 1 :
↑ q_3

1 0 1 1 0 1 0 1 :
↑ q_0

1 0 1 1 0 1 0 1 :
↑ q_5

Thus, the TM has converted both the occurrences of '111' from the input string to '101'.

Note: As we know, the particular problem we have discussed can also be solved using Mealy machine. Since the TM is a more powerful machine, it can always solve problems that are solvable using an FSM.

Example 4.5 Design a TM that recognizes words of the form $0^n\ 1^n$ for $n \geq 0$.

Solution Let us assume the initial configuration as shown in Fig. 4.8(a). The transition graph for the TM is shown in Fig. 4.8(b).

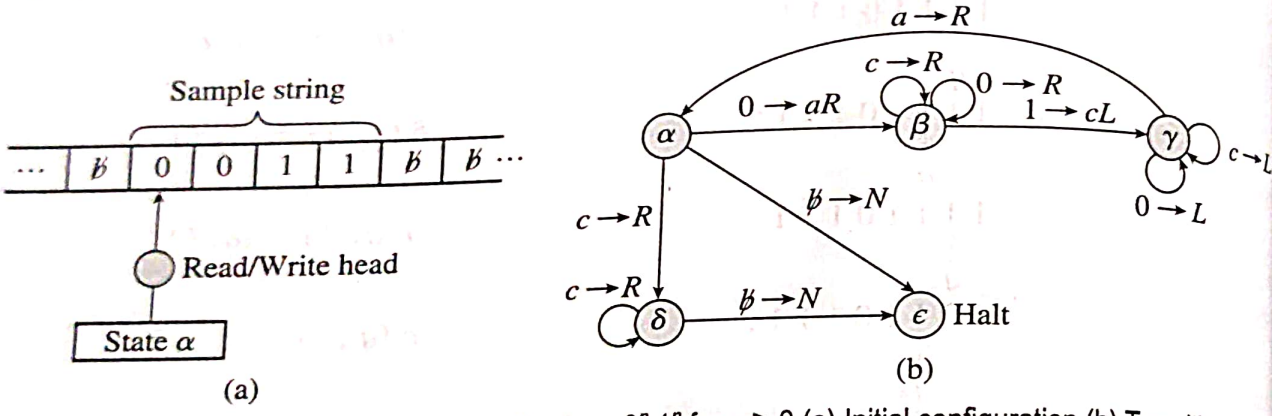


Figure 4.8 TM that recognizes strings of the form $0^n\ 1^n$ for $n \geq 0$ (a) Initial configuration (b) Transition graph

Algorithm

1. Replace the first 0 by some other symbol, say a, and move towards the right till you reach the first 1.
2. Replace this first 1 by some other symbol, say c, and move towards the left till you reach the 0 immediately next to the one that was previously replaced by a.
3. Repeat the procedure till all 0's are replaced by a's.

For this TM, we have

$$I = \{0, 1, a, c, ƀ\}$$
$$S = \{\alpha, \beta, \gamma, \delta, \epsilon = \text{halt}\}$$
$$D = \{L, R, N\}$$

The SFM for the TM is as shown in Table 4.7.

Table 4.7 SFM for a TM that recognizes string $0^n 1^n$

S \ I	0	1	a	c	b
α	$a\beta R$	—	—	δR	ϵN (accept)
β	R	$c\gamma L$	—	R	—
γ	L	—	αR	L	—
δ	—	—	—	R	ϵN (accept)
ϵ	—	—	—	—	—

Here, α is the initial state of the machine and ϵ is the halt state. The transition diagram is drawn as shown in Fig. 4.8(b).

Simulation

1. Let us simulate the working of the machine for the string '0011'.

$0\ 0\ 1\ 1\ \not b$
$\uparrow$
α — initial configuration

$a\ 0\ 1\ 1\ \not b$
$\uparrow$
β — $\delta(\alpha, 0) = (a\ \beta\ R)$

$a\ 0\ 1\ 1\ \not b$
$\uparrow$
β — $\delta(\beta, 0) = (R)$

$a\ 0\ c\ 1\ \not b$
$\uparrow$
γ — $\delta(\beta, 1) = (c\ \gamma\ L)$

$a\ 0\ c\ 1\ \not b$
$\uparrow$
γ — $\delta(\gamma, 0) = (L)$

$a\ 0\ c\ 1\ \not b$
$\uparrow$
α — $\delta(\gamma, a) = (\alpha\ R)$

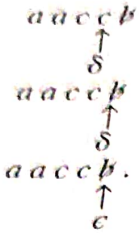
$a\ a\ c\ 1\ \not b$
$\uparrow$
β — $\delta(\alpha, 0) = (a\ \beta\ R)$

$a\ a\ c\ 1\ \not b$
$\uparrow$
β — $\delta(\beta, c) = (R)$

$a\ a\ c\ c\ \not b$
$\uparrow$
γ — $\delta(\beta, 1) = (c\ \gamma\ L)$

$a\ a\ c\ c\ \not b$
$\uparrow$
γ — $\delta(\gamma, c) = (L)$

$a\ a\ c\ c\ \not b$
$\uparrow$
α — $\delta(\gamma, a) = (\alpha\ R)$



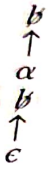
$\delta(\alpha, c) = (\delta R)$

$\delta(\delta, c) = (R)$

$\delta(\delta, b) = (c N)$

Thus, the string '0011' is accepted by the TM.

2. Let us simulate the working of this TM for the empty string.



initial configuration

$\delta(\alpha, b) = (c N)$

Empty string is accepted by the machine as it fits the pattern $0^n 1^n$ for $n = 0$. In case of empty string, the machine starts reading the trailing blank character, b.

Example 4.6 Design a TM that recognizes strings containing equal number of 0's and 1's.

Solution Let the initial configuration of the machine be as shown in Fig. 4.9(a).

Algorithm

1. Starting from the initial state, if you reach the first 0 replace it by '*', and move right till you reach the first 1; replace that also by the same symbol, '*'.
2. If you reach 1 first, then replace that with the symbol '*', and move right till you reach the first 0; replace that also by the same symbol, '*'.
3. Repeat the procedure till you finish reading all 1's and 0's. If any 1 or 0 remains, there must be an error, and the machine should reject the input.

For this TM, we have

$I = \{0, 1, ;, *, T, F\}$, where T = accepted; and F = rejected
$S = \{q_0, q_1, q_2, q_3, q_4 = \text{halt}\}$
$D = \{L, R, N\}$

The SFM for the TM is as shown in Table 4.8.

The initial state q_0 makes transition to q_1 after replacing the input symbol 0 by *. State q_1 hence carries the responsibility of finding the matching symbol 1. State q_3 is reached from q_0 on finding the symbol 1; hence, q_3 needs to find the matching 0 so that the number of 0's and 1's are same. The output symbol T is generated if the number of 0's and 1's are equal in the input string; else output F is generated.

Table 4.8 SFM for a TM that accepts strings containing equal number of 0's and 1's

S \ I	0	1	;	*	T	F
q_0	$*q_1R$	$*q_3R$	Tq_4N	R	—	—
q_1	R	$*q_2L$	Fq_4N	R	—	—
q_2	L	L	q_0R	L	—	—
q_3	$*q_2L$	R	Fq_4N	R	—	—
q_4	—	—	—	—	—	—

The transition diagram for the TM is as shown in Fig. 4.9(b).

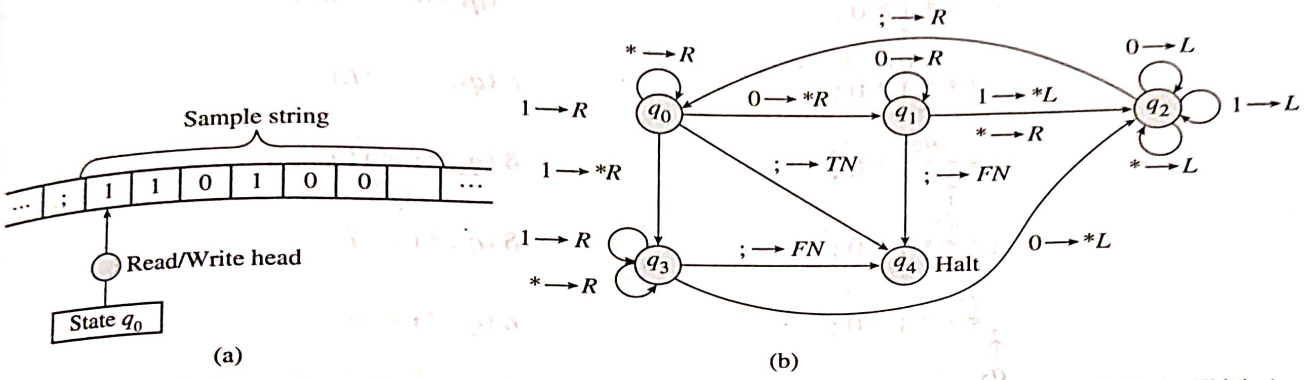
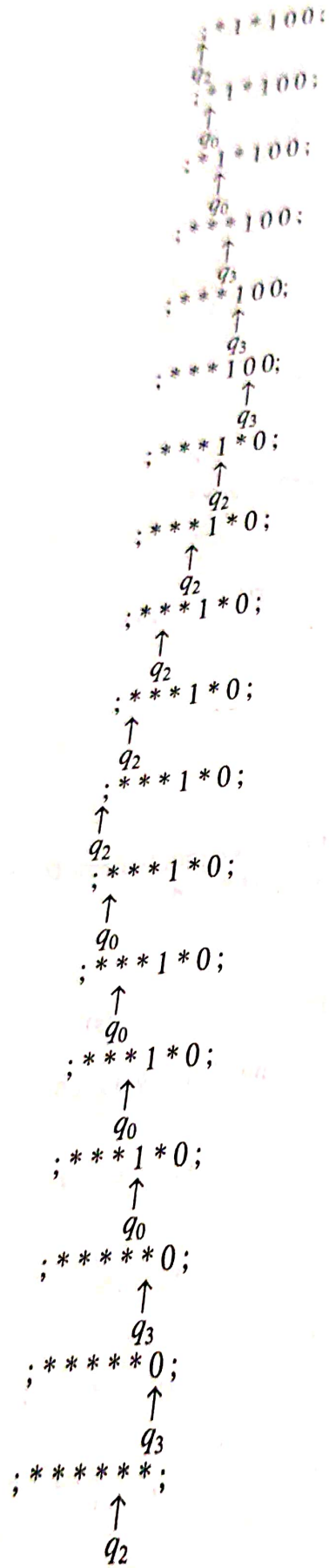


Figure 4.9 TM that recognizes strings containing equal number of 0's and 1's (a) Initial configuration (b) TG for TM that accepts all strings containing equal number of 0's and 1's

Simulation

1. Let us simulate the working of the TM for the string '110100'.

; 1 1 0 1 0 0 ; ↑ q_0	initial configuration
; * 1 0 1 0 0 ; ↑ q_3	$\delta\ (q_0, 1) = (* q_3 R)$
; * 1 0 1 0 0 ; ↑ q_3	$\delta\ (q_3, 1) = (R)$
; * 1 * 1 0 0 ; ↑ q_2	$\delta\ (q_3, 0) = (* q_2 L)$
; * 1 * 1 0 0 ; ↑ q_2	$\delta\ (q_2, 1) = (L)$



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

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

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

$\delta\ (q_0, 1) = (*\ q_3\ R)$

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

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

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

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

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

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

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

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

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

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

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

$\delta\ (q_0, 1) = (*\ q_3\ R)$

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

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

; * * * * * * ; ↑ q_2	$\delta\ (q_2, *) = (L)$
; * * * * * * ; ↑ q_2	$\delta\ (q_2, *) = (L)$
; * * * * * * ; ↑ q_2	$\delta\ (q_2, *) = (L)$
; * * * * * * ; ↑ q_2	$\delta\ (q_2, *) = (L)$
; * * * * * * ; ↑ q_2	$\delta\ (q_2, *) = (L)$
; * * * * * * ; ↑ q_0	$\delta\ (q_2, ;) = (q_0\ R)$
; * * * * * * ; ↑ q_0	$\delta\ (q_0, *) = (R)$
; * * * * * * ; ↑ q_0	$\delta\ (q_0, *) = (R)$
; * * * * * * ; ↑ q_0	$\delta\ (q_0, *) = (R)$
; * * * * * * ; ↑ q_0	$\delta\ (q_0, *) = (R)$
; * * * * * * ; ↑ q_0	$\delta\ (q_0, *) = (R)$
; * * * * * * ; ↑ q_0	$\delta\ (q_0, *) = (R)$
; * * * * * * T ↑ q_4	$\delta\ (q_0, ;) = (T\ q_4\ N)$

The output symbol T indicates that the string is accepted by the TM.

2. Let us now simulate the working of the TM for the string '1'.

; 1 ; ↑ q_0	initial configuration
; * ; ↑ q_3	$\delta\ (q_0, 1) = (*\ q_3\ R)$
; * F ↑ q_4	$\delta\ (q_3, ;) = (F\ q_4\ N)$

Thus, the string '1' is rejected by the machine as indicated by the output symbol F.

Example 4.7 Design a TM that recognizes binary palindromes.

Solution Let the initial configuration of the TM be as shown in Fig. 4.10(a).

In this case, we assume that the head initially points to the left end-marker ';', that is just prior to the actual start of the input. This assumption is the basis for the algorithm we write. Note that if we make a different assumption—for example, we may assume that the head points to the first input symbol—the algorithm as well as the SFM that is generated for the TM will be different.

Algorithm

1. Read the first symbol, which may be 0 or 1. Replace it by some other symbol, say ';'.
2. Move towards the right to find the right end of the sequence, that is, the right end marker ';'. Then, move one position to the left to point to the last symbol.
3. Read the last symbol. If it is equal to the symbol that we read at the other end, then replace it with ';', and continue the process.
4. If it is not equal, then the sequence is not a palindrome sequence.

For this TM, we have

$$I = \{0, 1, ;, F, T\}$$
$$S = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6 = \text{halt}\}$$
$$D = \{L, R, N\}$$

Here, T indicates that the input sequence is a palindrome and is therefore accepted by the TM; and F indicates that the input is not a palindrome sequence and is therefore rejected by the TM.

The SFM for the TM is as shown in Table 4.9.

Table 4.9 SFM for a TM that recognizes binary palindromes

S \ I	0	1	;	F	T
q_0	L	L	q_1R	—	—
q_1	$;q_2R$	$;q_4R$	Tq_6N	—	—
q_2	R	R	q_3L	—	—
q_3	$;q_0L$	Fq_6N	Tq_6N	—	—
q_4	R	R	q_5L	—	—
q_5	Fq_6N	$;q_0L$	Tq_6N	—	—
q_6	—	—	—	—	—

The transition graph is drawn as shown in Fig. 4.10(b).

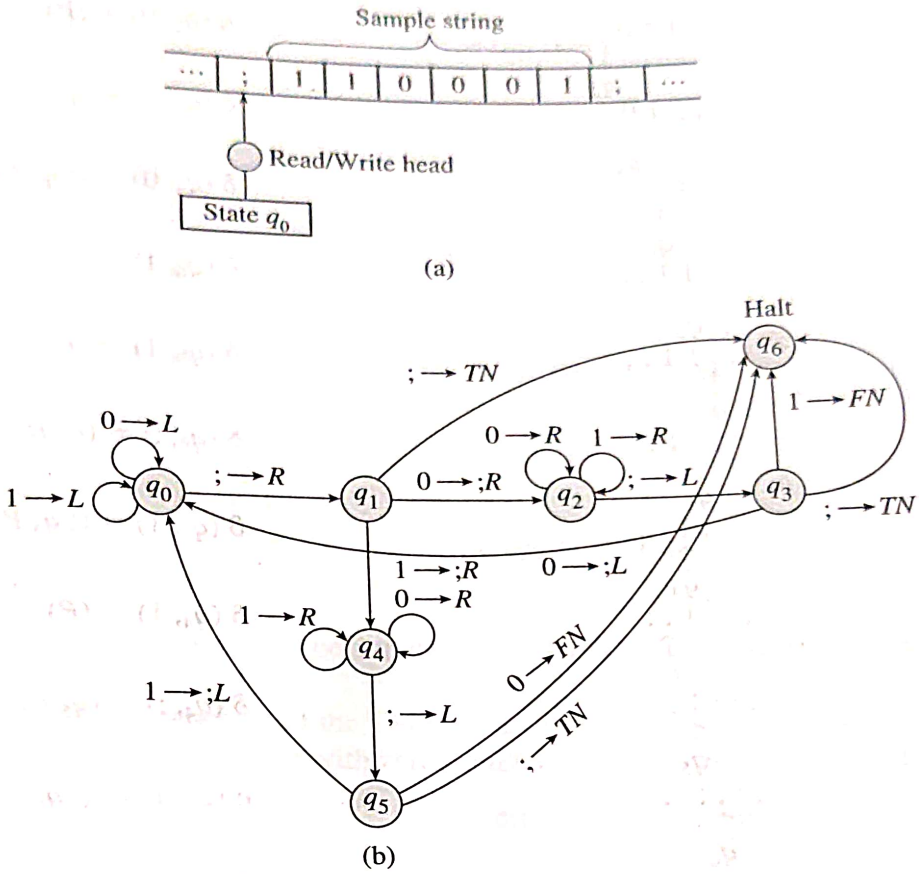


Figure 4.10 TM that recognizes binary palindromes (a) Initial configuration (b) TG for TM that recognizes binary palindromes

Simulation

1. Let us simulate the working of the TM that we have constructed for the string '0110', which is a binary palindrome sequence of even length.

```
;0110;          initial configuration
↑
q0
;0110;          δ (q0, ;) = (q1 R)
 ↑
 q1
;;110;          δ (q1, 0) = (; q2 R)
  ↑
  q2
;;110;          δ (q2, 1) = (R)
   ↑
   q2
```

; ; 1 1 0 ; ↑ q_2	$\delta(q_2, 1) = (R)$
; ; 1 1 0 ; ↑ q_2	$\delta(q_2, 0) = (R)$
; ; 1 1 0 ; ↑ q_3	$\delta(q_2, ;) = (q_3\ L)$
; ; 1 1 ; ; ↑ q_0	$\delta(q_3, 0) = (;\ q_0\ L)$
; ; 1 1 ; ; ↑ q_0	$\delta(q_0, 1) = (L)$
; ; 1 1 ; ; ↑ q_0	$\delta(q_0, 1) = (L)$
; ; 1 1 ; ; ↑ q_1	$\delta(q_0, ;) = (q_1\ R)$
; ; ; 1 ; ; ↑ q_4	$\delta(q_1, 1) = (;\ q_4\ R)$
; ; ; 1 ; ; ↑ q_4	$\delta(q_4, 1) = (R)$
; ; ; 1 ; ; ↑ q_5	$\delta(q_4, ;) = (q_5\ L)$
; ; ; ; ; ; ; ↑ q_0	$\delta(q_5, 1) = (;\ q_0\ L)$
; ; ; ; ; ; ; ↑ q_1	$\delta(q_0, ;) = (q_1\ R)$
; ; ; T ; ; ↑ q_6	$\delta(q_1, ;) = (T\ q_6\ N)$

The output *T* indicates that the input string '0110' is a palindrome sequence.

2. Let us simulate the working of the TM for the input string '101', which is a palindrome sequence of odd length.

; 1 0 1 ; ↑ q_0	initial configuration
; 1 0 1 ; ↑ q_1	$\delta(q_0, ;) = (q_1\ R)$
; ; 0 1 ; ↑ q_4	$\delta(q_1, 1) = (;\ q_4\ R)$
; ; 0 1 ; ↑ q_4	$\delta(q_4, 0) = (R)$

; ; 0 1 ; ↑ q_4	$\delta(q_4, 1) = (R)$
; ; 0 1 ; ↑ q_5	$\delta(q_4, ;) = (q_5\ L)$
; ; 0 ; ; ↑ q_0	$\delta(q_5, 1) = (; q_0\ L)$
; ; 0 ; ; ↑ q_0	$\delta(q_0, 0) = (L)$
; ; 0 ; ; ↑ q_1	$\delta(q_0, ;) = (q_1\ R)$
; ; ; ; ; ↑ q_2	$\delta(q_1, 0) = (; q_2\ R)$
; ; ; ; ; ↑ q_3	$\delta(q_2, ;) = (q_3\ L)$
; ; T ; ; ↑ q_6	$\delta(q_3, ;) = (T\ q_6\ N)$

The output T indicates that the input string '101' is recognized by the TM as a palindrome string.

3. Let us simulate the working of the TM for the string ϵ, that is, the empty string, which is also a palindrome sequence with zero number of 0's and zero number of 1's.

; ; ↑ q_0	initial configuration
; ; ↑ q_1	$\delta(q_0, ;) = (q_1\ R)$
; T ↑ q_6	$\delta(q_1, ;) = (T\ q_6\ N)$

Thus, the empty palindrome string is also accepted by the TM.

4. Let us simulate the working of the TM for input string '01', which is not a palindrome string.

; 0 1 ; ↑ q_0	initial configuration
; 0 1 ; ↑ q_1	$\delta(q_0, ;) = (q_1\ R)$
; ; 1 ; ↑ q_2	$\delta(q_1, 0) = (; q_2\ R)$
; ; 1 ; ↑ q_2	$\delta(q_2, 1) = (R)$

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

: : 1 :
↑
q_3

$\delta(q_3, 1) = (F\ q_6\ N)$

: : F :
↑
q_6

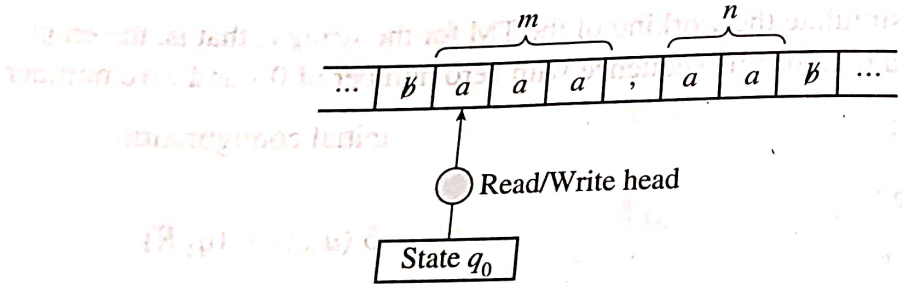
The output F indicates that the string '01' is not a palindrome string, and hence rejected by the TM.

Example 4.8 Design a TM, which compares two positive integers m and n and produc output G_t if $m > n$; L_t if $m < n$; and E_q if $m = n$.

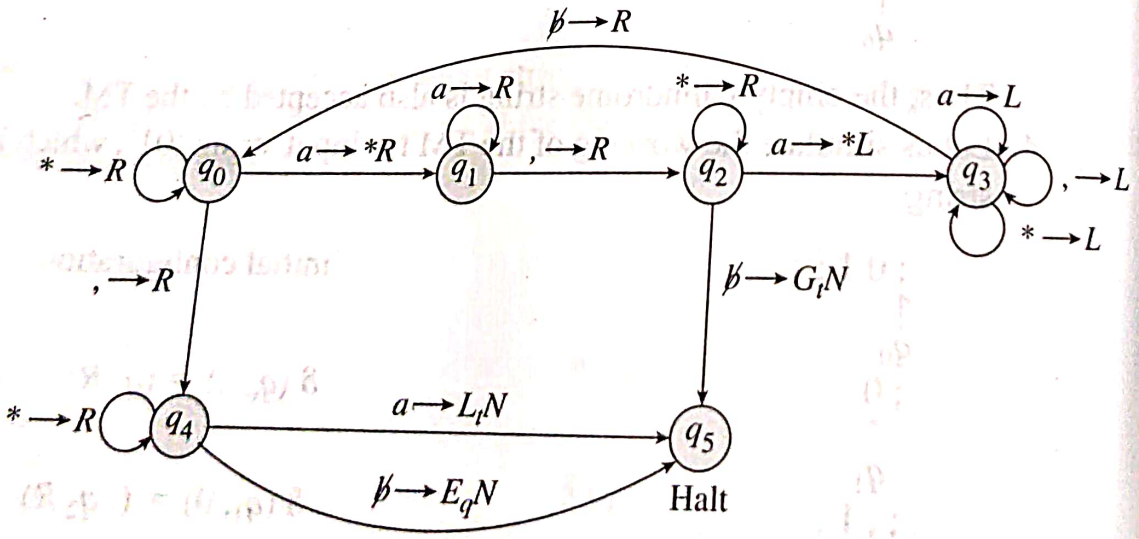
Solution This TM is a symbol manipulation system. For this, we need to represent numbers in the unary format. If we consider $\Sigma = \{a\}$, then the numbers can be represen using that many a's. For example, '2' in unary format can be written as 'aa' and '5' be written as '$aaaaa$'.

The initial configuration of the TM is assumed as shown in Fig. 4.11(a). Notice that two numbers, m and n, are separated by a punctuation symbol ',' (comma).

The transition diagram for the required TM is shown in Fig. 4.11(b).



(a)



(b)

Figure 4.11 TM that compares two positive integers (a) Initial configuration (b) Transition graph

Algorithm

1. Replace one a from m by '*'
2. Move right till you get the first a of n, replace it with '*'
3. If no a is remaining of n that is to be replaced by '*' then $m > n$
4. Move left to find the next a of m; if found repeat the aforementioned two steps
5. If no a is remaining of m then search if any a is remaining that of n, if yes then, $m < n$; if no then, $m = n$

In short, keep replacing each a of m and n by '*', until both or one of them gets fully replaced with *'s. If finally both are totally replaced by *'s, then they must be equal; otherwise, the one with some remaining a's must be greater than the other.

For this TM, we have

$$I = \{a, , , *, \not b, G_t, L_t, E_q\}$$
$$S = \{q_0, q_1, q_2, q_3, q_4, q_5 = \text{halt}\}$$
$$D = \{L, R, N\}$$

The SFM for the TM is shown in Table 4.10.

Table 4.10 SFM for a TM that compares two positive integers

S \ I	a	,	*	$\not b$	G	L	E
q_0	$*q_1R$	q_4R	R	—	—	—	—
q_1	R	q_2R	—	—	—	—	—
q_2	$*q_3L$	—	R	G_tq_5N	—	—	—
q_3	L	L	L	q_0R	—	—	—
q_4	L_tq_5N	—	R	E_qq_5N	—	—	—
q_5	—	—	—	—	—	—	—

Initially, the TM is in state q_0, which replaces one a from m by '*', and makes a transition to a new state q_1. State q_1 is responsible for finding the beginning of the next number n; on accomplishing this, q_1 makes transition to q_2. State q_2 replaces one a from n by '*' to match the one that we have replaced from m earlier. Essentially, we subtract 1 from both the numbers.

Now, if the TM replaces one a from m by a '*', but there is no a left to be replaced in n, then it means that in state q_2 the TM does not find any a; it instead reads a blank character $\not b$. In this case, the TM generates the output G_t to indicate that the first number m is greater than the second number n.

Let us consider another scenario. If there is no a left in the number m, that is, the TM reads comma ',' while in state q_0; then, it changes to state q_4. State q_4 now needs to check whether there is any a left in the second number n. If the TM finds an a in n while in state

q_4, then it means that the second number is greater, or that the first number m is less
the second number n. Hence, the TM generates the output L_t.

On the other hand, if the TM reads a blank character $\not b$ in state q_4, then the TM
ates output as E_q, indicating that both numbers are equal, and there are no more a's
be replaced in either of them.

The TM thus represents the numbers in unary format and performs the compari
repetitive subtraction.

Simulation

1. Let us simulate the working of the TM that we have constructed for $m = 3$, that i
'aaa' and $n = 2$, that is, $n =$ 'aa'.

$\not b\, a\, a\, a\, ,\, a\, a\, \not b$ $\uparrow$ q_0	initial configuration
$\not b * a\, a\, ,\, a\, a\, \not b$ $\uparrow$ q_1	$\delta\,(q_0, a) = (* \; q_1 \; R)$
$\not b * a\, a\, ,\, a\, a\, \not b$ $\uparrow$ q_1	$\delta\,(q_1, a) = (R)$
$\not b * a\, a\, ,\, a\, a\, \not b$ $\uparrow$ q_1	$\delta\,(q_1, a) = (R)$
$\not b * a\, a\, ,\, a\, a\, \not b$ $\uparrow$ q_1	$\delta\,(q_1, ,) = (q_2 \; R)$
$\not b * a\, a\, ,\, * \, a\, \not b$ $\uparrow$ q_2	$\delta\,(q_2, a) = (* \; q_3 \; L)$
$\not b * a\, a\, ,\, * \, a\, \not b$ $\uparrow$ q_3	$\delta\,(q_3, ,) = (L)$
$\not b * a\, a\, ,\, * \, a\, \not b$ $\uparrow$ q_3	$\delta\,(q_3, a) = (L)$
$\not b * a\, a\, ,\, * \, a\, \not b$ $\uparrow$ q_3	$\delta\,(q_3, a) = (L)$
$\not b * a\, a\, ,\, * \, a\, \not b$ $\uparrow$ q_3	$\delta\,(q_3, *) = (L)$
$\not b * a\, a\, ,\, * \, a\, \not b$ $\uparrow$ q_3	$\delta\,(q_3, \not b) = (q_0 \; R)$
$\not b * a\, a\, ,\, * \, a\, \not b$ $\uparrow$ q_0	$\delta\,(q_0, *) = (R)$
$\not b * * \, a\, ,\, * \, a\, \not b$ $\uparrow$ q_1	$\delta\,(q_0, a) = (* \; q_1 \; R)$
$\not b * * \, a\, ,\, * \, a\, \not b$ $\uparrow$ q_1	$\delta\,(q_1, a) = (R)$

$\not b * * a , * a \not b$ $\uparrow$ q_2	$\delta (q_1, ,) = (q_2 R)$
$\not b * * a , * a \not b$ $\uparrow$ q_2	$\delta (q_2, *) = (R)$
$\not b * * a , * * \not b$ $\uparrow$ q_3	$\delta (q_2, a) = (* q_3 L)$
$\not b * * a , * * \not b$ $\uparrow$ q_3	$\delta (q_3, *) = (L)$
$\not b * * a , * * \not b$ $\uparrow$ q_3	$\delta (q_3, ,) = (L)$
$\not b * * a , * * \not b$ $\uparrow$ q_3	$\delta (q_3, a) = (L)$
$\not b * * a , * * \not b$ $\uparrow$ q_3	$\delta (q_3, *) = (L)$
$\not b * * a , * * \not b$ $\uparrow$ q_3	$\delta (q_3, *) = (L)$
$\not b * * a , * * \not b$ $\uparrow$ q_0	$\delta (q_3, \not b) = (q_0 R)$
$\not b * * a , * * \not b$ $\uparrow$ q_0	$\delta (q_0, *) = (R)$
$\not b * * a , * * \not b$ $\uparrow$ q_0	$\delta (q_0, *) = (R)$
$\not b * * * , * * \not b$ $\uparrow$ q_1	$\delta (q_0, a) = (* q_1 R)$
$\not b * * * , * * \not b$ $\uparrow$ q_2	$\delta (q_1, ,) = (q_2 R)$
$\not b * * * , * * \not b$ $\uparrow$ q_2	$\delta (q_2, *) = (R)$
$\not b * * * , * * \not b$ $\uparrow$ q_2	$\delta (q_2, *) = (R)$
$\not b * * * , * * G_t$ $\uparrow$ q_5	$\delta (q_2, \not b) = (G_t q_5 N)$

Thus, the output of the TM indicates that $m = 3$ is greater than $n = 2$.

2. Let us simulate the working of the TM for $m = 0$ (i.e., ϵ), and $n = 1$ (i.e., 'a'). As the first number is 0, the head points to the separator symbol ',', to begin with.

$\not b , a \not b$ $\uparrow$ q_0	initial configuration

$\flat , a \flat$ $\quad\quad \delta (q_0, ,) = (q_4 R)$
$\uparrow$
q_4
$\flat , L_t \flat$ $\quad\quad \delta (q_4, a) = (L_t q_5 N)$
$\uparrow$
q_5

The output L_t indicates that $m = 0$ is less than $n = 1$.

3. Let us consider the case when both numbers are equal: let $m = n = 1 =$ 'a'.

$\flat a , a \flat$ $\quad\quad$ initial configuration
$\uparrow$
q_0
$\flat * , a \flat$ $\quad\quad \delta (q_0, a) = (* q_1 R)$
$\uparrow$
q_1
$\flat * , a \flat$ $\quad\quad \delta (q_1, ,) = (q_2 R)$
$\uparrow$
q_2
$\flat * , * \flat$ $\quad\quad \delta (q_2, a) = (* q_3 L)$
$\uparrow$
q_3
$\flat * , * \flat$ $\quad\quad \delta (q_3, ,) = (L)$
$\uparrow$
q_3
$\flat * , * \flat$ $\quad\quad \delta (q_3, *) = (L)$
$\uparrow$
q_3
$\flat * , * \flat$ $\quad\quad \delta (q_3, \flat) = (q_0 R)$
$\uparrow$
q_0
$\flat * , * \flat$ $\quad\quad \delta (q_0, *) = (R)$
$\uparrow$
q_0
$\flat * , * \flat$ $\quad\quad \delta (q_0, ,) = (q_4 R)$
$\uparrow$
q_4
$\flat * , * \flat$ $\quad\quad \delta (q_4, *) = (R)$
$\uparrow$
q_4
$\flat * , * E_q$ $\quad\quad \delta (q_4, \flat) = (E_q q_5 N)$
$\uparrow$
q_5

Thus, m and n are equal, which is indicated by the output E_q.

Example 4.9 Design a TM that performs the addition of two unary numbers.

Solution Let us consider the pair of numbers expressed in unary form using a symbol 'a', that is, a number x is represented by x consecutive a's. Let the initial configuration of the machine be as shown in Fig. 4.12(a). As both the unary numbers are represented using the letter 'a', they are separated by the delimiter 'c' on the tape. The initial state of the TM is α, as specified in Fig. 4.12(a). The TG for this TM is shown in Fig. 4.12(b).

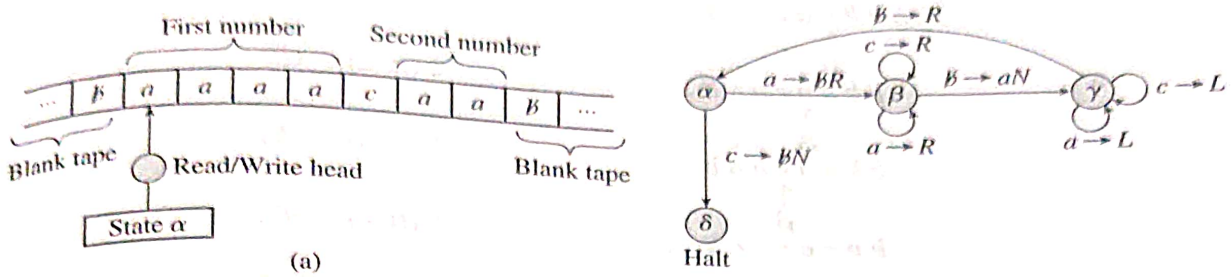


Figure 4.12 TM that performs addition of two unary numbers (a) Initial configuration (b) TG for TM that adds two unary numbers

Algorithm

Addition is nothing but concatenation. This is exactly similar to what is taught in pre-primary level mathematics. To achieve '$m + n$', let us represent m and n by some object such as blue balls and then perform aggregation. Put m blue balls into a bucket and put n blue balls into the same bucket; then count the total number of blue balls in the bucket. It gives the value after addition.

Replace each a from the first number, that is, the string of a's on the left side of c by a blank character $\not b$ and add one a after the string of a's representing the second number. In this way, after completion of addition, the string of a's equal to the result of addition resides on the right side of c. At the end, replace the c with a blank character $\not b$.

For this TM, we have

$$I = \{a, \not b, c\}$$
$$S = \{\alpha, \beta, \gamma, \delta = \text{halt}\}$$
$$D = \{L, R, N\}$$

The simplified functional matrix for the TM is given in Table 4.11.

Table 4.11 SFM for a TM that adds two unary numbers

S \ I	a	$\not b$	c
α	$\not b \beta R$	—	$\not b \delta N$
β	R	aγN	R
γ	L	αR	L

Note: In this example, entries for δ state are not shown because they are null entries, as it is a halt state. In all previous examples, we have used '-' (hyphens) to indicate them as null/unspecified entries.

Simulation

Let us simulate the working of the TM for the following example:

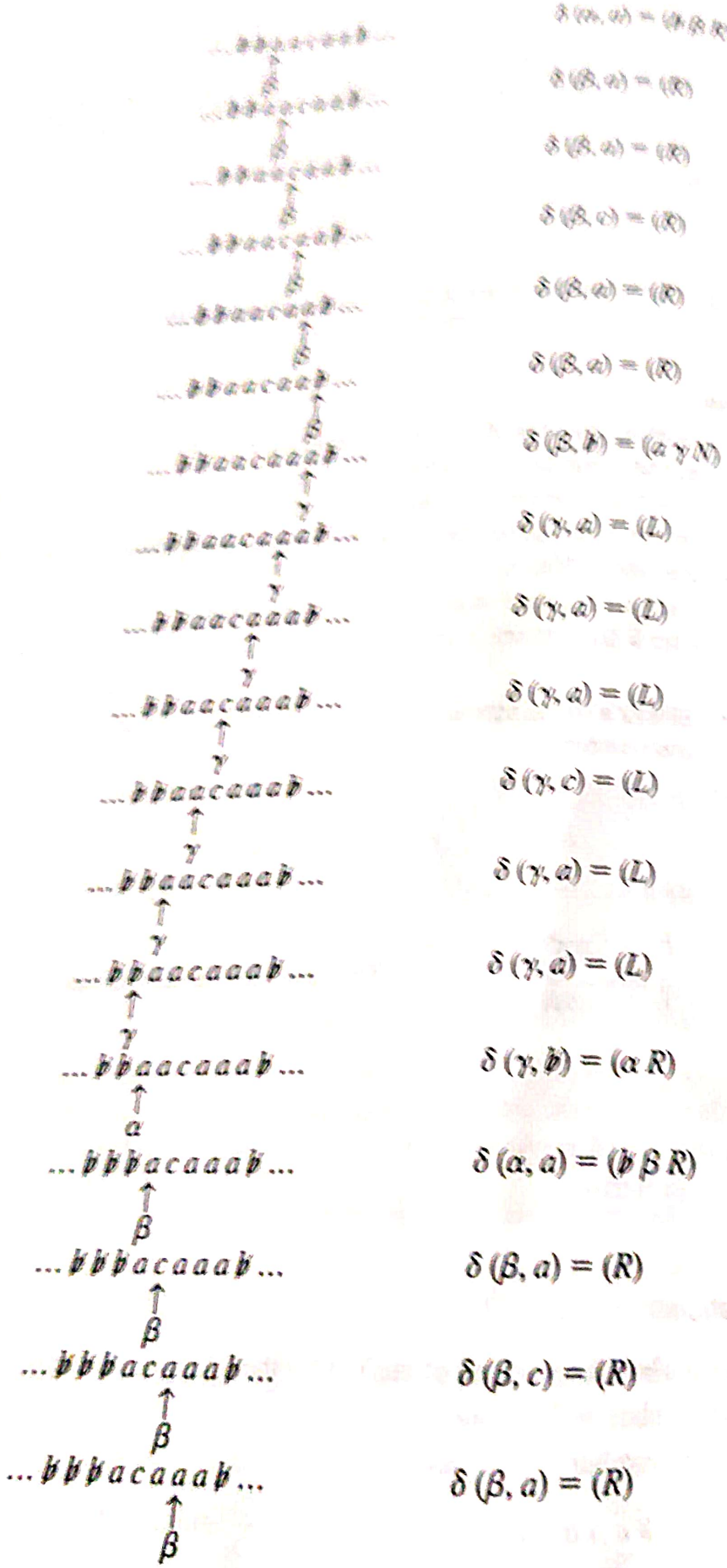
First number = 3 = '*aaa*'

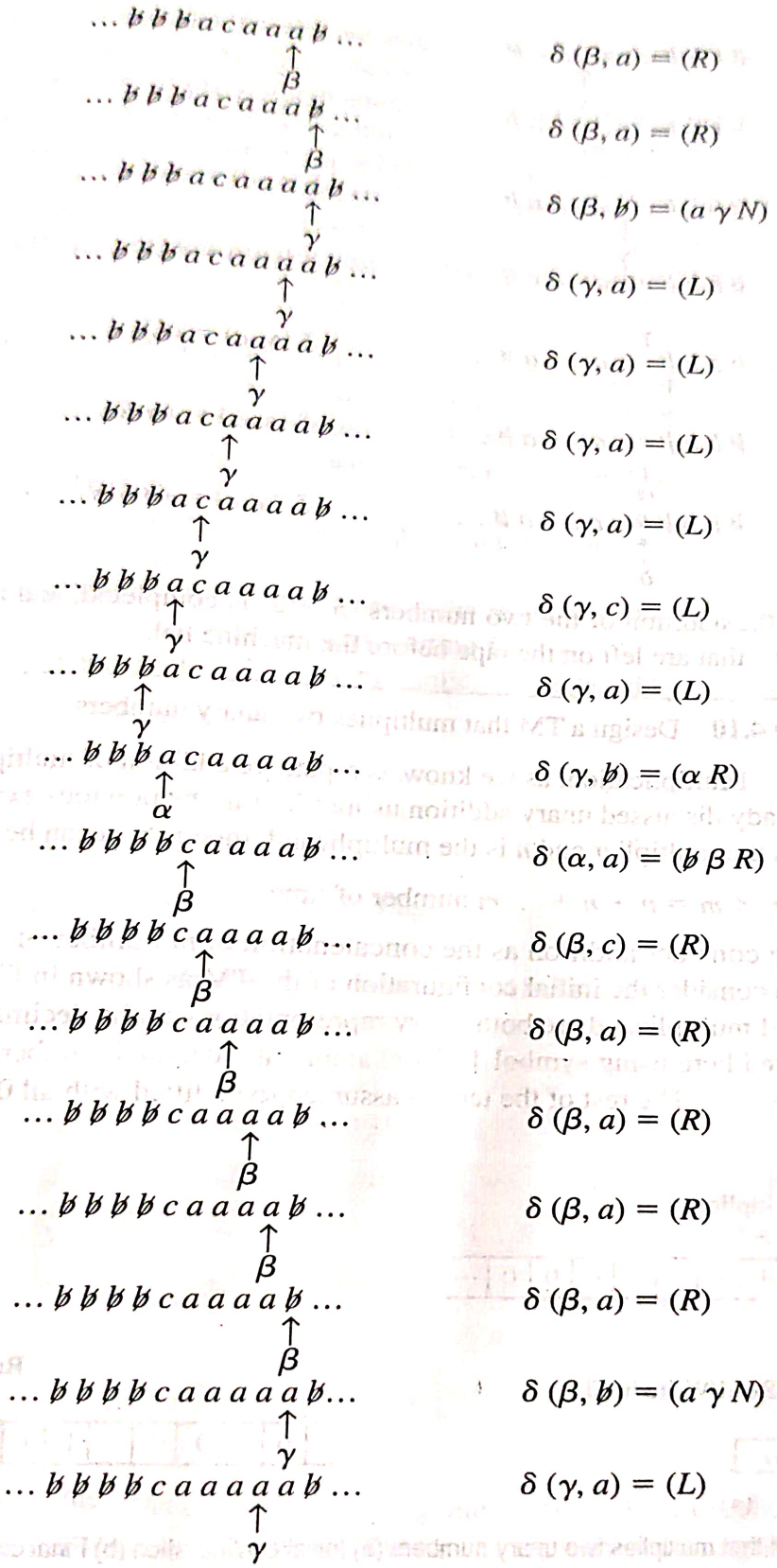
Second number = 2 = '*aa*'

$\ldots \not b\ a\ a\ a\ c\ a\ a\ \not b \ldots$ initial configuration

$\uparrow$

α


δ (q, a) = (b, R, q)
δ (q, a) = (R, q)
δ (q, a) = (R, q)
δ (q, c) = (R, q)
δ (q, a) = (R, q)
δ (q, a) = (R, q)
δ (q, b) = (a, γ, N)
δ (γ, a) = (L, γ)
δ (γ, a) = (L, γ)
δ (γ, a) = (L, γ)
δ (γ, c) = (L, γ)
δ (γ, a) = (L, γ)
δ (γ, a) = (L, γ)
δ (γ, b) = (a, R, q)
δ (a, a) = (b, β, R)
δ (β, a) = (R, β)
δ (β, c) = (R, β)
δ (β, a) = (R, β)


... b b b a c a a a b ...
β
δ (β, a) = (R)
... b b b a c a a a b ...
β
δ (β, a) = (R)
... b b b a c a a a a b ...
γ
δ (β, b) = (a γ N)
... b b b a c a a a a b ...
γ
δ (γ, a) = (L)
... b b b a c a a a a b ...
γ
δ (γ, a) = (L)
... b b b a c a a a a b ...
γ
δ (γ, a) = (L)
... b b b a c a a a a b ...
γ
δ (γ, a) = (L)
... b b b a c a a a a b ...
γ
δ (γ, c) = (L)
... b b b a c a a a a b ...
γ
δ (γ, a) = (L)
... b b b a c a a a a b ...
α
δ (γ, b) = (α R)
... b b b b c a a a a b ...
β
δ (α, a) = (b β R)
... b b b b c a a a a b ...
β
δ (β, c) = (R)
... b b b b c a a a a b ...
β
δ (β, a) = (R)
... b b b b c a a a a b ...
β
δ (β, a) = (R)
... b b b b c a a a a b ...
β
δ (β, a) = (R)
... b b b b c a a a a b ...
β
δ (β, a) = (R)
... b b b b c a a a a a b...
γ
δ (β, b) = (a γ N)
... b b b b c a a a a a b ...
γ
δ (γ, a) = (L)

$$\ldots \not b \not b \not b \not b c a a a a a \not b \ldots \qquad \delta(\gamma, a) = (L)$$
$$\ldots \not b \not b \not b \not b c a a a a \not b \ldots \qquad \delta(\gamma, a) = (L)$$
$$\ldots \not b \not b \not b \not b c a a a a \not b \ldots \qquad \delta(\gamma, a) = (L)$$
$$\ldots \not b \not b \not b \not b c a a a a \not b \ldots \qquad \delta(\gamma, a) = (L)$$
$$\ldots \not b \not b \not b \not b c a a a a \not b \ldots \qquad \delta(\gamma, c) = (L)$$
$$\ldots \not b \not b \not b \not b c a a a a \not b \ldots \qquad \delta(\gamma, \not b) = (\alpha\ R)$$
$$\ldots \not b \not b \not b \not b \not b a a a a \not b \ldots \qquad \delta(\alpha, c) = (\not b\ \delta\ N)$$

Thus, the addition of the two numbers '3 + 2' is completed, and that is indicated b the five a's that are left on the tape before the machine halts.

Example 4.10 Design a TM that multiplies two unary numbers.

Solution Multiplication, as we know, is repetitive addition of multiplicand to itself. W have already discussed unary addition using a TM in the previous example.

If m is the multiplier and n is the multiplicand, then $n \times m$ can be viewed as:

$$n \times m = n + n + \ldots m \text{ number of times}$$

Thus, we consider addition as the concatenation of m number of n's.

Let us consider the initial configuration of the TM as shown in Fig. 4.13(a). The mu tiplier and multiplicand are both unary representations of the decimal numbers. They ar represented here using symbol 1; for example, the decimal number 2 in unary format written as '11'. The rest of the tape is assumed to be filled with all 0's instead of $\not b$'s. Th

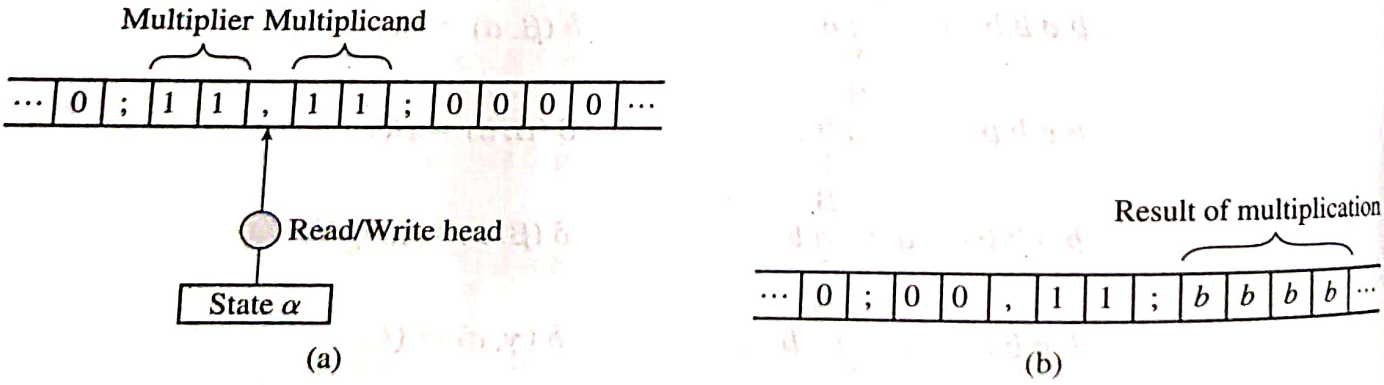


Figure 4.13 TM that multiplies two unary numbers (a) Initial configuration (b) Final configuration of TM (for the operation $2 \times 2 = 4$).

multiplier and multiplicand are separated by a comma ',' and delimited at both the ends by semicolons ';'. The head points to the separator symbol ',' initially.

The result of the multiplication is written after the right end-marker semicolon (;). We have an example final configuration of the TM for a sample multiplication (2×2), as shown in Fig. 4.13(b). Observe how the result is written and where it is written onto the tape. The result is also represented in unary format but using the symbol b. The multiplier at the end gets replaced with 0's. The algorithm thus is a destructive one, as it modifies the parameters sent to it (in this case, the parameter modified is the multiplier).

Algorithm

1. Replace one '1' of the multiplier by '0', that is, subtract one from the multiplier.
2. To add the multiplicand to the result area, that is, beyond the right end-marker ';' replace one '1' of the multiplicand by some symbol, say 'a'
3. Find the end of the result where '0' can be found, replace it with symbol 'b' (result is represented by all 'b's)
4. Repeat the aforementioned steps 2 and 3 till all the '1's in the multiplicand are all replaced by 'a's
5. The multiplicand is added once to the result area; hence, reset the multiplicand to all '1's again and repeat the steps starting from 1.
6. Stop when all the '1's in the multiplier are replaced by all '0's

For this TM, we have:

$I = \{0, 1, a, b, ;, ,\}$
$S = \{\alpha, \beta, \gamma, \delta, \epsilon, f = \text{halt}\}$
$D = \{L, R, N\}$

The SFM for the TM is shown in Table 4.12.

Table 4.12 SFM for a TM that multiplies two unary numbers

S \ I	0	1	a	b	;	,
α	L	$0\beta R$	—	—	ϕN	L
β	R	aR	—	—	γL	R
γ	—	—	$1\,\delta\,R$	—	R	L
δ	$b\epsilon L$	R	—	R	R	—
ϵ	L	L	$1\,\delta\,R$	L	L	αN
ϕ	—	—	—	—	—	—

In state α, the TM starts by replacing one '1' from the multiplier by 0, that is, reducing the multiplier by 1. In state β, the TM moves right by replacing all 1's from the multiplicand by a's, and changes the state to γ once the right end-marker ';' is found. States γ,

δ, and ϵ are responsible for concatenating the multiplicand to the right end once. In the process, all the 1's in the multiplicand that were replaced by all a's are replaced again by 1's. The concatenated result is represented in unary form using the symbol b. The TM state f is entered once all the multiplier 1's are replaced by 0's, which means that the TM halts when the multiplicand is added (concatenated) to itself multiplier number of times. Let us see a simulation of this TM for a sample multiplication.

Simulation

Let us simulate the working of this TM for the following:

Multiplier = $m = 2 =$ '11'

Multiplicand = $n = 2 =$ '11'.

$\ldots 0\,;1\,1\,,1\,1\,;0\,0\,0\,0\ldots$ initial configuration

$\uparrow$

α

$\ldots 0\,;1\,1\,,1\,1\,;0\,0\,0\,0\ldots$ $\delta(\alpha, ,) = (L)$

$\uparrow$

α

$\ldots 0\,;1\,0\,,1\,1\,;0\,0\,0\,0\ldots$ $\delta(\alpha, 1) = (0\,\beta\,R)$

$\uparrow$

β

(Decremented multiplier by 1)

$\ldots 0\,;1\,0\,,1\,1\,;0\,0\,0\,0\ldots$ $\delta(\beta, ,) = (R)$

$\uparrow$

β

(Concatenation of the multiplicand to the result area begins from here...)

$\ldots 0\,;1\,0\,,a\,1\,;0\,0\,0\,0\ldots$ $\delta(\beta, 1) = (a\,R)$

$\uparrow$

β

(Replacing all '1's by 'a's from multiplicand is for performing concatenation the multiplicand to the result)

$\ldots 0\,;1\,0\,,a\,a\,;0\,0\,0\,0\ldots$ $\delta(\beta, 1) = (a\,R)$

$\uparrow$

β

$\ldots 0\,;1\,0\,,a\,a\,;0\,0\,0\,0\ldots$ $\delta(\beta, ;) = (\gamma\,L)$

$\uparrow$

γ

$\ldots 0\,;1\,0\,,a\,1\,;0\,0\,0\,0\ldots$ $\delta(\gamma, a) = (1\,\delta\,R)$

$\uparrow$

δ

(This step again replaces the 'a's in the multiplicand with 1; therefore, at concatenation in the result area, the multiplicand will be restored as all '1's)

$\ldots 0\,;1\,0\,,a\,1\,;0\,0\,0\,0\ldots$ $\delta(\delta, ;) = (R)$

$\uparrow$

δ

$\ldots 0\,;1\,0\,,a\,1\,;b\,0\,0\,0\ldots$ $\delta(\delta, 0) = (b\,\epsilon\,L)$

$\uparrow$

ϵ

$\ldots 0\,;1\,0\,,a\,1\,;b\,0\,0\,0\ldots$ $\delta(\epsilon, ;) = (L)$

$\uparrow$

ϵ

$\ldots 0 ; 1 0 , \underset{\epsilon}{a} 1 ; b 0 0 0 \ldots$ $\quad \delta(\epsilon, 1) = (L)$

$\ldots 0 ; 1 0 , \underset{\delta}{1} 1 ; b 0 0 0 \ldots$ $\quad \delta(\epsilon, a) = (1\ \delta\ R)$

(Multiplicand is completely replaced by 1's again)

$\ldots 0 ; 1 0 , 1 1 ; \underset{\delta}{b} 0 0 0 \ldots$ $\quad \delta(\delta, 1) = (R)$

$\ldots 0 ; 1 0 , 1 1 ; b \underset{\delta}{0} 0 0 \ldots$ $\quad \delta(\delta, ;) = (R)$

$\ldots 0 ; 1 0 , 1 1 ; b 0 \underset{\delta}{0} 0 \ldots$ $\quad \delta(\delta, b) = (R)$

$\ldots 0 ; 1 0 , 1 1 ; b b \underset{\epsilon}{0} 0 \ldots$ $\quad \delta(\delta, 0) = (b\ \epsilon\ L)$

(Concatenation of multiplicand to the result area is complete. Result is represented as a string of b's)

$\ldots 0 ; 1 0 , 1 1 ; b \underset{\epsilon}{b} 0 0 \ldots$ $\quad \delta(\epsilon, b) = (L)$

$\ldots 0 ; 1 0 , 1 1 ; \underset{\epsilon}{b} b 0 0 \ldots$ $\quad \delta(\epsilon, ;) = (L)$

$\ldots 0 ; 1 0 , \underset{\epsilon}{1} 1 ; b b 0 0 \ldots$ $\quad \delta(\epsilon, 1) = (L)$

$\ldots 0 ; 1 0 \underset{\epsilon}{,} 1 1 ; b b 0 0 \ldots$ $\quad \delta(\epsilon, 1) = (L)$

$\ldots 0 ; 1 0 \underset{\alpha}{,} 1 1 ; b b 0 0 \ldots$ $\quad \delta(\epsilon, ,) = (\alpha\ N)$

$\ldots 0 ; 1 \underset{\alpha}{0} , 1 1 ; b b 0 0 \ldots$ $\quad \delta(\alpha, ,) = (L)$

$\ldots 0 ; \underset{\alpha}{1} 0 , 1 1 ; b b 0 0 \ldots$ $\quad \delta(\alpha, 0) = (L)$

$\ldots 0 ; 0 \underset{\beta}{0} , 1 1 ; b b 0 0 \ldots$ $\quad \delta(\alpha, 1) = (0\ \beta\ R)$

(Decremented the multiplier by 1 again; the second iteration begins)

$\ldots 0 ; 0 0 , \underset{\beta}{1} 1 ; b b 0 0 \ldots$ $\quad \delta(\beta, 0) = (R)$

$\ldots 0 ; 0 0 , 1 \underset{\beta}{1} ; b b 0 0 \ldots$ $\quad \delta(\beta, ,) = (R)$

(Concatenation of the multiplicand, for the second time, to the result area here; since multiplication is repetitive addition, we use repetitive concate here)

... 0 ; 0 0 , a 1 ; b b 0 0 ... $\quad \delta(\beta, 1) = (a\ R)$

(Replacing all '1's by '*a*'s from multiplicand is for performing concatena the multiplicand to the result)

... 0 ; 0 0 , a a ; b b 0 0 ... $\quad \delta(\beta, 1) = (a\ R)$

... 0 ; 0 0 , a a ; b b 0 0 ... $\quad \delta(\beta, ;) = (\gamma\ L)$

... 0 ; 0 0 , a 1 ; b b 0 0 ... $\quad \delta(\gamma, a) = (1\ \delta\ R)$

(Once it is concatenated to the result area, the *a* in the multiplicand is rep again by 1)

... 0 ; 0 0 , a 1 ; b b 0 0 ... $\quad \delta(\delta, ;) = (R)$

... 0 ; 0 0 , a 1 ; b b 0 0 ... $\quad \delta(\delta, b) = (R)$

... 0 ; 0 0 , a 1 ; b b 0 0 ... $\quad \delta(\delta, b) = (R)$

... 0 ; 0 0 , a 1 ; b b b 0 ... $\quad \delta(\delta, 0) = (b\ \epsilon\ L)$

... 0 ; 0 0 , a 1 ; b b b 0 ... $\quad \delta(\epsilon, b) = (L)$

... 0 ; 0 0 , a 1 ; b b b 0 ... $\quad \delta(\epsilon, b) = (L)$

... 0 ; 0 0 , a 1 ; b b b 0 ... $\quad \delta(\epsilon, ;) = (L)$

... 0 ; 0 0 , a 1 ; b b b 0 ... $\quad \delta(\epsilon, 1) = (L)$

... 0 ; 0 0 , 1 1 ; b b b 0 ... $\quad \delta(\epsilon, a) = (1\ \delta\ R)$

(Multiplicand is completely replaced by 1's again)

... 0 ; 0 0 , 1 1 ; b b b 0 ... $\quad \delta(\delta, 1) = (R)$

$\ldots 0\,;\,0\,0\,,\,1\,1\,;\,\underset{\uparrow}{b}\,b\,b\,0\ldots$ $\quad \delta\,(\delta,\,;) = (R)$

$\ldots 0\,;\,0\,0\,,\,1\,1\,;\,b\,\underset{\uparrow}{b}\,b\,0\ldots$ $\quad \delta\,(\delta,\,b) = (R)$

$\ldots 0\,;\,0\,0\,,\,1\,1\,;\,b\,b\,\underset{\uparrow}{b}\,0\ldots$ $\quad \delta\,(\delta,\,b) = (R)$

$\ldots 0\,;\,0\,0\,,\,1\,1\,;\,b\,b\,b\,\underset{\uparrow}{0}\ldots$ $\quad \delta\,(\delta,\,b) = (R)$

$\ldots 0\,;\,0\,0\,,\,1\,1\,;\,b\,b\,b\,\underset{\uparrow}{b}\ldots$ $\quad \delta\,(\delta,\,0) = (b\,\epsilon\,L)$

(Concatenation of multiplicand to the result area is completed once again. The result is represented as a string of b's)

$\ldots 0\,;\,0\,0\,,\,1\,1\,;\,b\,b\,b\,\underset{\uparrow}{b}\ldots$ $\quad \delta\,(\epsilon,\,b) = (L)$

$\ldots 0\,;\,0\,0\,,\,1\,1\,;\,b\,b\,\underset{\uparrow}{b}\,b\ldots$ $\quad \delta\,(\epsilon,\,b) = (L)$

$\ldots 0\,;\,0\,0\,,\,1\,1\,;\,b\,\underset{\uparrow}{b}\,b\,b\ldots$ $\quad \delta\,(\epsilon,\,b) = (L)$

$\ldots 0\,;\,0\,0\,,\,1\,1\,;\,\underset{\uparrow}{b}\,b\,b\,b\ldots$ $\quad \delta\,(\epsilon,\,;) = (L)$

$\ldots 0\,;\,0\,0\,,\,1\,\underset{\uparrow}{1}\,;\,b\,b\,b\,b\ldots$ $\quad \delta\,(\epsilon,\,1) = (L)$

$\ldots 0\,;\,0\,0\,,\,\underset{\uparrow}{1}\,1\,;\,b\,b\,b\,b\ldots$ $\quad \delta\,(\epsilon,\,1) = (L)$

$\ldots 0\,;\,0\,\underset{\uparrow}{0}\,,\,1\,1\,;\,b\,b\,b\,b\ldots$ $\quad \delta\,(\epsilon,\,,) = (\alpha\,N)$

$\ldots 0\,;\,\underset{\uparrow}{0}\,0\,,\,1\,1\,;\,b\,b\,b\,b\ldots$ $\quad \delta\,(\alpha,\,,) = (L)$

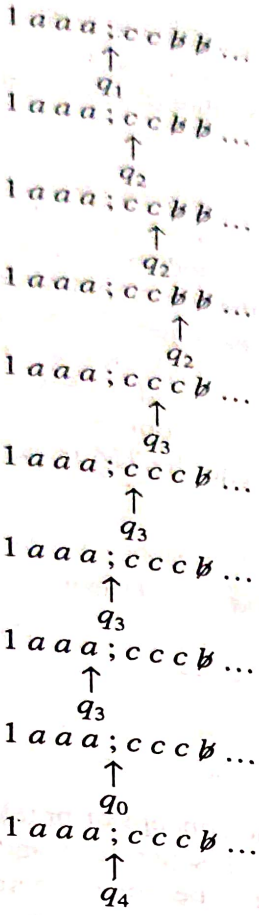
$\ldots 0\,;\,\underset{\uparrow}{0}\,0\,,\,1\,1\,;\,b\,b\,b\,b\ldots$ $\quad \delta\,(\alpha,\,0) = (L)$

$\ldots 0\,;\,\underset{\uparrow}{0}\,0\,,\,1\,1\,;\,b\,b\,b\,b\ldots$ $\quad \delta\,(\alpha,\,0) = (L)$

$\ldots \underset{\uparrow}{0}\,;\,0\,0\,,\,1\,1\,;\,b\,b\,b\,b\ldots$ $\quad \delta\,(\alpha,\,;) = (\phi\,N)$

(TM halts; multiplication is complete)

Thus, we see that the result of the multiplication: 2×2 is '*bbbb*', that is, 4.



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

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

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

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

$\delta(q_2, b) = (c\ q_3\ L)$

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

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

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

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

$\delta(q_0, ;) = (q_4\ N)$

The TM halts with the result *ccc*, which is the expected value, that is, 3.

We could also replace all the *a*'s by 0's again, in order to retain the input parameter, so as to have a non-destructing algorithm.

4.7 COMPLEXITY OF A TURING MACHINE

The complexity of a TM is directly proportional to the size of the functional matrix. In other words, we can say that the complexity of a TM depends on the number of symbols that are being used and the number of states of the TM. Hence

$$\text{Complexity of a TM} = |\Gamma| \times |Q| \quad (\text{or } |I| \times |S|),$$

where, $|\Gamma|$ = Cardinality of tape alphabet (i.e., number of tape symbols),
and $|Q|$ = Number of states of the TM.

Let us consider Example 4.13 in which we designed a TM that finds $\log_2(n)$, where n is a perfect power of 2, and is represented in binary format. For this example, we have

$\Gamma = \{1, 0, a, c, ;, b\}$

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

Therefore, the complexity of the TM = $|\Gamma| \times |Q| = 6 \times 5 = 30$

Thus, while designing a TM, we must ensure that it has minimum complexity, that is, we should design a TM such that it has lesser number of input symbols and states.

4.8 COMPOSITE AND ITERATIVE TURING MACHINES

Two or more Turing machines can be combined to solve a complex problem, such that the output of one TM forms the input to the next TM, and so on. This is called *composition*.

For realizing a composite TM (or a CTM), the functional matrices of the component TMs are combined by re-labeling the symbols, as required, and suitably branching to an appropriate state rather than the halt state at the completion of the performance of each component TM. Figure 4.17(a) depicts a composition of n TMs.

Another way of having a combination TM is by applying its own output as input repetitively. This is called *iteration* or *recursion*, and the TM is said to be an iterative TM (or ITM). For an example, refer to Fig. 4.17(b).

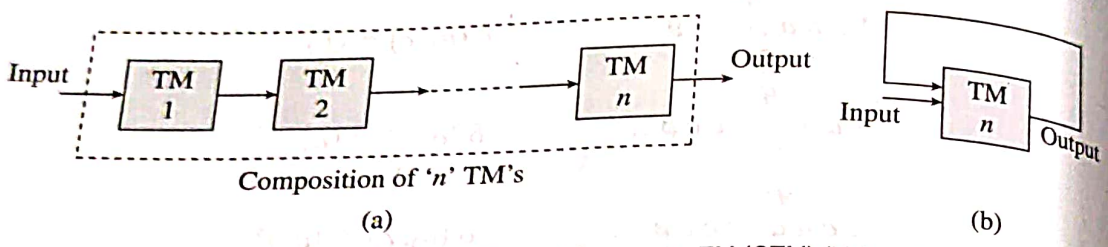


Figure 4.17 Composite and iterative TMs (a) Composite TM (CTM) (b) Iterative TM (ITM)

The idea of a composite TM gives rise to the concept of breaking a complicated job into a number of smaller jobs, implementing each separately, and then combining them together to get the answer for the job required to be done. Therefore, we can divide a problem into simple jobs and design different TMs for each job. This is a typical *separation of concerns* achieved in software development; it is analogous to the function composition that we know from discrete mathematics: $f \circ g\,(x) = f\,(g\,(x))$. The output of $g\,(x)$ is given as input to function f. In a way, modular programming can be considered to be influenced by CTM.

Functionally, most of the TMs that we have implemented earlier in the chapter, for example, multiplication as repetitive addition, division as repetitive subtraction, and so on, are examples of iterative TMs.

Example 4.14 *Design a TM to find the value of n^2, where n is any integer ≥ 0.*

Solution *Let us consider the initial configuration to be as shown in Fig. 4.18 (a).*

The number n is represented in unary form using 0's, as shown in Fig. 4.18(a). We find n^2 is in terms of the multiplication '$n \times n$'. This involves copying n after the comma ',' onto the tape, and using that as the multiplicand—represented in unary form using symbol 1—as shown in Fig. 4.18(b). We can then perform the multiplication '$n \times n$', as we have done in Example 4.10. Figure 4.18(b) is the initial configuration for the TM that performs multiplication as we have seen earlier.

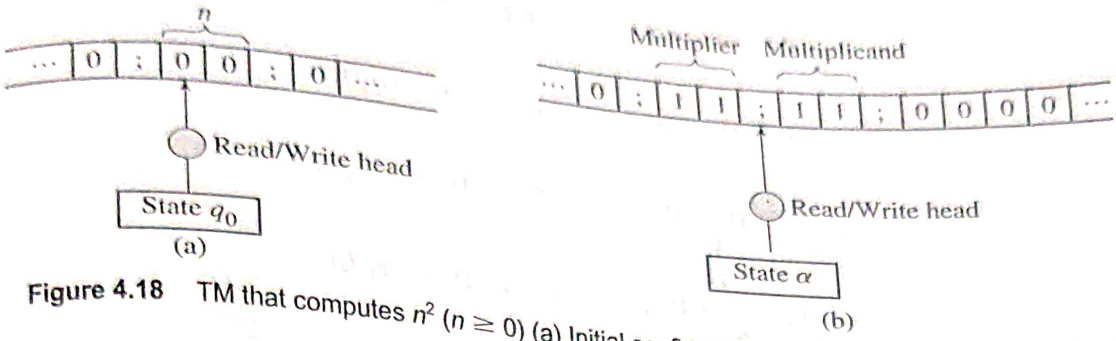


Figure 4.18 TM that computes n^2 ($n \geq 0$) (a) Initial configuration (b) Configuration after copying n onto tape

Observe that the TM that computes n^2 can be considered as a composition of two TMs. The first TM prepares the output in the form suggested in Figure 4.18(b), which can be used as input for the second TM that performs multiplication. The second TM is similar to the one that we have already designed in Example 4.10; the SFM for this TM is described in Table 4.12 with α as the initial state. Hence, we need to design the first TM, which converts the initial configuration shown in Fig. 4.18(a) into the form that can be used as input by the second TM, whose initial configuration is shown in Fig. 4.18(b).

Algorithm

1. Replace one 0 at a time by symbol 1 and copy it after the right end-marker, ';'.
2. Repeat this process till all the 0's from n are replaced by 1's.
3. Thus, another copy of n gets prepared after the right end-marker, ';'. This is done to store n as the multiplier as well as multiplicand onto the tape.
4. Now, replace the end-marker, ';', by a comma, ',', and add another end-marker, ';', at the end of the copy of n created at the right end.

The SFM for the required TM is shown in Table 4.16.

Table 4.16 SFM for the TM that prepares input for multiplication

S \ I	0	;	1	,
q_0	$1q_1R$	$,q_5R$	—	—
q_1	R	q_2R	—	—
q_2	$1q_3L$	—	R	—
q_3	—	q_4L	L	—
q_4	L	—	q_0R	—
q_5	$;q_6L$	—	R	—
q_6	—	—	L	αN

Simulation

Let us simulate the working of the TM for $n = 2$.

$$\begin{array}{ll} \ldots 0\ ;\ 0\ 0\ ;\ 0\ 0\ \ldots & \text{initial configuration} \\ \quad\uparrow & \\ \quad q_0 & \\ \ldots 0\ ;\ 1\ 0\ ;\ 0\ 0\ \ldots & \delta\,(q_0, 0) = (1\ q_1\ R) \\ \quad\quad\uparrow & \\ \quad\quad q_1 & \end{array}$$

$$\begin{array}{ll}
\ldots 0\,;\,1\,0\,;\,0\,0\ldots & \delta\,(q_1,\,0) = (R) \\
\uparrow & \\
q_1 & \\
\ldots 0\,;\,1\,0\,;\,0\,0\ldots & \delta\,(q_1,\,;) = (q_2\,R) \\
\uparrow & \\
q_2 & \\
\ldots 0\,;\,1\,0\,;\,1\,0\ldots & \delta\,(q_2,\,0) = (1\,q_3\,L) \\
\uparrow & \\
q_3 & \\
\ldots 0\,;\,1\,0\,;\,1\,0\ldots & \delta\,(q_3,\,;) = (q_4\,L) \\
\uparrow & \\
q_4 & \\
\ldots 0\,;\,1\,0\,;\,1\,0\ldots & \delta\,(q_4,\,0) = (L) \\
\uparrow & \\
q_4 & \\
\ldots 0\,;\,1\,0\,;\,1\,0\ldots & \delta\,(q_4,\,1) = (q_0\,R) \\
\uparrow & \\
q_0 & \\
\ldots 0\,;\,1\,1\,;\,1\,0\ldots & \delta\,(q_0,\,0) = (1\,q_1\,R) \\
\uparrow & \\
q_1 & \\
\ldots 0\,;\,1\,1\,;\,1\,0\ldots & \delta\,(q_1,\,;) = (q_2\,R) \\
\uparrow & \\
q_2 & \\
\ldots 0\,;\,1\,1\,;\,1\,0\ldots & \delta\,(q_2,\,1) = (R) \\
\uparrow & \\
q_2 & \\
\ldots 0\,;\,1\,1\,;\,1\,1\,0\ldots & \delta\,(q_2,\,0) = (1\,q_3\,L) \\
\uparrow & \\
q_3 & \\
\ldots 0\,;\,1\,1\,;\,1\,1\,0\ldots & \delta\,(q_3,\,1) = (L) \\
\uparrow & \\
q_3 & \\
\ldots 0\,;\,1\,1\,;\,1\,1\,0\ldots & \delta\,(q_3,\,;) = (q_4\,L) \\
\uparrow & \\
q_4 & \\
\ldots 0\,;\,1\,1\,;\,1\,1\,0\ldots & \delta\,(q_4,\,1) = (q_0\,R) \\
\uparrow & \\
q_0 &
\end{array}$$

(*n* is completely copied to the right end)

$$\begin{array}{ll}
\ldots 0\,;\,1\,1\,,\,1\,1\,0\ldots & \delta\,(q_0,\,;) = (,\,q_5\,R) \\
\uparrow & \\
q_5 & \\
\ldots 0\,;\,1\,1\,,\,1\,1\,0\ldots & \delta\,(q_5,\,1) = (R) \\
\uparrow & \\
q_5 & \\
\ldots 0\,;\,1\,1\,,\,1\,1\,0\ldots & \delta\,(q_5,\,1) = (R) \\
\uparrow & \\
q_5 &
\end{array}$$

... 0 ; 1 1 , 1 1 ; 0 ... $\delta(q_5, 0) = (;\ q_6\ L)$
    ↑ q_6

... 0 ; 1 1 , 1 1 ; 0 ... $\delta(q_6, 1) = (L)$
    ↑ q_6

... 0 ; 1 1 , 1 1 ; 0 ... $\delta(q_6, 1) = (L)$
    ↑ q_6

... 0 ; 1 1 , 1 1 ; 0 ... $\delta(q_6, ,) = (\alpha\ N)$
    ↑ α

Thus, we see that the original sequence, '...0 ; 0 0 ; 0 ...' got replaced by '... 0 ; 1 1 , 1 1 ; 0 ...'. The number n, which was originally represented as '00' is now represented as '11', and is also duplicated in order to act as a multiplicand/multiplier pair for computing $n^2 = n \times n$. The new sequence now matches the initial configuration of the TM that performs multiplication. Hence, $\delta(q_6, ,) = (\alpha\ N)$ thus moves the TM to the initial state of the second (multiplication) TM.

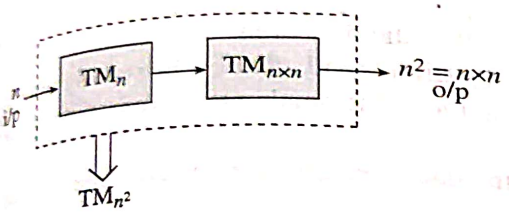


Figure 4.19 Computing n^2 using composite TM

Thus, the SFM in Table 4.16 is followed by the SFM for the TM that performs multiplication (refer to Table 4.12). This makes a composite TM, as required. The square of a given number is thus represented as a function composition (refer to Fig. 4.19).

In Fig, 4.19, TM$_n$ is the TM whose SFM is shown in Table 4.16, and TM$_{n\times n}$ is the TM which performs the multiplication. TM$_n{}^2$ is thus implemented as a composition of two TMs.

4.9 UNIVERSAL TURING MACHINE

Visualize a TM that simulates a given TM for a given input. So far, we have been simulating the working of a TM using a series of IDs. However, it is feasible to build a simulating TM if the input tape and the SFM of the TM to be simulated are available on the simulating TM's tape, along with the simulating algorithm to be implemented in the form of an SFM. Here, we are essentially building a program, which simulates another program for a given input.

Let us call the simulating program (or TM) as a meta-program (or meta-TM). The meta-TM takes the SFM for the TM to be simulated as one of the inputs (*program area* on the meta-TM's tape). We will also need the initial ID or initial configuration, along with the input string for the simulation (*data area* on the meta-TM tape).

We now implement the SFM for the meta-TM that represents the simulating algorithm (*system area*). The simulating algorithm is responsible for checking the current state and the current input symbol of the TM to be simulated by visiting the data area; it then, accordingly, picks the quintuple from the program area, and visits the data area again to simulate the action specified by the quintuple—it changes the state, if any, the symbol read, if any, and the direction as required. After one such simulation step, the data area of the meta-TM starts depicting the next ID of the TM being simulated.

A universal Turing machine (UTM) is capable of simulating any TM T, if the following information is available on its tape:

1. The description of T in terms of its SFM (*program area* of the tape)
2. The initial configuration of T with the processing data (input string) to be fed to T (*data area* of the tape)

This means that the UTM should have an *imitation algorithm* (simulating logic in the form of its SFM) in order to correctly interpret the rules of the operation given in the SFM of the TM being simulated, that is, T.

The UTM should also have a table look-up facility and should perform the following steps:

Imitation algorithm

1. Scan the tape cell on the data area of the tape and read the symbol from the same area that T reads from to start with; next, read the initial state of T.
2. Move the tape to the program area containing the SFM of T, and find the row in the SFM for the state symbol read in Step 1.
3. Find the column for the input symbol read in Step 1 and read the triplet (new symbol, new state, direction to move) stored as the entry, which is the entry at the intersection of the required row and column.
4. Move the tape to reach the appropriate tape cell in the data area.
5. Replace the symbol by the new symbol from the triplet read, change the state symbol to the next state symbol from the triplet read, and move the head in the required direction as specified in the triplet.
6. Read the next symbol to be read by T from the data area and the current state symbol. Then, go to Step 2.

Table 4.17 Example SFM

S \ I	0	1	$\flat$
α	$1\beta R$	$1\alpha R$	$\flat\alpha R$
β	***	***	$\flat\gamma N$

Since the UTM is a TM, the aforementioned imitation algorithm is implemented as an SFM.

We know that every TM, including the UTM, has a linear tape. Hence, we cannot arrange the SFM (which is a matrix or a table) for the TM to be simulated as it is, onto the UTM's tape. Hence, we must store it linearly either in row-major order or column-major order. For example, let us consider the SFM for T (sample TM) as shown in Table 4.17.

This SFM can be represented as a linear sequence (column-major ordering) as follows:

$$0\,\alpha\,1\,\beta\,R\,0\,\beta\,*\,*\,*\,1\,\alpha\,1\,\alpha\,R\,1\,\beta\,*\,*\,*\,\flat\,\alpha\,\flat\,\alpha\,R\,\flat\,\beta\,\flat\,\gamma\,N$$

The unspecified entry ('-') here is translated to '***', in order to keep the size of the quintuple intact, that is, to ensure that there are five symbols to represent each cell. This helps identifying the end of one quintuple and start of the next, in the linear storage.

The UTM tape consists of the program area, which is the SFM for the other TM stored either in row-major or column-major order, as we have just seen in the case of the example SFM in Table 4.17. Thus, we see that the UTM needs to consume not only the input symbols but also the state symbols, as well as the direction symbols as inputs on its tape. As

a result, the new tape alphabet for UTM will have an additional ten symbols, namely: $\lceil =$ $\{0, 1, b, \alpha, \beta, \gamma, L, R, N, *\}$. As we know, the symbols are user-defined, and different users assume different symbols while designing TMs. Now, if we expect the UTM to simulate all these TMs, it becomes practically impossible to impose finiteness onto the tape alphabet, $\lfloor$. Hence, there is a need to encode the symbols in some unique way.

For this, let us assume that there are altogether m distinct symbols. Hence, we can assign a unique and easy-to-decode binary code to each of these symbols with n or more bits, where $2^n \geq m$.

Let us consider the aforementioned example, in which we have ten different symbols, namely, $\{0, 1, b, \alpha, \beta, \gamma, L, R, N, *\}$, that is, $m = 10$.

We can represent or encode these symbols using a 4-bit binary code ($2^4 > 10 > 2^3$) as follows:

$0 = 0\,0\,0\,0$; $\quad 1 = 0\,0\,0\,1$; $\quad b = 0\,0\,1\,0$;
$\alpha = 0\,0\,1\,1$; $\quad \beta = 0\,1\,0\,0$; $\quad \gamma = 0\,1\,0\,1$;
$L = 0\,1\,1\,0$; $\quad R = 0\,1\,1\,1$; $\quad N = 1\,0\,0\,0$;
$* = 1\,0\,0\,1$

There are more encoding techniques as well. We shall discuss one such technique later in Chapter 9. Essentially, the binary encoding restricts the size of the tape alphabet for UTM to two letters—0 and 1.

Note: The concept of UTM laid the foundation for *stored-program computers* and interpretive implementation of *programming languages*. The SFM for UTM can be visualized as part of the *operating system*, which is the program (or finite set of programs) capable of loading and simulating the other programs.

Hence, we see that the UTM is a meta-program that takes other programs as input and simulates them. This is based the concept of *program as data*, which was later adopted in Lambda-calculus as well.

4.10 MULTI-TAPE TURING MACHINE

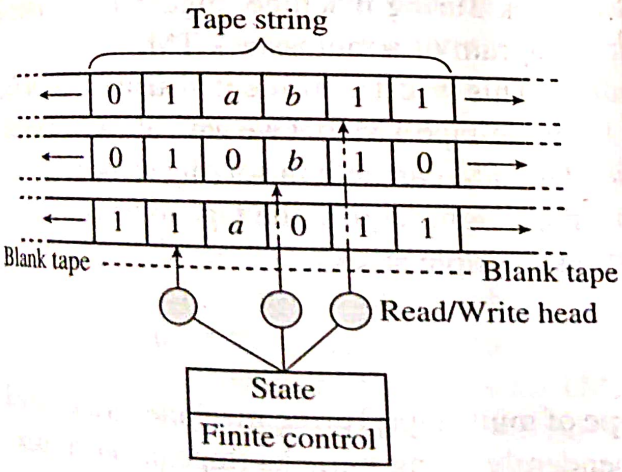


Figure 4.20 Multi-tape TM

Multi-tape Turing machines are similar to the single-tape Turing machines that we have discussed so far, but with some constant k number of independent tapes, having their own read/write heads. These machines have independent control over all the heads—any of these can move and read/write their own tapes. All these tapes are unbounded at both the ends just as in the single-tape TM. Figure 4.20 shows a multi-tape TM.

The multi-tape TM model intuitively seems to be much more powerful than the single-tape model. However, any multi-tape machine, no matter how large the k may be, can be simulated only by a single-tape machine using quadratically more computation time. Thus, multi-tape machines

cannot calculate any more functions than the single-tape machines do, and the computation complexity (the time taken to perform the computation) is also not affected dramatically by a change between single-tape and multi-tape machines. In short, the multi-tape TM and single-tape TM are equivalent in power (except for some difference in execution time). In turn, the multi-tape TM just adds to the convenience and not to the power of computation.

Formal Definition

A k-tape Turing machine is denoted by:

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

where,

Q: Finite set of states
Γ: Finite set of allowable tape symbols including blank character $\not b$
Σ: Subset of Γ, excluding blank character $\not b$; it is the set of input symbols
B: A symbol in Γ; it is the blank character $\not b$.
q_0: Start (or initial) state $\in Q$
F: Set of final states (or halt states) $\subseteq Q$
δ: Functional matrix, such that, $\delta: (Q \times \Gamma^k) \rightarrow (Q \times (\Gamma \times \{L, R, N\})^k)$

A k-tape TM can read up to k symbols at every instance, while in a current state. It then can change the state, erase the current symbol and write a new symbol for each of the k symbols, and can also decide the direction for each of the k heads, independent of each other.

4.11 MULTI-STACK TURING MACHINE

The symbols to the left of the head of the TM can be stored onto one stack, while the symbols on the right of the head can be placed on the other stack.

On each stack, symbols closer to the Turing machine's head are placed closer to the top of the stack. This type of organization is called multi-stack Turing machine or two-stack Turing machine. Figure 4.21 shows a diagram of a multi-stack TM.

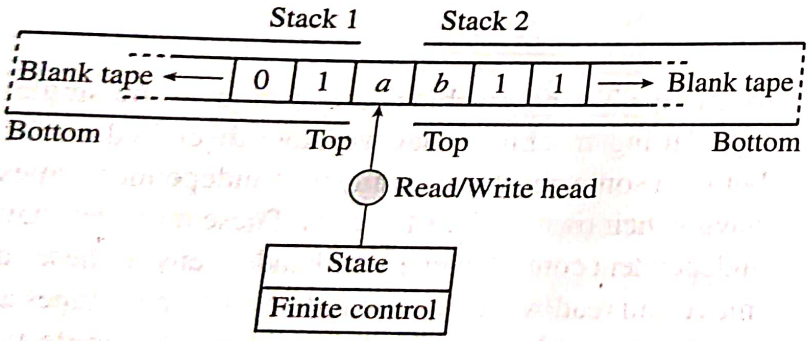


Figure 4.21 Multi-stack TM

This is just a different visualization of the single-tape TM that we have already studied. This TM can help in solving some problems that might require the tape to be considered as multiple stacks.

4.12 MULTI-TRACK TURING MACHINE

A multi-track Turing machine is a specific type of multi-tape Turing machine. In a standard k-tape Turing machine, k heads move independently along k tracks (tapes). On the other hand, in a k-track Turing machine, one head reads and writes on all tracks simultaneously.

A tape position in a k-track Turing machine denotes k symbols from the tape alphabet. This is equivalent to the standard single-tape Turing machine except that it reads/writes k symbols at one go, and therefore, accepts recursively enumerable languages.

Formal Definition

A k-track Turing machine is denoted by

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

where,

Q: Finite set of states
Γ: Finite set of allowable tape symbols including blank $\not b$
Σ: Subset of 'Γ' excluding the blank $\not b$; it is the set of input symbols
B: A symbol for blank character $\not b$
q_0: Start (or initial) state $\in Q$
F: Set of final states (or halt states) $\subseteq Q$
δ: Functional matrix, such that δ: $(Q \times \Gamma^k) \rightarrow (Q \times \Gamma^k \times \{L, R, N\})$

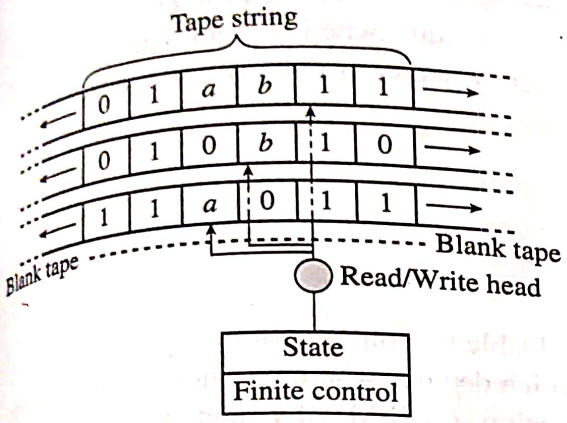


Figure 4.22 Multi-track TM

Note the difference in the definition of the functional matrix for multi-tape and multi-track TMs. Multi-track TMs can only read k symbols and print k new symbols after erasing the old symbols, but the direction change can only be done once. This is because they do not have k independent heads as the multi-tape TMs do; they only have k tracks and one head. Figure 4.22 helps us visualise a multi-track TM.

4.13 SOLVABLE, SEMI-SOLVABLE, AND UNSOLVABLE PROBLEMS

If there is a TM, which when applied to any problem, always eventually terminates with the correct 'yes' or 'no' answer, we call the problem *solvable*. For example, consider the problem of determining whether or not a given binary number is a palindrome (refer to Example 4.7). For any input binary number, the TM halts with the answer T (i.e., 'yes') or F (i.e., 'no'), based on whether it is a palindrome sequence or not, respectively. Most of the problems we have solved so far are solvable problems.

Now, if there is a TM, which when applied to any problem, always eventually terminates when the answer is 'yes', and may or may not terminate when the answer is 'no', we call the problem *semi-solvable* or *partially solvable*. For instance, let us consider Example 4.12. Any division problem is undefined if the divisor is zero. In such a case the TM we have designed might loop forever. This problem is thus partially solvable.

However, if there is no TM, which when applied to a problem, eventually terminates with the answer 'yes', we call the problem *unsolvable*. We are yet to see any such problem that is unsolvable, though there are many such problems existing in the world. One such problem is the *halting problem* that we are going to discuss in this chapter.

4.15 RECURSIVELY ENUMERABLE AND RECURSIVE LANGUAGES

The language that is accepted by a TM is called *recursively enumerable language* (
recursively enumerable set).

The term enumerable is derived from the fact that it is precisely these languages, whos
strings can be enumerated (listed) by a TM. For example, if $L(M)$ is such a language and
w is any string in $L(M)$, then M eventually halts on input w; but if the input string belongs
$\sim L(M)$—the complement of the set $L(M)$—then the TM M might fail to halt on this inpu
However, as long as M is still running on some input, we can never tell whether or no
M will eventually accept the input and halt, if we allow M to run long enough; or if
will run forever. Recursively enumerable languages belong to the semi-solvable class o
problems that we have discussed earlier.

It is convenient to single out a sub-class of recursively enumerable sets, called recursiv
languages (or recursive sets), which are the languages accepted by at least one TM tha
halts on all inputs. The TM M either reaches the 'accept halt' state if the input belongs to
the $L(M)$, or the 'reject halt' state if the input does not belong to $L(M)$. Recursive languages
belong to the solvable class of problems that we have discussed earlier.

Note that the class of recursive languages is a subset (specialization) of the more generic
class of recursively enumerable languages. Hence, in general, TMs are said to accept
recursively enumerable languages.

Formal Definitions

The following are the formal definitions for recursively enumerable and recursive sets.

Recursively enumerable set A set S of words over Σ is said to be recursively enu-
merable, if there is a TM over Σ, which accepts every word in S and either rejects or
loops for every word in $\sim S$ ($\sim S = \Sigma^* - S$). This can be represented as

$$\text{Accept TM} = S$$
$$\text{Reject (TM)} \cup \text{loop (TM)} = \Sigma^* - S$$

Recursive set A set S of words over Σ is said to be recursive, if there is a TM over Σ,
which accepts every word in S and rejects every word in $\sim S$ ($\sim S = \Sigma^* - S$). This can
be represented as

$$\text{Accept (TM)} = S$$
$$\text{Reject (TM)} = \Sigma^* - S$$
$$\text{Loop (TM)} = \phi$$

4.16 FUNCTIONS

A TM may be viewed as a computer of functions that accept arguments (input parameters)
in unary form and produce results (outputs) in unary form.

For example, let us assume that an integer $i \geq 0$ is represented by the string a^i. If a func-
tion has k integer arguments, that is, $i_1, i_2, \ldots, i_k$, then these integers are initially placed on
the tape separated by some delimiter, such as ','. For example, $a^{i1}, a^{i2}, \ldots, a^{ik}$. If the TM
halts (i.e., accepts or rejects) with a tape consisting of a^m for some m, then we say that

$$f(i_1, i_2, \ldots, i_k) = m,$$

where, f is the function of k arguments computed by this TM.

Note that the TMs can be designed to compute a function of one argument, a different function of two arguments, and so on. Furthermore, if the TM M computes a function f of k arguments, then f need not have a value for all different k-tuples of integers, $i_1, i_2, \ldots, i_k$. In other words, the function f may not be defined for all combinations of values for all the arguments. For example, division is not defined for all values of divisors—it is undefined for divisor $= 0$.

4.16.1 Total Recursive Functions

If $f(i_1, i_2, \ldots i_k)$ is defined for all values of arguments, $i_1, \ldots, i_k$, then f is said to be a total recursive function. These total recursive functions correspond to the recursive languages, since they are computed by a TM that always halts.

All common arithmetic functions on integers, such as multiplication, $[\log_2 n]$, and 2^{2n}, are total recursive functions. Most of the examples that we have solved so far are total recursive functions.

4.16.2 Partial Recursive Functions

If $f(i_1, i_2, \ldots, i_k)$ is not defined for all values of arguments $i_1, \ldots, i_k$, then f is said to be a partial recursive function. In other words, a function $f(i_1, \ldots, i_k)$ computed by a TM, which may or may not halt on a given input, is said to be a partial recursive function.

Partial recursive functions are analogous to recursively enumerable languages, since they are computed by a TM that halts on acceptance, while for any other input it may or may not halt. For example, factorial ($n!$) is only defined for integers, $n \geq 0$; it is undefined for negative integers. Another example that we have seen is that of division, which is undefined if divisor $= 0$.

Most of the real world problems are partial functions (or recursively enumerable). For example, while using an ATM, we cannot withdraw amount zero, or an amount that is more than the balance in the account (except in case of overdraft facility). Similarly, one cannot have negative weight; hence, the weighing machine must show positive values only.

4.17 CHURCH–TURING HYPOTHESIS

This hypothesis is also known as *Church–Turing conjecture*, *Church's Thesis*, or *Church's conjecture*. It essentially states that everything that is algorithmically computable is computable by a Turing machine. If some method (algorithm) exists to carry out a calculation, then the same calculation can also be carried out by a Turing machine.

American mathematician, Alonzo Church, created a method called λ-calculus (Lambda-calculus) for defining functions. It is an equivalent computational model of the UTM.

Statement of the hypothesis The intuitive notion of a computable function can be identified with the class of partial recursive functions. In other words, every problem having an algorithmic solution can be solved using a machine having the foregoing set of instructions.

4.18 POST'S CORRESPONDENCE PROBLEM

Let $A = w_1, w_2, \ldots, w_k$, and $B = x_1, x_2, \ldots, x_k$ be strings over some alphabet Σ.

Post's correspondence problem (PCP) is to find the correspondence sequence of integers $i_1, i_2, \ldots, i_m$, for $m \geq 1$ such that

$$w_{i1}, w_{i2}, \ldots, w_{im} = x_{i1}, x_{i2}, \ldots, x_{im}$$

The sequence, '$i_1, i_2, \ldots, i_m$' is considered to be the solution for the PCP instance. Each PCP instance is constituted by some set of values for A and B.

Note that the entire class of PCP instances is unsolvable. If we consider the PCP as a generic class of all such instances, then it is *unsolvable*. Furthermore, there exists no generic algorithm that can find a solution for any such PCP instance; hence, it is also an *undecidable* problem. However, for a few values of A and B, it might have a solution.

Let us now look at a few examples to understand this better.

Example 4.15 Let $\Sigma = \{1, 0\}$ and let A and B be defined as shown in Table 4.18. Find the correspondence sequence of integers, $i_1, i_2, \ldots, i_m$, for $m \geq 1$, such that

$$w_{i1}, w_{i2}, \ldots, w_{im} = x_{i1}, x_{i2}, \ldots, x_{im}$$

Table 4.18 Post's correspondence problem with a solution

	A	B
i	w_i	x_i
1	0	000
2	01000	01
3	01	1

Solution The given PCP instance has a solution for $m = 4$:

$$i_1 = 2$$
$$i_2 = 1$$
$$i_3 = 1$$
$$i_4 = 3$$

Observe that

$$w_2\, w_1\, w_1\, w_3 = (0\ 1\ 0\ 0\ 0)\ (0)\ (0)\ (0\ 1) = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1$$
$$x_2\, x_1\, x_1\, x_3 = (0\ 1)\ (0\ 0\ 0)\ (0\ 0\ 0)\ (1) = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1$$

Hence, $w_2\, w_1\, w_1\, w_3 = x_2\, x_1\, x_1\, x_3$

Example 4.16 Let $\Sigma = \{1, 0\}$ and let A and B be defined as shown in Table 4.19. Find the correspondence sequence of integers, $i_1, i_2, \ldots i_m$, for $m \geq 1$, such that

$$w_{i1}, w_{i2}, \ldots w_{im} = x_{i1}, x_{i2}, \ldots, x_{im}$$

Table 4.19 Post's Correspondence problem without a solution

	A	B
i	w_i	x_i
1	*ba*	*bab*
2	*abb*	*bb*
3	*bab*	*Abb*

Solution We see that this PCP instance is unsolvable, since we cannot find any correspondence sequence of integers as required.

Thus, every instance of the given PCP might not have a solution. In other words, the given PCP is undecidable; it means that the given PCP does not have any algorithmic solution.

There are a wide variety of other problems, which are undecidable.

As there is no algorithm that can handle an entire class of PCP instances, PCP is considered an unsolvable problem. There is no TM (no algorithmic solution) that can be built to solve a class of PCP instances. This also indicates that mathematics

cannot always be reduced to the construction of algorithms, as even in narrow areas of mathematics, such as group theory, there are many algorithmically unsolvable problems.

4.10 ADDITIONAL TURING MACHINE EXAMPLES

Let us now look at some more examples of TMs.

Example 4.17 Design a TM that increments the value of any binary number by one. The output should also be a binary number, whose value is one more than the given number.

Solution Recall that we have designed a Mealy machine for solving the same problem in Chapter 2 (refer to Section 2.10.2, Example 2.16).

The following was the algorithm for the machine:

1. Read bit by bit from least significant bit (LSB) to most significant bit (MSB), that is, from right to left.
2. Keep on replacing the 1's by 0's till you reach the first 0.
3. Replace this first 0 by 1.
4. Keep the remaining bits as they are.

For example, let us consider the binary number '1001'. According to the aforementioned method, the following changes are made stepwise:

1 0 0 1 → replace by 0

1 0 0 0 → first 0, replace by 1

1 0 1 0 → remaining bits as they are without any change

Thus, '1010' is the output obtained after incrementing value of '1001' by one. We shall apply the same algorithm to build the SFM for the TM we want to design.

The initial configuration of the TM is considered as shown in Fig. 4.24(a). Using the aforementioned algorithm, we form the SFM as shown in Table 4.20. The transition graph for the same can be drawn as shown in Fig. 4.24(b).

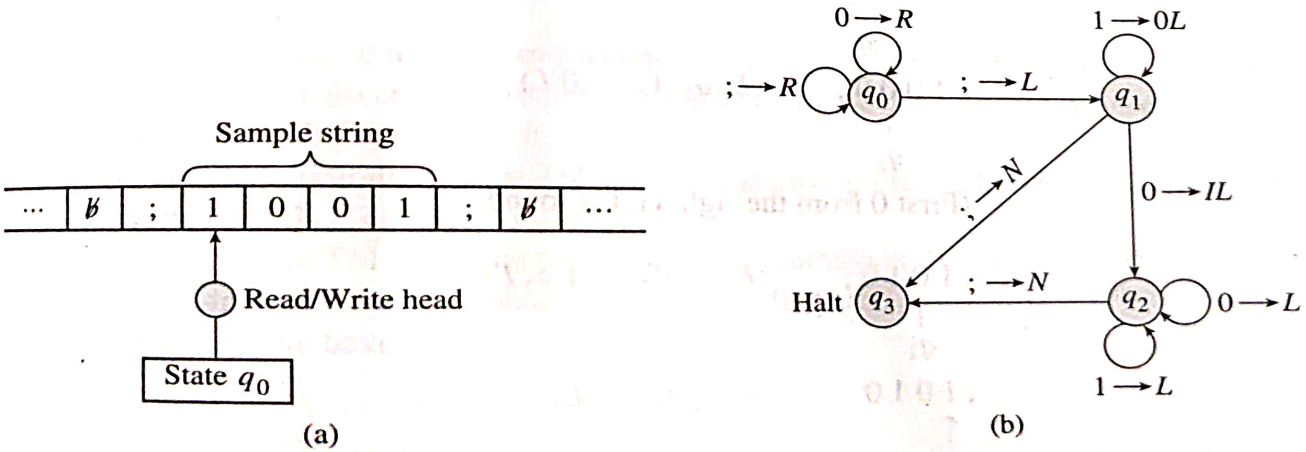


Figure 4.24 TM that adds one to any binary number (a) Initial configuration (b) Transition graph

Table 4.20 SFM for TM that adds one to any binary number

S \ I	0	1	;
q_0	R	R	q_1L
q_1	$1q_2L$	$0L$	q_3N
q_2	L	L	q_3N
q_3	—	—	—

For this TM, we have

$I = \{0, 1, ;\}$
$S = \{q_0, q_1, q_2, q_3 = \text{halt}\}$
$D = \{L, R, N\}$

State q_0 is responsible for finding the right end of the string so that the TM can start reading from the LSB. State q_1, while searching for the first 0 from the right end, replaces all the 1's by 0's. It then replaces the first 0 also by 1 and transits to state q_2. State q_2 ignores the remaining 0's and 1's—that is, it keeps all remaining bits as they are.

Now, let us simulate the working of the TM on some input binary strings.

Simulation

1. Let us simulate the working of the aforementioned TM for binary number '1001'.

```
;1001;        initial configuration
 ↑
q0
;1001;        δ(q0, 1) = (R)
 ↑
q0
;1001;        δ(q0, 0) = (R)
 ↑
q0
;1001;        δ(q0, 0) = (R)
 ↑
q0
;1001;        δ(q0, 1) = (R)
 ↑
q0
;1001;        δ(q0, ;) = (q1 L)
 ↑
q1
;1000;        δ(q1, 1) = (0 L)
 ↑
q1
```

(First 0 from the right end is found)

```
;1010;        δ(q1, 0) = (1 q2 L)
 ↑
q2
;1010;        δ(q2, 0) = (L)
↑
q2
```

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

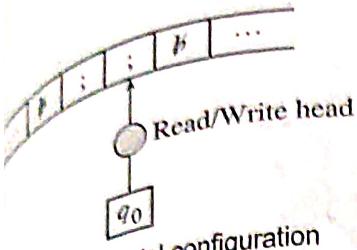


Figure 4.25 Initial configuration when input is ϵ

2. Let us check the working of the aforementioned TM for a boundary condition, that is, when an empty string is fed as the input.

When the input is an empty string, then the initial configuration will be as shown in Fig. 4.25. Initially, the head points to the right end-marker, ';', to indicate that the input string is empty.

The simulation will be as follows:

;; Initial configuration
↑
q_0
;; $\delta(q_0, ;) = (q_1\ L)$
↑
q_0
;; $\delta(q_1, ;) = (q_3\ N)$
↑
q_0

Example 4.18 Design a TM that computes the function $f(x, y)$, which is defined as follows:

$$f(x, y) = x - y, \text{ if } x > y$$
$$= 0, \text{ if } x \leq y$$

Solution Let us consider x and y as integers represented in unary format using symbol a. For computing the aforementioned function, we must compare x and y, and then subtract y from x, if $x > y$.

We have already designed a TM that compares two numbers in Example 4.8 in Section 4.6.

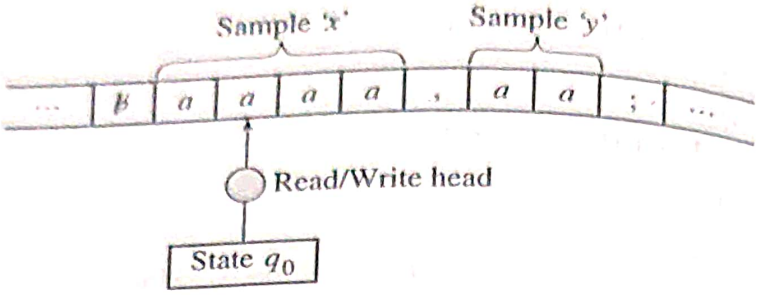
We are now going to use the same technique, but with a little modification to sets I and S, because here, we do not have to produce the result of the comparison; we need to show the result of the subtraction only if $x > y$.

The initial configuration of the TM will be as shown in Fig. 4.26(a) and the TG for the same is shown in Fig. 4.26(b).

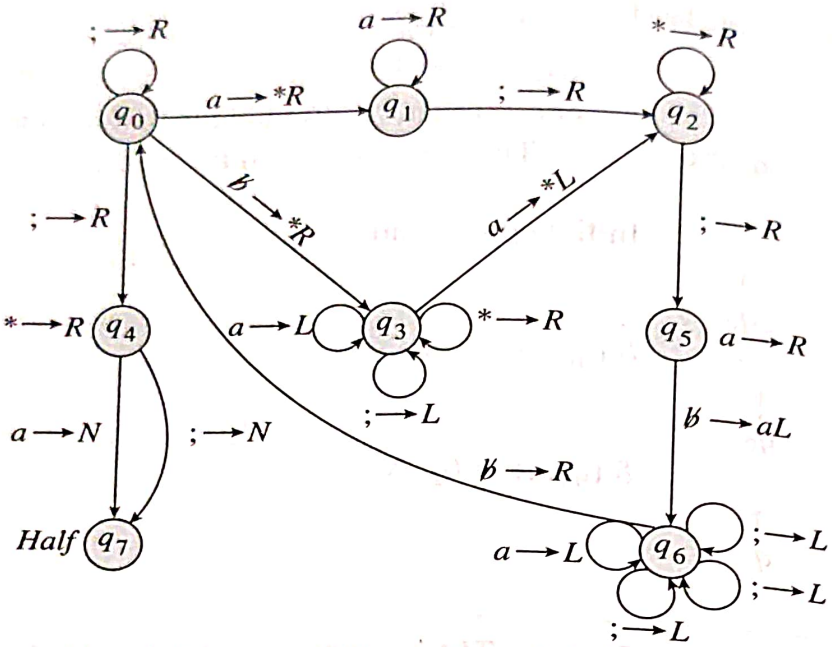
If $x > y$, then the TM will write the result of the subtraction, '$x - y$', after the semicolon (;). Else the machine will not write anything onto the tape.

For this TM, we have

$$I = \{a, *, ', ;, \not b\}$$
$$S = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7 = \text{halt}\}$$
$$D = \{L, R, N\}$$



(a)



(b)

Figure 4.26 TM that computes $f(x, y)$ (a) Initial configuration (b) Transition graph

or the TM that computes $f(x, y)$ is given in Table 4.21.

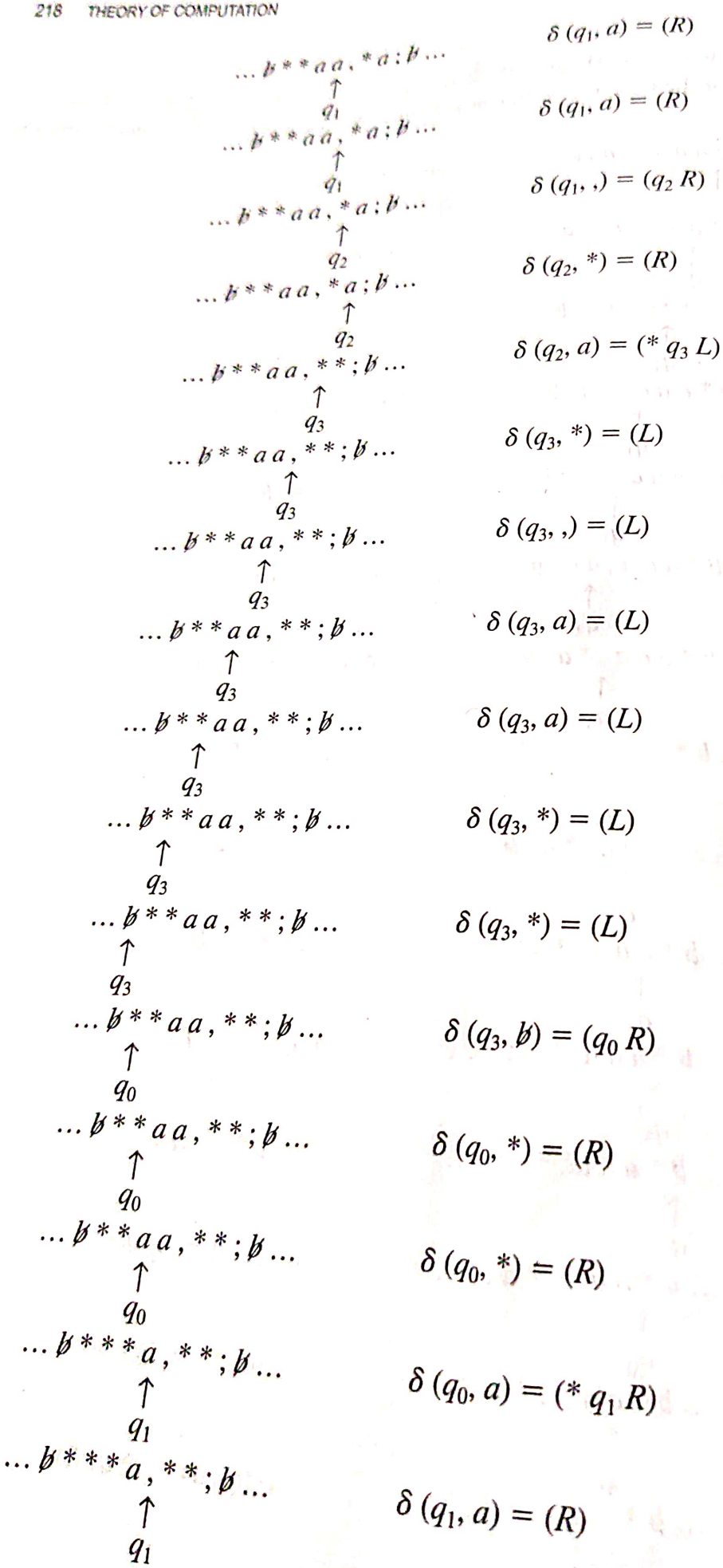
Table 4.21 SFM for a TM that computes $f(x, y)$

	a	,	*	;	$\not b$
q_0	$*q_1R$	q_4R	R	—	—
q_1	R	q_2R	—	—	—
q_2	$*q_3L$	—	R	q_5R	—
q_3	L	L	L	—	q_0R
q_4	q_7N	—	R	q_7N	—
q_5	R	—	—	—	$a\ q_6L$
q_6	L	L	L	L	q_0R

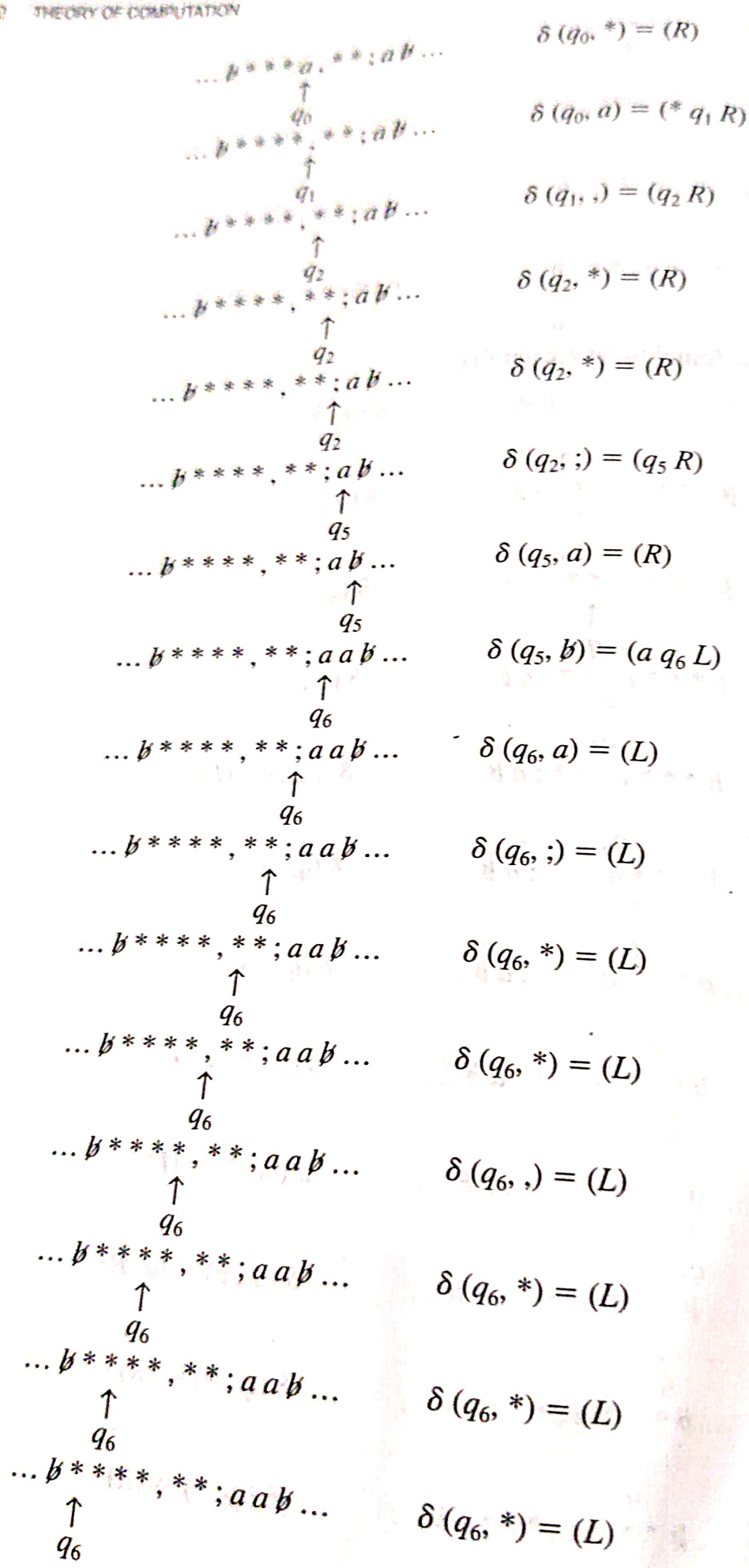
Simulation

1. Let us simulate the working of the TM that we have constructed for $x = 4$ and $y = 2$, that is, for the case $x > y$. In unary from, x will be represented as '*aaaa*' and y is represented as '*aa*'.

$\ldots \not b\ a\ a\ a\ a\ ,\ a\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_0	initial configuration
$\ldots \not b\ *\ a\ a\ a\ ,\ a\ a\ ;\ \not b \ldots$ $\uparrow$ q_1	$\delta\ (q_0, a) = (*\ q_1\ R)$
$\ldots \not b\ *\ a\ a\ a\ ,\ a\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_1	$\delta\ (q_1, a) = (R)$
$\ldots \not b\ *\ a\ a\ a\ ,\ a\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_1	$\delta\ (q_1, a) = (R)$
$\ldots \not b\ *\ a\ a\ a\ ,\ a\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_1	$\delta\ (q_1, a) = (R)$
$\ldots \not b\ *\ a\ a\ a\ ,\ a\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_2	$\delta\ (q_1, ,) = (q_2\ R)$
$\ldots \not b\ *\ a\ a\ a\ ,\ *\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_3	$\delta\ (q_2, a) = (*\ q_3\ L)$
$\ldots \not b\ *\ a\ a\ a\ ,\ *\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_3	$\delta\ (q_3, ,) = (L)$
$\ldots \not b\ *\ a\ a\ a\ ,\ *\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_3	$\delta\ (q_3, a) = (L)$
$\ldots \not b\ *\ a\ a\ a\ ,\ *\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_3	$\delta\ (q_3, a) = (L)$
$\ldots \not b\ *\ a\ a\ a\ ,\ *\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_3	$\delta\ (q_3, a) = (L)$
$\ldots \not b\ *\ a\ a\ a\ ,\ *\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_3	$\delta\ (q_3, *) = (L)$
$\ldots \not b\ *\ a\ a\ a\ ,\ *\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_0	$\delta\ (q_3, \not b) = (q_0\ R)$
$\ldots \not b\ *\ a\ a\ a\ ,\ *\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_0	$\delta\ (q_0, *) = (R)$
$\ldots \not b\ *\ *\ a\ a\ ,\ *\ a\ ;\ \not b\ \ldots$ $\uparrow$ q_1	$\delta\ (q_0, a) = (*\ q_1\ R)$


δ (q1, a) = (R)
... b * * a a , * a ; b ...
q1
δ (q1, a) = (R)
... b * * a a , * a ; b ...
q1
δ (q1, ,) = (q2 R)
... b * * a a , * a ; b ...
q2
δ (q2, *) = (R)
... b * * a a , * a ; b ...
q2
δ (q2, a) = (* q3 L)
... b * * a a , * * ; b ...
q3
δ (q3, *) = (L)
... b * * a a , * * ; b ...
q3
δ (q3, ,) = (L)
... b * * a a , * * ; b ...
q3
δ (q3, a) = (L)
... b * * a a , * * ; b ...
q3
δ (q3, a) = (L)
... b * * a a , * * ; b ...
q3
δ (q3, *) = (L)
... b * * a a , * * ; b ...
q3
δ (q3, *) = (L)
... b * * a a , * * ; b ...
q3
δ (q3, b) = (q0 R)
... b * * a a , * * ; b ...
q0
δ (q0, *) = (R)
... b * * a a , * * ; b ...
q0
δ (q0, *) = (R)
... b * * * a , * * ; b ...
q1
δ (q0, a) = (* q1 R)
... b * * * a , * * ; b ...
q1
δ (q1, a) = (R)

$\ldots \not b * * * a , * * ; \not b \ldots$ $\uparrow$ q_2	$\delta(q_1, ,) = (q_2\ R)$
$\ldots \not b * * * a , * * ; \not b \ldots$ $\uparrow$ q_2	$\delta(q_2, *) = (R)$
$\ldots \not b * * * a , * * ; \not b \ldots$ $\uparrow$ q_2	$\delta(q_2, *) = (R)$
$\ldots \not b * * * a , * * ; \not b \ldots$ $\uparrow$ q_5	$\delta(q_2, ;) = (q_5\ R)$
(x is found to be greater than y)	
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_6	$\delta(q_5, \not b) = (a\ q_6\ L)$
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_6	$\delta(q_6, ;) = (L)$
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_6	$\delta(q_6, *) = (L)$
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_6	$\delta(q_6, *) = (L)$
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_6	$\delta(q_6, ,) = (L)$
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_6	$\delta(q_6, a) = (L)$
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_6	$\delta(q_6, *) = (L)$
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_6	$\delta(q_6, *) = (L)$
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_6	$\delta(q_6, *) = (L)$
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_6	$\delta(q_6, \not b) = (q_0\ R)$
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_0	$\delta(q_0, *) = (R)$
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_0	$\delta(q_0, *) = (R)$
$\ldots \not b * * * a , * * ; a \not b \ldots$ $\uparrow$ q_0	$\delta(q_0, *) = (R)$


δ (q0, *) = (R)
δ (q0, a) = (* q1 R)
δ (q1, ,) = (q2 R)
δ (q2, *) = (R)
δ (q2, *) = (R)
δ (q2, ;) = (q5 R)
δ (q5, a) = (R)
δ (q5, b) = (a q6 L)
δ (q6, a) = (L)
δ (q6, ;) = (L)
δ (q6, *) = (L)
δ (q6, *) = (L)
δ (q6, ,) = (L)
δ (q6, *) = (L)
δ (q6, *) = (L)
δ (q6, *) = (L)

to the recursive languages, since they are computed by a TM that always halts.

If $f(i_1, i_2, \ldots, i_k)$ is not defined for all values of arguments $i_1, \ldots, i_k$, then f is said to be a *partial recursive function*. In other words, a function $f(i_1, \ldots, i_k)$ computed by a TM, which may or may not halt on a given input, is said to be a partial recursive function. Partial recursive functions are analogous to the recursively enumerable languages, since they are computed by a TM that halts on acceptance and may or may not halt for any other input.

A *linear bounded automaton* (LBA) is a restricted form of a TM. While a TM has a tape, which is considered unbounded at both the ends, the LBA only has a finite contiguous portion of tape, whose length is a linear function of the length of the initial input. However, it is computationally equivalent to a Turing machine. The LBA is a somewhat more realistic model of computers than a Turing machine.

EXERCISES

This section lists a few unsolved problems to help the readers understand the topic better and practice a few examples related to TMs.

Objective Questions

(L) 4.1 An arbitrary TM M is given, and a language L is defined as follows
$L = (0 + 00)^*$ if M accepts at least one string.
$L = (0 + 00 + 000)^*$ if M accepts at least two strings.
$L = (0 + 00 + 000 + 0000)^*$ if M accepts at least three strings
Similarly,
$L = (0 + 00 + 000 + \ldots + 0\text{^}n)^*$ if M accepts at least $n - 1$ strings.
Choose the correct statement.
(a) We cannot say anything about L, as the question of whether or not a TM accepts a string is undecidable.
(b) L is context-sensitive but not regular.
(c) L is context-free but not regular.
(d) L is not a finite set.

(L) 4.2 If the strings of a language L that is accepted by a TM can be effectively enumerated in lexicographic (i.e., alphabetic) order, which of the following statements is true?
(a) L is necessarily finite.
(b) L is regular but not necessarily finite.
(c) L is context-free but not necessarily regular.
(d) L is recursive but not necessarily context-free.

(L) 4.3 If the strings of a language L that are accepted by a multi-track TM M can be effectively enumerated in lexicographic (i.e., alphabetic) order, which of the following statements is true?
(a) L is necessarily finite.
(b) L is regular but not necessarily finite.
(c) L is context-free but not necessarily regular.
(d) L is recursive but not necessarily context-free.

(R) 4.4 The language accepted by a TM is called _____ language.

(U) 4.5 State whether the following statements are true or false.
(a) FSM and TM are equivalent machines.
(b) For every FSM there exists an equivalent TM.
(c) For every TM there exists an equivalent FSM.
(d) FSM is more powerful than TM.

(R) 4.6 SFM means _____.

(L) 4.7 A single-tape TM M has two states, q_0 and q_1, of which q_0 is the starting state. The tape alphabet of M is $\{0, 1, \not b\}$ and its input alphabet is $\{0, 1\}$. The symbol $\not b$ is used to indicate the end of the input string. The transition function of M is described in Table 4.24.

Table 4.24 Transition function of M

	0	1	b
q_0	$q_1\,1\,R$	$q_1\,1\,R$	Halt
q_1	$q_1\,1\,R$	$q_0\,1\,L$	$q_0\,b\,L$

Which of the following statements is false about M?
(a) M halts on any string in $(0 + 1)^+$
(b) M halts on any string in $(00 + 1)^*$
(c) M does not halt on all strings ending in 0
(d) M does not halt on all strings ending in 1

U 4.8 Consider a TM M whose SFM is as shown in Table 4.25, where Q_0 is the initial state and Q_f is a final (halt) state.

Table 4.25 Simplified functional matrix

	0	1	B
Q_0	$Q_0\,0\,R$	$Q_2\,1\,L$	$Q_f\,N$
Q_1	$Q_2\,1\,L$	$Q_1\,1\,R$	$Q_f\,N$
Q_2	$Q_2\,1\,L$	$Q_2\,0\,L$	$Q_f\,N$
Q_f	—	—	—

Choose the correct statement:
(a) The machine accepts all strings over $\{0, 1\}$ ending with 1.
(b) The machine accepts 0^*.
(c) The machine accepts 01.
(d) The machine accepts 001.

U 4.9 Which of the following languages are accepted by a TM?
(i) $L = \{a^n b^n \mid n = 0, 1, 2, \ldots\}$
(ii) The set of palindromes over alphabet $\{a, b\}$
(iii) $L = \{a^n, b^m c^{2m} \mid n, m \geq 0\}$
(a) Only (i)
(b) Only (ii)
(c) (i) and (iii)
(d) All of these
(e) None of these

U 4.10 With respect to the power of recognition of languages, which of the following statements is false?
(a) Linear bound automata are equivalent to Turing machines.
(b) Linear bound automata are equivalent to multi-track TMs.
(c) Linear bound automata are equivalent to deterministic finite automata.
(d) Linear bound automata are equivalent to multi-tape TMs.

U 4.11 The C language is
(a) a context-free language
(b) a context-sensitive language
(c) a regular language.
(d) completely parsable only by TMs.

U 4.12 Choose the correct statement:
(a) There exists a universal TM, which can simulate any TM M on its input w.
(b) There does not exist a universal Turing machine, which can simulate any TM M on its input w.
(c) The universal language is recursive.

U 4.13 State whether the following statement is true of false:
Total recursive function is a special case of partial recursive function.

Review Questions

L 4.1 Compare FSM and TM.
U 4.2 Explain the halting problem.
C 4.3 Design a TM for multiplying two unary numbers. Show the stepwise functioning of the TM for the input sequences:
(a) 111×1111 (b) 111×11
C 4.4 Design a TM to recognize an arbitrary string divisible by 4 from $\Sigma = \{0, 1, 2\}$.
C 4.5 Design a TM to compute $n!$ (factorial n). Show the stepwise functioning of the TM for the input $n = 3$.
C 4.6 Design a TM to convert a binary-coded decimal number into a unary number. Validate the design for:
(a) 1001 (b) 0000
U 4.7 What is a universal Turing machine?
C 4.8 Design a TM to find the GCD of two given numbers.
U 4.9 Write a short note on the halting problem.
C 4.10 Design a TM to compare two numbers, which will produce the output L if the first number is lesser than the second number, output G if the first number is greater than the second number, and output E otherwise.

(C) 4.11 Design a TM, which computes the 2's complement of a given binary number.

(C) 4.12 Construct a TM for the language, $L = \{a^m b^n \mid m \geq n, n \geq 1\}$.

(C) 4.13 Construct a TM for checking if a given set of parentheses are well-formed.

(U) 4.14 Write a short note on solvability and semi-solvability.

(U) 4.15 Write a short note on recursive TM.

(L) 4.16 Consider the TM $M = \{(q_0, q_1, q_2, q_f), (0, 1), (0, 1, B), \delta, q_0, B, (q_f)\}$.
Describe the language $L(M)$, if δ consists of the following sets of rules:
$\delta(q_0, 0) = (q_1\ 1\ R)$;
$\delta(q_1, 1) = (q_0\ 0\ R)$;
$\delta(q_1, B) = (q_f\ B\ R)$.

(C) 4.17 Design a TM that accepts the language $\{0^n\ 1^n\ 0^n \mid n \geq 1\}$. In addition, give the transition function and the transition diagram.

(E) 4.18 Construct a TM that accepts the language $(a^*\ b\ a^*\ b)$.

(C) 4.19 Design TMs that recognize the following languages:
(a) $\{0^n\ 1^n\ 0^n \mid n \geq 1\}$
(b) $\{WW^R \mid W \text{ is in } (0 + 1)^*\}$
(c) The set of strings with equal number of 0's and 1's

(U) 4.20 Define the TM, explain its working, and give the applications of the same.

(C) 4.21 Construct TMs that recognize the following languages:
(a) $L = \{0^n\ 1^m \mid n, m >= 0\}$
(b) $L = \{x \in \{0, 1\}^* \mid x \text{ ends in } 00\}$

(U) 4.22 Define and explain a multi-tape TM.

(U) 4.23 Define recursive and recursively enumerable languages.

(U) 4.24 Explain the following for a TM:
(a) Power of TM over finite state machine
(b) Universal TM

(E) 4.25 Design a TM to replace '110' by '101' i
binary input string.

(U) 4.26 Write a short note on Post's correspond
problem (PCP).

(C) 4.27 Draw transitions tables for TMs that acc
the following languages:
(a) $\{a^i b^j \mid i < j\}$
(b) The language of balanced string
parentheses

(C) 4.28 Draw a transition table for a TM that accep
the language of all non-palindromes ov
$\{a, b\}$.

(R) 4.29 Define the following:
(a) Multi-track TM
(b) Multi-tape TM
(c) Recursively enumerable language
(d) Recursive language

(C) 4.30 Design a TM to compute the function n^2.

(C) 4.31 Design a TM to accept the language, $L = \{x$
$\in \{0, 1\}^* \mid x$ contains equal number of 0's
and 1's}. Simulate the operation for the string
'110100'.

(C) 4.32 Design a TM to find the value of $\log_2 n$,
where n is any binary number and a perfect
power of 2.

(E) 4.33 Determine the solution for the following
instance of Post's correspondence problem
in Table 4.26.

Table 4.26 Post's correspondence problem

	A	B
i	w_i	x_i
1	01	0
2	110010	0
3	1	1111
4	11	01

Answers to Objective Questions

4.1. (a) 4.2. (d) 4.3. (d) 4.4. recursively enumerable 4.5. (a) False (b) True (c) False (d) True 4.6. simplified functional matrix 4.7. (a) 4.8. (b) 4.9. (d) 4.10. (c) 4.11. (a) 4.12. (a) 4.13. True