4 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, $\not b$'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 $\not b$.
4. A finite set of states denoted by S; the machine resides in one of these states.

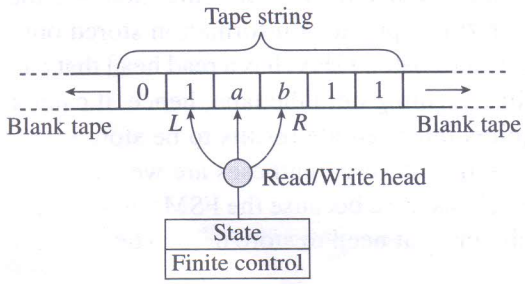


Figure 4.1 Turing machine

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:

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

Table 4.2 Simplified functional matrix

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

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.

simplified functional matrix

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	—	—	—

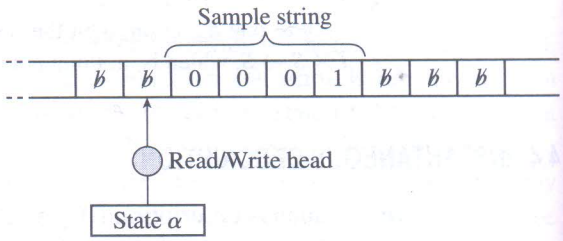


Figure 4.2 Initial configuration of example TM

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':

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

4.5 TRANSI

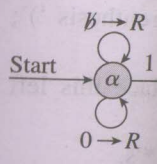


Figure 4.3

$$\vdash_M \quad \not b\,0\,0\,0\,\alpha\,1\,\not b \qquad \delta(\alpha, 0) = (R)$$

$$\vdash_M \quad \not b\,0\,0\,0\,0\,\beta\,\not b \qquad \delta(\alpha, 1) = (0\,\beta\,R)$$

$$\vdash_M \quad \not b\,0\,0\,0\,0\,\gamma\,\not b \qquad \delta(\beta, \not b) = (\gamma\,N)$$

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:

$\not b\,0\,0\,0\,1\,\not b$ $\uparrow$ α	initial configuration
$\not b\,0\,0\,0\,1\,\not b$ $\uparrow$ α	moved one position to right using $\delta(\alpha, \not b)$
$\not b\,0\,0\,0\,1\,\not b$ $\uparrow$ α	moved one position to right using $\delta(\alpha, 0)$
$\not b\,0\,0\,0\,1\,\not b$ $\uparrow$ α	moved right position to right using $\delta(\alpha, 0)$
$\not b\,0\,0\,0\,1\,\not b$ $\uparrow$ α	moved one position to right using $\delta(\alpha, 0)$
$\not b\,0\,0\,0\,0\,\not b$ $\uparrow$ β	changed 1 to 0, changed state to β, and moved one position to right using $\delta(\alpha, 1)$
$\not b\,0\,0\,0\,0\,\not b$ $\uparrow$ γ (halt)	changed state to γ with no movement using $\delta(\beta, \not b)$, halt after that

4.5 TRANSITION GRAPH FOR TURING MACHINE

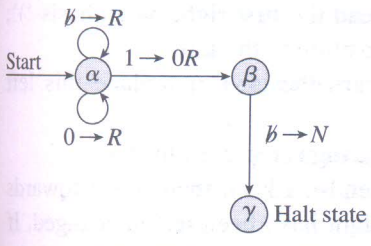


Figure 4.3 TG for example TM

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.

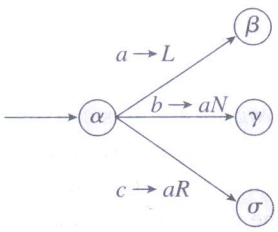


Figure 4.4 Example transition graph

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 $(\not b, \beta)$ is $(\not b, \beta, R)$, that is, $\delta(\beta, \not b) = (\not b, \beta, R)$, 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).

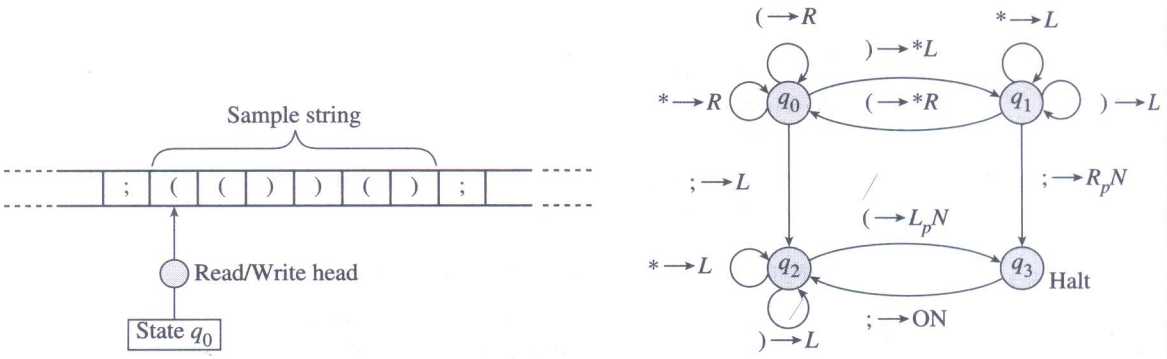
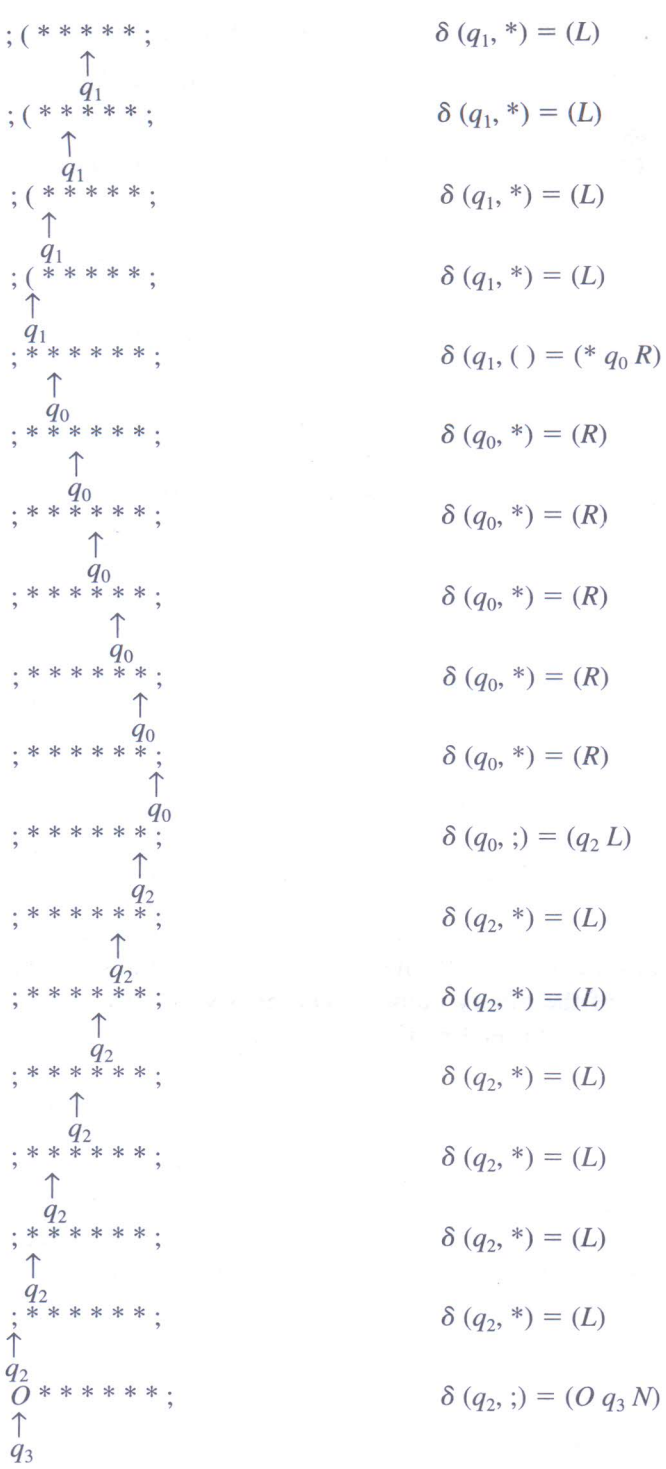


Figure 4.5 TM for checking if a set of parentheses are well-formed (a) Initial configuration (b) TG for TM that checks if parentheses are well-formed

Simulation

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

; (( ) ( )) ; ↑ q_0	initial configuration
; (( ) ( )) ; ↑ q_0	$\delta\ (q_0, (\ ) = (R)$
; (( ) ( )) ; ↑ q_0	$\delta\ (q_0, (\ ) = (R)$
; ((* ( )) ; ↑ q_1	$\delta\ (q_0,)\ ) = (*\ q_1, L)$
; (* * ( )) ; ↑ q_0	$\delta\ (q_1, (\ ) = (*\ q_0\ R)$
; (* * ( )) ; ↑ q_0	$\delta\ (q_0, *) = (R)$
; (* * ( )) ; ↑ q_0	$\delta\ (q_0, (\ ) = (R)$
; (* * (*)) ; ↑ q_1	$\delta\ (q_0,)\ ) = (*\ q_1\ L)$
; (* * * *) ; ↑ q_0	$\delta\ (q_1, (\ ) = (*\ q_0\ R)$
; (* * * *) ; ↑ q_0	$\delta\ (q_0\ *) = (R)$
; (* * * * * ; ↑ q_1	$\delta\ (q_0,)\ ) = (*\ q_1\ L)$



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 '('.

; (; ↑ q_0	initial configuration
; (; ↑ q_0	$\delta\ (q_0, (\ ) = (R)$
; (; ↑ q_2	$\delta\ (q_0, ;) = (q_2\ L)$
; L_p ; ↑ q_3	$\delta\ (q_2, (\ ) = (L_p\ q_3\ N)$

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:

; ; ↑ q_0	initial configuration
; ; ↑ q_2	$\delta\ (q_0, ;) = (q_2\ L)$
O ; ↑ q_3	$\delta\ (q_2, ;) = (O\ q_3\ N)$

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).

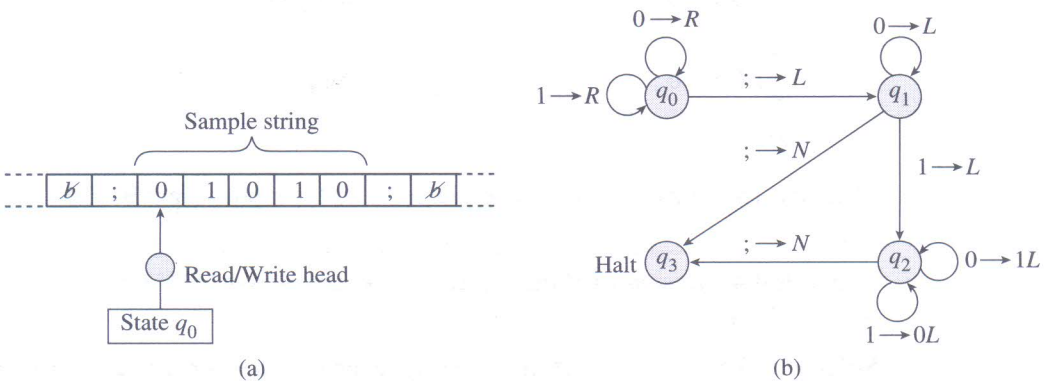


Figure 4.6 TM to find the 2's complement of a given binary number (a) Initial configuration (b) TG for TM that finds 2's complement of a binary number

Simulation

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

```
;0 1 0 1 0;          initial configuration
 ↑
 q0
;0 1 0 1 0;          δ (q0, 0) = (R)
  ↑
  q0
;0 1 0 1 0;          δ (q0, 1) = (R)
   ↑
   q0
;0 1 0 1 0;          δ (q0, 0) = (R)
    ↑
    q0
;0 1 0 1 0;          δ (q0, 1) = (R)
     ↑
     q0
;0 1 0 1 0;          δ (q0, 0) = (R)
      ↑
      q0
```

;0 1 0 1 0 ; ↑ q_1	$\delta(q_0, ;) = (q_1\ L)$
;0 1 0 1 0 ; ↑ q_1	$\delta(q_1, 0) = (L)$
;0 1 0 1 0 ; ↑ q_2	$\delta(q_1, 1) = (q_2\ L)$
;0 1 1 1 0 ; ↑ q_2	$\delta(q_2, 0) = (1\ L)$
;0 0 1 1 0 ; ↑ q_2	$\delta(q_2, 1) = (0\ L)$
;1 0 1 1 0 ; ↑ q_2	$\delta(q_2, 0) = (1\ L)$
;1 0 1 1 0 ; ↑ q_3	$\delta(q_2, ;) = (q_3\ N)$

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 sequence of 0's and 1's.

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

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

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	—	—	—

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

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 identify the sequence '111'. Once the sequence is identified, state q_2 makes a transition to state q_3 and points to the middle 1, which is replaced by 0 in state q_3; thus, the string is replaced by '101'. Once the replacement is done, the TM moves to state q_4, which resets to state q_0 for repeating the same process again to search for another sequence of '111' that may exist.

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

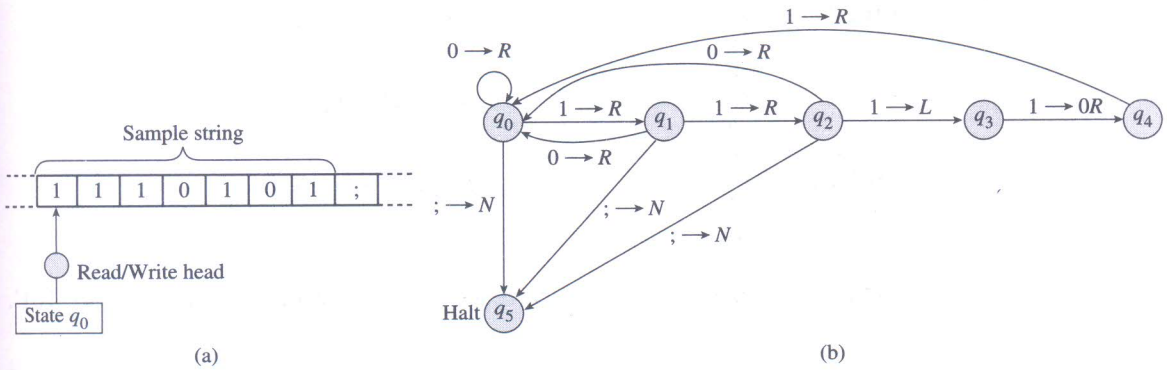


Figure 4.7 TM that replaces all occurrences of '111' by '101' (a) Initial configuration (b) TG for TM that replaces each occurrence of '111' by '101'

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

$$
\begin{array}{ll}
1\,0\,1\,1\,0\,1\,0\,1\,; & \\
\quad\uparrow & \\
\quad q_4 & \\
1\,0\,1\,1\,0\,1\,0\,1\,; & \delta\,(q_3, 1) = (0\,q_4\,R) \\
\quad\quad\uparrow & \\
\quad\quad q_0 & \\
1\,0\,1\,1\,0\,1\,0\,1\,; & \delta\,(q_4, 1) = (q_0\,R) \\
\quad\quad\quad\uparrow & \\
\quad\quad\quad q_5 & \delta\,(q_0, ;) = (q_5\,N)
\end{array}
$$

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).

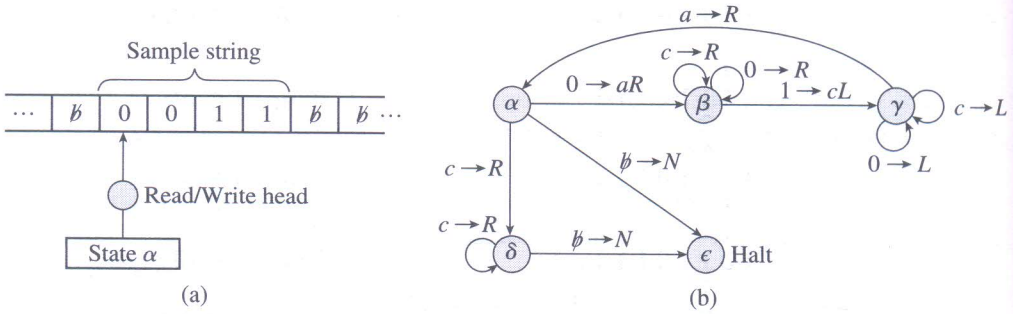


Figure 4.8 TM that recognizes strings of the form $0^n\,1^n$ for $n \geq 0$ (a) Initial configuration (b) Transition graph

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

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

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	cN (accept)
β	R	$c\gamma L$	—	R	—
γ	L	—	αR	L	—
δ	—	—	—	R	cN (accept)
c	—	—	—	—	—

Here, α is the initial state of the machine and c 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'.

$0\ 0\ 1\ 1\ \flat$ $\uparrow$ α	initial configuration
$a\ 0\ 1\ 1\ \flat$ $\uparrow$ β	$\delta\ (\alpha, 0) = (a\ \beta\ R)$
$a\ 0\ 1\ 1\ \flat$ $\uparrow$ β	$\delta\ (\beta, 0) = (R)$
$a\ 0\ c\ 1\ \flat$ $\uparrow$ γ	$\delta\ (\beta, 1) = (c\ \gamma\ L)$
$a\ 0\ c\ 1\ \flat$ $\uparrow$ γ	$\delta\ (\gamma, 0) = (L)$
$a\ 0\ c\ 1\ \flat$ $\uparrow$ α	$\delta\ (\gamma, a) = (\alpha\ R)$
$a\ a\ c\ 1\ \flat$ $\uparrow$ β	$\delta\ (\alpha, 0) = (a\ \beta\ R)$
$a\ a\ c\ 1\ \flat$ $\uparrow$ β	$\delta\ (\beta, c) = (R)$
$a\ a\ c\ c\ \flat$ $\uparrow$ γ	$\delta\ (\beta, 1) = (c\ \gamma\ L)$
$a\ a\ c\ c\ \flat$ $\uparrow$ γ	$\delta\ (\gamma, c) = (L)$
$a\ a\ c\ c\ \flat$ $\uparrow$ α	$\delta\ (\gamma, a) = (\alpha\ R)$

$$\begin{array}{ll} a\ a\ c\ c\ \not b & \delta\ (\alpha, c) = (\delta\ R) \\ \quad\uparrow & \\ \quad\delta & \\ a\ a\ c\ c\ \not b & \delta\ (\delta, c) = (R) \\ \quad\quad\uparrow & \\ \quad\quad\delta & \\ a\ a\ c\ c\ \not b\ . & \delta\ (\delta, \not b) = (\epsilon\ N) \\ \quad\quad\quad\uparrow & \\ \quad\quad\quad\epsilon & \end{array}$$

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

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

$$\begin{array}{ll} \not b & \text{initial configuration} \\ \uparrow & \\ \alpha & \\ \not b & \delta\ (\alpha, \not b) = (\epsilon\ N) \\ \uparrow & \\ \epsilon & \end{array}$$

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, $\not b$.

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).

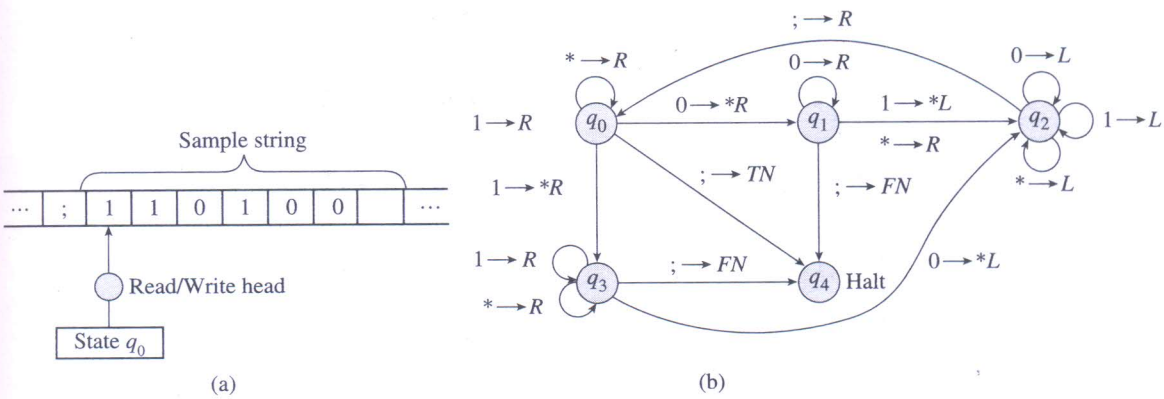


Figure 4.9 TM that recognizes strings containing equal number of 0's and 1's (a) Initial configuration (b) TG for TM that accepts all strings containing equal number of 0's and 1's

Simulation

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

```
; 1 1 0 1 0 0 ;        initial configuration
  ↑
  q0
; * 1 0 1 0 0 ;        δ (q0, 1) = (* q3 R)
    ↑
    q3
; * 1 0 1 0 0 ;        δ (q3, 1) = (R)
      ↑
      q3
; * 1 * 1 0 0 ;        δ (q3, 0) = (* q2 L)
    ↑
    q2
; * 1 * 1 0 0 ;        δ (q2, 1) = (L)
  ↑
  q2
```

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).

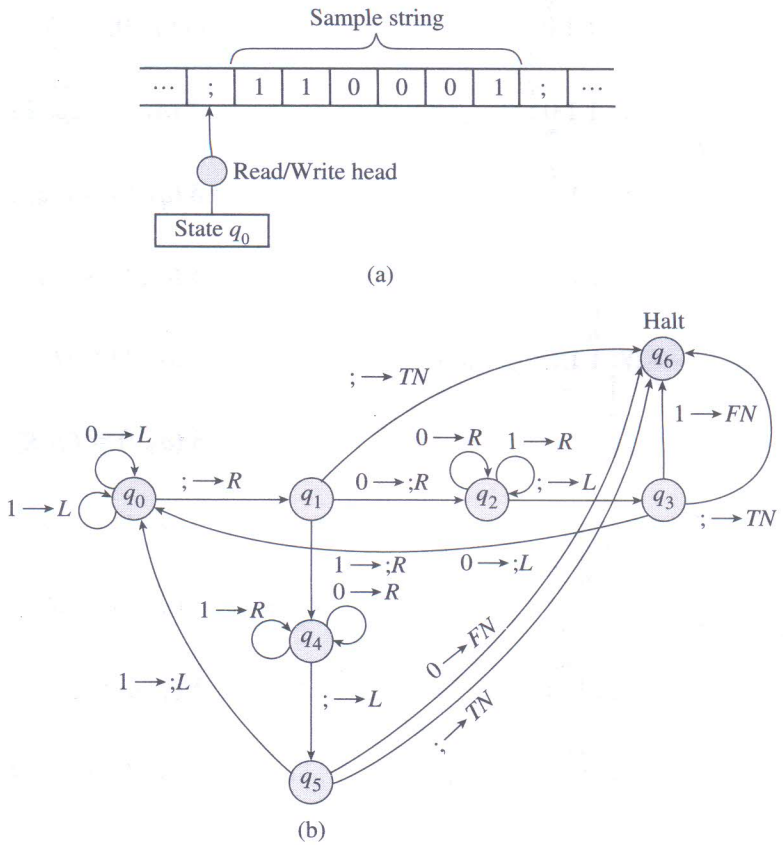


Figure 4.10 TM that recognizes binary palindromes (a) Initial configuration (b) TG for TM that recognizes binary palindromes

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.

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

; ; 1 ;
  ↑
  q_3
; ; F ;
  ↑
  q_6

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

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

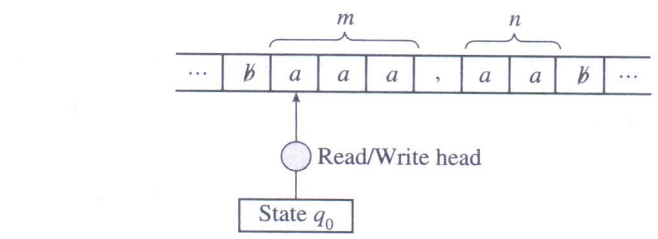
The output F indicates that the string '01' is not a palindrome string, and hence, is rejected by the TM.

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

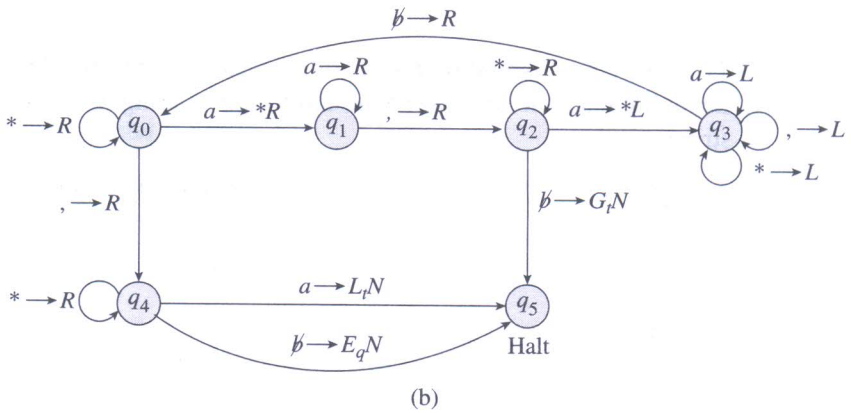
Solution This TM is a symbol manipulation system. For this, we need to represent the numbers in the unary format. If we consider $\Sigma = \{a\}$, then the numbers can be represented using that many a's. For example, '2' in unary format can be written as 'aa' and '5' can be written as '$aaaaa$'.

The initial configuration of the TM is assumed as shown in Fig. 4.11(a). Notice that the 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 read- - blank character $\not b$. In this case, the TM generates the output G_t to indicate that the first n than the second number n.

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

$\not b\ ,\ a\ \not b$ $\qquad\qquad \delta\ (q_0, ,) = (q_4\ R)$

$\uparrow$
q_4

$\not b\ ,\ L_t\ \not b$ $\qquad\qquad \delta\ (q_4, a) = (L_t\ q_5\ N)$

$\uparrow$
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'.

$\not b\ a\ ,\ a\ \not b$ $\qquad\qquad$ initial configuration

$\uparrow$
q_0

$\not b\ *\ ,\ a\ \not b$ $\qquad\qquad \delta\ (q_0, a) = (*\ q_1\ R)$

$\uparrow$
q_1

$\not b\ *\ ,\ a\ \not b$ $\qquad\qquad \delta\ (q_1, ,) = (q_2\ R)$

$\uparrow$
q_2

$\not b\ *\ ,\ *\ \not b$ $\qquad\qquad \delta\ (q_2, a) = (*\ q_3\ L)$

$\uparrow$
q_3

$\not b\ *\ ,\ *\ \not b$ $\qquad\qquad \delta\ (q_3, ,) = (L)$

$\uparrow$
q_3

$\not b\ *\ ,\ *\ \not b$ $\qquad\qquad \delta\ (q_3, *) = (L)$

$\uparrow$
q_3

$\not b\ *\ ,\ *\ \not b$ $\qquad\qquad \delta\ (q_3, \not b) = (q_0\ R)$

$\uparrow$
q_0

$\not b\ *\ ,\ *\ \not b$ $\qquad\qquad \delta\ (q_0, *) = (R)$

$\uparrow$
q_0

$\not b\ *\ ,\ *\ \not b$ $\qquad\qquad \delta\ (q_0, ,) = (q_4\ R)$

$\uparrow$
q_4

$\not b\ *\ ,\ *\ \not b$ $\qquad\qquad \delta\ (q_4, *) = (R)$

$\uparrow$
q_4

$\not b\ *\ ,\ *\ E_q$ $\qquad\qquad \delta\ (q_4, \not b) = (E_q\ q_5\ N)$

$\uparrow$
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).

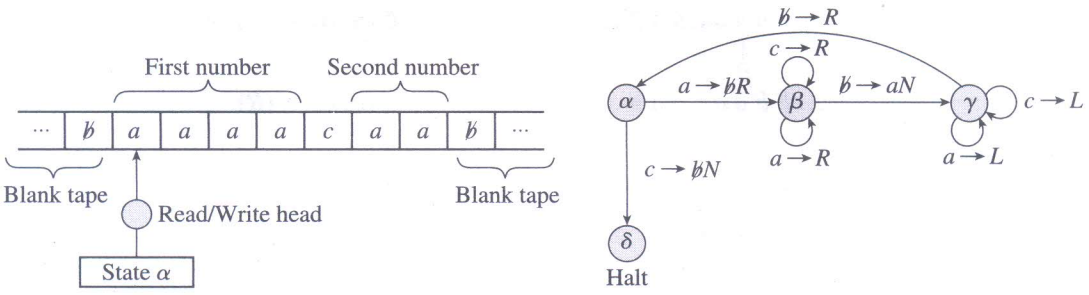


Figure 4.12 TM that performs addition of two unary numbers (a) Initial configuration (b) TG for TM that adds two unary numbers

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 ♭ 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 ♭.

Table 4.11 SFM for a TM that adds two unary numbers

S \ I	a	♭	c
α	♭βR	—	♭δN
β	R	aγN	R
γ	L	αR	L

For this TM, we have

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

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

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:

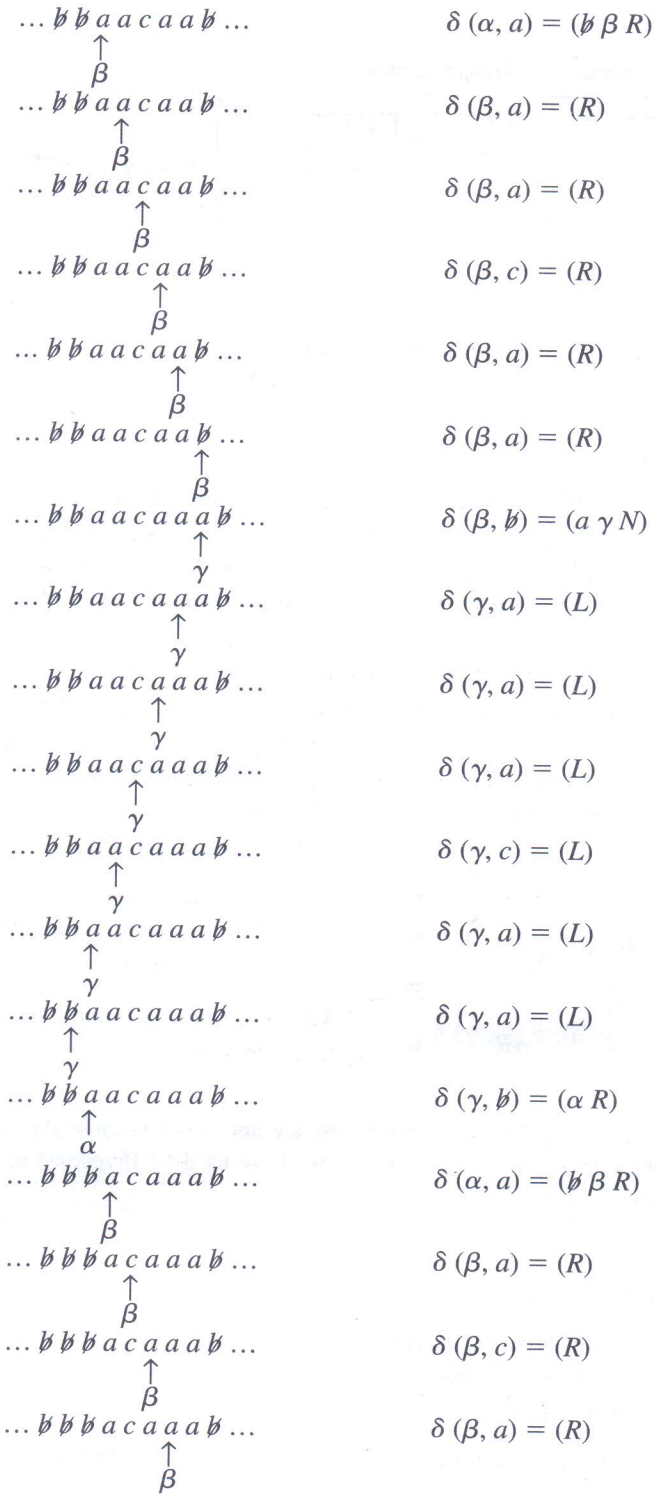
First number = 3 = '*aaa*'

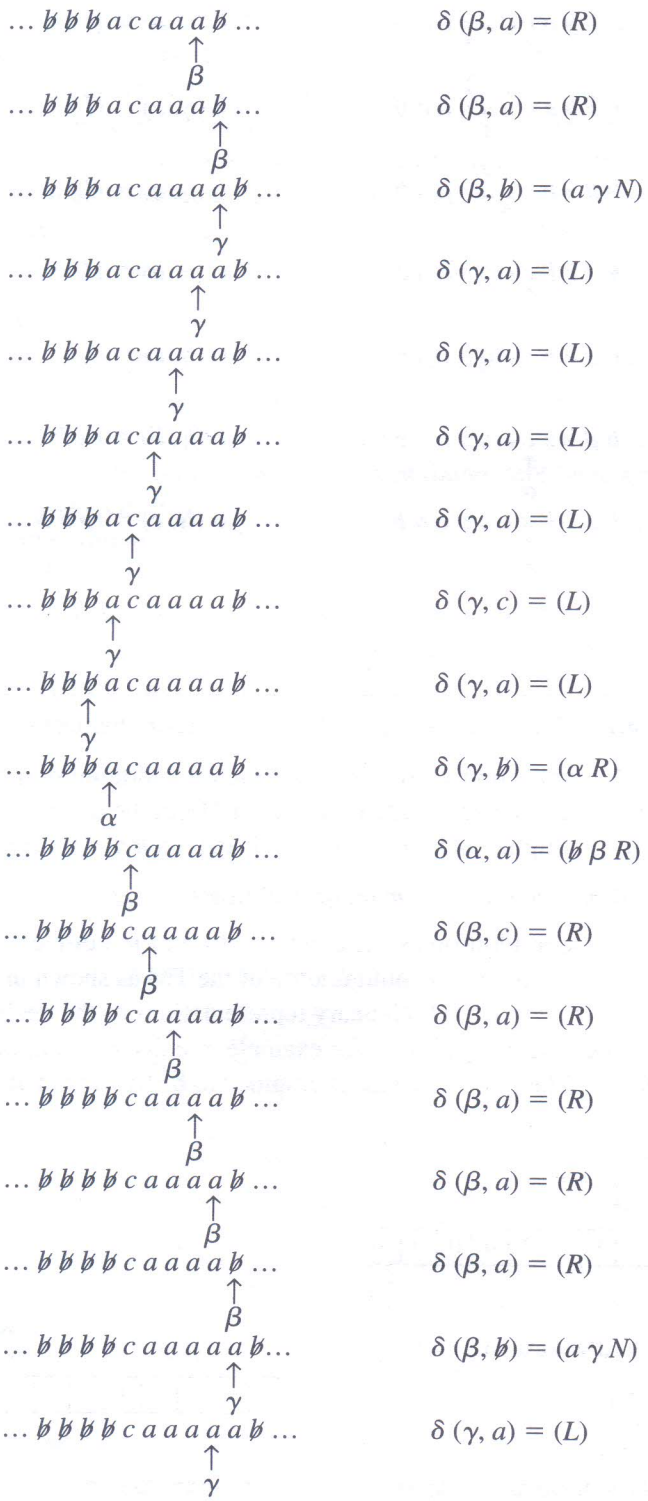
Second number = 2 = '*aa*'

... ♭ *a a a c a a* ♭ ... initial configuration

↑

α


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


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

$$\dots \not b \not b \not b \not b\ c\ a\ a\ \underset{\underset{\gamma}{\uparrow}}{a}\ a\ a\ \not b \dots \qquad \delta(\gamma, a) = (L)$$

$$\dots \not b \not b \not b \not b\ c\ a\ \underset{\underset{\gamma}{\uparrow}}{a}\ a\ a\ a\ \not b \dots \qquad \delta(\gamma, a) = (L)$$

$$\dots \not b \not b \not b \not b\ c\ \underset{\underset{\gamma}{\uparrow}}{a}\ a\ a\ a\ a\ \not b \dots \qquad \delta(\gamma, a) = (L)$$

$$\dots \not b \not b \not b \not b\ \underset{\underset{\gamma}{\uparrow}}{c}\ a\ a\ a\ a\ a\ \not b \dots \qquad \delta(\gamma, a) = (L)$$

$$\dots \not b \not b \not b \underset{\underset{\gamma}{\uparrow}}{\not b}\ c\ a\ a\ a\ a\ a\ \not b \dots \qquad \delta(\gamma, c) = (L)$$

$$\dots \not b \not b \not b \not b\ \underset{\underset{\alpha}{\uparrow}}{c}\ a\ a\ a\ a\ a\ \not b \dots \qquad \delta(\gamma, \not b) = (\alpha\, R)$$

$$\dots \not b \not b \not b \not b\ \underset{\underset{\delta}{\uparrow}}{\not b}\ a\ a\ a\ a\ a\ \not b \dots \qquad \delta(\alpha, c) = (\not b\, \delta\, N)$$

Thus, the addition of the two numbers '3 + 2' is completed, and that is indicated by 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. We 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:

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

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 multiplier and multiplicand are both unary representations of the decimal numbers. They are represented here using symbol 1; for example, the decimal number 2 in unary format is written as '11'. The rest of the tape is assumed to be filled with all 0's instead of $\not b$'s. The

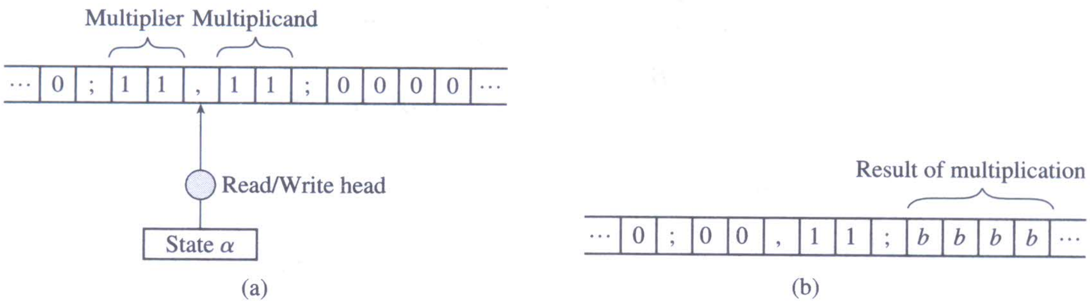


Figure 4.13 TM that multiplies two unary numbers (a) Initial configuration (b) Final configuration of TM (for the operation 2 × 2 = 4).

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 this 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 halt 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'.

... 0 ; 1 1 , 1 1 ; 0 0 0 0 ... initial configuration

α

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

α

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

β

(Decremented multiplier by 1)

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

β

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

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

β

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

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

β

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

γ

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

δ

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

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

δ

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

ϵ

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

ϵ

Note: If we want to design a non-destructive TM, we must ensure that the original arguments (or parameters) are kept intact. Therefore, we must construct the TM such that before entering into the halt state, it replaces all '0's in the multiplier by all '1's again.

Thus, we see that multiplication, which is not possible for an FSM, is achieved using a TM.

Example 4.11 Design a TM that finds the greatest common divisor (GCD) of two given numbers.

Solution Let the initial configuration of the TM be as shown in Fig. 4.14(a). We observe that the two numbers—in this example, we consider the numbers, 4 and 2—are stored onto the tape without any separator in between; further, we observe that both the numbers are represented in unary format using the symbol 1.

The initial state of the TM is α, and the head points to the last 1 of the unary representation of the first number among the given pair of numbers. As the sample pair is 4 and 2, we observe that the numbers in the tape are '1111' and '11'; and hence, the head points to the last 1 in '1111'.

Algorithm

Examine the numbers on the tape to find which of the two is larger. This is achieved by a repetitive subtraction process: First change one 1 in the first number to a; then change one 1 in the second number to b. Then return to the first number to change another 1 to a, and so on. This effectively subtracts the smaller number from the larger one.

If x is the smaller number and y is the larger number, then after the aforementioned processing, the pair x, y is changed to x and 'y - x'. This process is recursively carried out again on the newly-obtained pair. When the machine halts, it leaves one number in the pair as 0, and the other as the GCD.

The SFM for the TM is as shown in Table 4.13.

Table 4.13 SFM for a TM that finds the GCD of two given numbers

S \ I	0	1	a	b
α	δR	$a\beta N$	L	L
β	γL	$b\alpha N$	R	R
γ	αR	αR	$1L$	$0L$
δ	*Halt*	αL	$0R$	$1R$

For this TM, we have

$$I = \{0, 1, a, b\}$$
$$S = \{\alpha, \beta, \gamma, \delta\}$$

Halt state is not explicitly mentioned. One can introduce it if required, as we did for the examples so far. Wherever there is an entry 'Halt' in the transition Table 4.13, one can introduce a transition to such a halt state. The change is done to showcase a variation in the representation.

$$D = \{L, R, N\}$$

The transition diagram is as shown in Fig. 4.14(b).

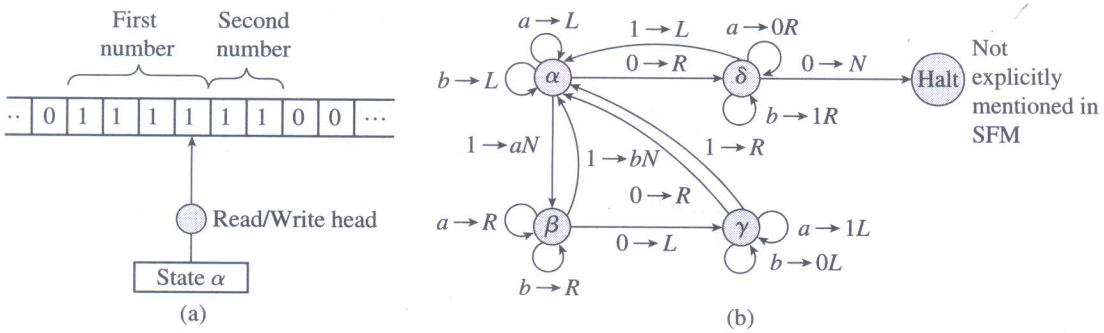


Figure 4.14 TM that finds the GCD of two given numbers (a) Initial configuration (b) TG for TM that finds GCD of two given numbers

Simulation

Let us simulate the working of the TM that we have constructed for the input numbers:

$x = 4 =$ '1111'
$y = 2 =$ '11'

... 0 1 1 1 1 1 1 0 ... ↑ α	initial configuration
... 0 1 1 1 a 1 1 0 ... ↑ β	$\delta(\alpha, 1) = (a\ \beta\ N)$
... 0 1 1 1 a 1 1 0 ... ↑ β	$\delta(\beta, a) = (R)$
... 0 1 1 1 a b 1 0 ... ↑ α	$\delta(\beta, 1) = (b\ \alpha\ N)$
... 0 1 1 1 a b 1 0 ... ↑ α	$\delta(\alpha, b) = (L)$
... 0 1 1 1 a b 1 0 ... ↑ α	$\delta(\alpha, a) = (L)$
... 0 1 1 a a b 1 0 ... ↑ β	$\delta(\alpha, 1) = (a\ \beta\ N)$
... 0 1 1 a a b 1 0 ... ↑ β	$\delta(\beta, a) = (R)$
... 0 1 1 a a b 1 0 ... ↑ β	$\delta(\beta, a) = (R)$
... 0 1 1 a a b 1 0 ... ↑ β	$\delta(\beta, b) = (R)$

$\ldots 0\, a\, a\, b\, b\, 0\, 0\, 0 \ldots$ $\quad \delta(\alpha, a) = (L)$

$\ldots 0\, a\, a\, b\, b\, 0\, 0\, 0 \ldots$ $\quad \delta(\alpha, 0) = (\delta\, R)$

$\ldots 0\, 0\, a\, b\, b\, 0\, 0\, 0 \ldots$ $\quad \delta(\delta, a) = (0\, R)$

$\ldots 0\, 0\, 0\, b\, b\, 0\, 0\, 0 \ldots$ $\quad \delta(\delta, a) = (0\, R)$

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

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

GCD

$\ldots 0\, 0\, 0\, 1\, 1\, 0\, 0\, 0 \ldots$ $\quad \delta(\delta, 0) = \text{Halt}$

therefore, GCD = "1 1" = 2

As the second number of the newly-obtained pair is 0, the GCD of 4 and 2 is represented by the number of 1's remaining on the tape. Hence, the answer is 2.

Example 4.12 Design a TM that divides one number by the other, and finds the result of the division as well as the remainder if any.

Solution Let us represent the numbers in the unary format using the symbol 1. For example, '3' can be represented as '111' in unary format. Let us assume the initial configuration of the TM as shown in the Fig. 4.15.

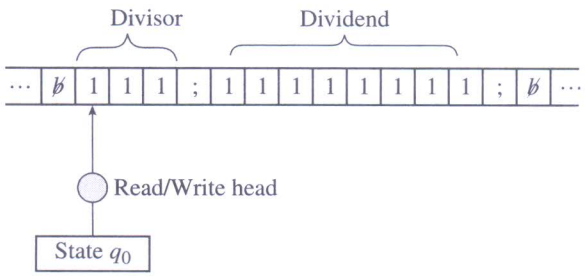


Figure 4.15 Initial configuration of a TM that performs number division

We observe that the initial state of the TM is q_0; and that the divisor and dividend are separated by a comma ',' . The machine head initially points to the start of the divisor.

After the division is performed, when the machine halts, the result of the division, which is also represented in unary format using the symbol 1, is written onto the tape immediately after the right end-marker ';'. The number of a's in the divisor area represents the remainder of the division.

Algorithm

Division is repetitive subtraction of the divisor from the dividend. In the previous example, we have used repetitive subtraction to find the GCD of two numbers. Similarly, in Example 4.8, we designed a TM that compares two positive integers using subtraction. We now use a similar algorithm for division.

Replace one 1 of the divisor by a; and for each a in the divisor, replace one 1 of the dividend by b. Repeat the process until the whole divisor is subtracted from the dividend. Write 1 at the right end indicating that one cycle of subtraction is complete.

Iteratively, subtract the divisor again from the dividend till you have exhausted the dividend completely. At the end of every iteration, write 1 at the right end; after all the iterations are completed, the number of 1's at the right end represents the result of the division.

If the entire divisor cannot be subtracted from the dividend, then the number of a's that replaces the 1's from the divisor represents the remainder of the division.

The SFM for the TM is as shown in Table 4.14.

Table 4.14 SFM for a TM that performs number division

S \ I	1	,	;	∅	a	b
q_0	aq_1R	q_4R	—	—	—	—
q_1	R	q_2R	—	—	—	—
q_2	bq_3L	—	q_7L	—	—	R
q_3	L	L	—	—	q_0R	L
q_4	R	—	R	$1q_5R$	—	R
q_5	L	q_6L	L	—	—	L
q_6	—	—	—	q_0R	$1L$	—
q_7	—	q_8L	—	—	—	L
q_8	L	—	—	—	$1q_9N$	—
q_9	—	—	—	—	—	—

For this TM, we have:

$I = \{1, , , ; , \not\emptyset, a, b\}$
$S = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9 = \text{halt}\}$
$D = \{L, R, N\}$

Simulation

Let us simulate the working of TM for

Divisor = 3 = '111'
Dividend = 8 = '11111111'

$\ldots \not\emptyset\ 1\ 1\ 1\ ,\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ ;\ \not\emptyset \ldots$ initial configuration
$\uparrow$
q_0

$\ldots \not\emptyset\ a\ 1\ 1\ ,\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ ;\ \not\emptyset \ldots$ $\delta\ (q_0, 1) = (a\ q_1\ R)$
$\uparrow$
q_1

(Subtraction of the divisor from dividend begins)

The number of *a*'s in the divisor area represents the remainder, and the number of 1's after the right end-marker ';' represents the result of the division. Hence, the result obtained when 8 is divided by 3 is 2, and the remainder is 2.

Example 4.13 Design a TM to find the value of $\log_2(n)$, where n is any binary number and a perfect power of 2.

Solution It is given that n is a binary number and a perfect power of 2.
We know:

$2^0 = 1 = 1$ (binary)
$2^1 = 2 = 10$ (binary)
$2^2 = 4 = 100$ (binary)
$2^3 = 8 = 1000$ (binary)

From the aforementioned listing, we have

$\log_2(2^0) = \log_2(1) = 0$ (zero number of 0's after first 1 in the binary format)
$\log_2(2^1) = \log_2(2) = 1$ (one 0 after first 1 in the binary format, which is 10)
$\log_2(2^2) = 2\log_2(2) = 2 \times 1 = 2$ (two 0's after first 1 in binary format, which is 100)

The conclusion is that, the value of $\log_2(n)$, where n is a binary number and a perfect power of 2, is equal to the number of 0's after the first 1 in the given number n.

Algorithm

Count of the number of zeros after the beginning '1' of the binary number '*n*'; it gives us the required answer.

1. Read the first digit, that is, 1, and ignore it.
2. Read the following 0 from the string that follows the 1, replace it by a symbol, *a*, and write a symbol, *c*, after the right end-marker.
3. Repeat the procedure till all the 0's that follow 1 get replaced by *a*'s.
4. The number of *c*'s written after the right end-marker gives the required value of $\log_2(n)$.

Thus, the input string is in binary format and the resultant value is in unary format using symbol *c*.

Note: We could use a simpler algorithm as well, which simply replaces the first digit 1 with a blank character ♭; the remaining number of 0's gives the required answer. However, this would destroy the input parameter, and hence we do not use this algorithm.

The initial configuration of the required TM is shown in Fig. 4.16(a), and its transition graph is shown in Fig. 4.16(b).

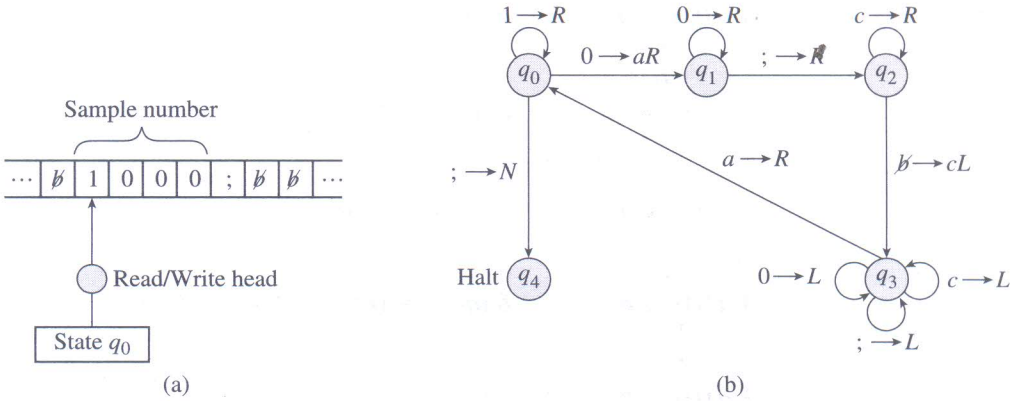


Figure 4.16 Finding the value of $\log_2(n)$ (a) Initial configuration (b) TG for TM that finds $\log_2(n)$

The SFM is created as shown in Table 4.15.

Table 4.15 SFM for a TM that finds $\log_2$ (n)

S \ I	1	0	;	a	c	⊔
q_0	R	aq_1R	q_4N	—	—	—
q_1	—	R	q_2R	—	—	—
q_2	—	—	—	—	R	cq_3L
q_3	—	L	L	q_0R	L	—
q_4	—	—	—	—	—	—

Simulation

Let us simulate the working of the TM that we have constructed for $n = 1000$ ($= 8 = 2^3$). We know that for this number n, $\log_2(n) = 3$, which can be represented as ccc in unary form.

1 0 0 0 ; ⊔ ⊔ … initial configuration
↑
q_0

1 0 0 0 ; ⊔ ⊔ … $\delta(q_0, 1) = (R)$
↑
q_0

1 a 0 0 ; ⊔ ⊔ … $\delta(q_0, 0) = (a\, q_1\, R)$
↑
q_1

1 a 0 0 ; ⊔ ⊔ … $\delta(q_1, 0) = (R)$
↑
q_1

$1\ a\ a\ a\ ;\ c\ c\ \not b\ \not b\ \ldots$ $\qquad \delta\ (q_0, 0) = (a\ q_1\ R)$
$\uparrow$
q_1

$1\ a\ a\ a\ ;\ c\ c\ \not b\ \not b\ \ldots$ $\qquad \delta\ (q_1, ;) = (q_2\ R)$
$\uparrow$
q_2

$1\ a\ a\ a\ ;\ c\ c\ \not b\ \not b\ \ldots$ $\qquad \delta\ (q_2, c) = (R)$
$\uparrow$
q_2

$1\ a\ a\ a\ ;\ c\ c\ \not b\ \not b\ \ldots$ $\qquad \delta\ (q_2, c) = (R)$
$\uparrow$
q_2

$1\ a\ a\ a\ ;\ c\ c\ c\ \not b\ \ldots$ $\qquad \delta\ (q_2, \not b) = (c\ q_3\ L)$
$\uparrow$
q_3

$1\ a\ a\ a\ ;\ c\ c\ c\ \not b\ \ldots$ $\qquad \delta\ (q_3, c) = (L)$
$\uparrow$
q_3

$1\ a\ a\ a\ ;\ c\ c\ c\ \not b\ \ldots$ $\qquad \delta\ (q_3, c) = (L)$
$\uparrow$
q_3

$1\ a\ a\ a\ ;\ c\ c\ c\ \not b\ \ldots$ $\qquad \delta\ (q_3, ;) = (L)$
$\uparrow$
q_3

$1\ a\ a\ a\ ;\ c\ c\ c\ \not b\ \ldots$ $\qquad \delta\ (q_3, a) = (q_0\ R)$
$\uparrow$
q_0

$1\ a\ a\ a\ ;\ c\ c\ c\ \not b\ \ldots$ $\qquad \delta\ (q_0, ;) = (q_4\ N)$
$\uparrow$
q_4

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

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

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

$\Gamma = \{1, 0, a, c, ;, \not b\}$
$Q = \{q_0, q_1, q_2, q_3, q_4 = \text{halt}\}$

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).

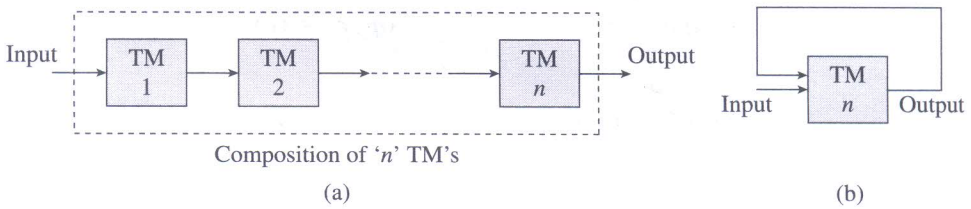


Figure 4.17 Composite and iterative TMs (a) Composite TM (CTM) (b) Iterative TM (ITM)

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.

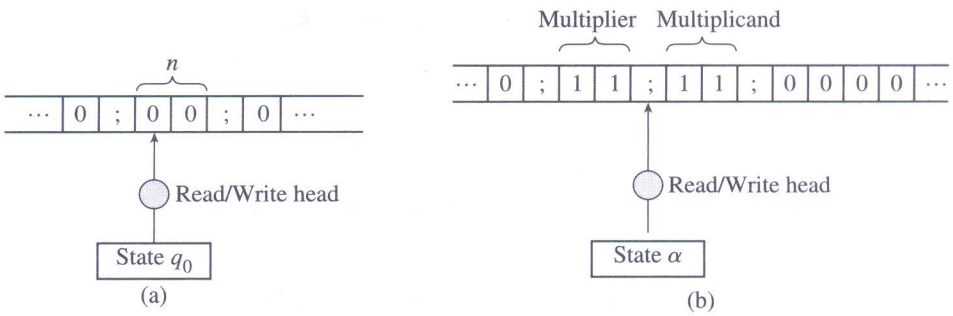


Figure 4.18 TM that computes n^2 ($n \geq 0$) (a) Initial configuration (b) Configuration after copying n onto tape

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 $n = 2$.

... 0 ; 0 0 ; 0 0 ... initial configuration
↑
q_0

... 0 ; 1 0 ; 0 0 ... $\delta(q_0, 0) = (1\ q_1\ R)$
↑
q_1