a) Show how, given a TM that computes f, you can construct a TM that accepts the graph of f as a language.

b) Show how, given a TM that accepts the graph of f, you can construct a TM that computes f.

c) A function is said to be *partial* if it may be undefined for some arguments. If we extend the ideas of this exercise to partial functions, then we do not require that the TM computing f halts if its input x is one of the integers for which $f(x)$ is not defined. Do your constructions for parts (a) and (b) work if the function f is partial? If not, explain how you could modify the construction to make it work.

Exercise 8.2.5: Consider the Turing machine

$$M = (\{q_0, q_1, q_2, q_f\}, \{0,1\}, \{0,1,B\}, \delta, q_0, B, \{q_f\})$$

Informally but clearly describe the language $L(M)$ if δ consists of the following sets of rules:

* a) $\delta(q_0, 0) = (q_1, 1, R)$; $\delta(q_1, 1) = (q_0, 0, R)$; $\delta(q_1, B) = (q_f, B, R)$.

b) $\delta(q_0, 0) = (q_0, B, R)$; $\delta(q_0, 1) = (q_1, B, R)$; $\delta(q_1, 1) = (q_1, B, R)$; $\delta(q_1, B) = (q_f, B, R)$.

! c) $\delta(q_0, 0) = (q_1, 1, R)$; $\delta(q_1, 1) = (q_2, 0, L)$; $\delta(q_2, 1) = (q_0, 1, R)$; $\delta(q_1, B) = (q_f, B, R)$.

8.3 Programming Techniques for Turing Machines

Our goal is to give you a sense of how a Turing machine can be used to compute in a manner not unlike that of a conventional computer. Eventually, we want to convince you that a TM is exactly as powerful as a conventional computer. In particular, we shall learn that the Turing machine can perform the sort of calculations on other Turing machines that we saw performed in Section 8.1.2 by a program that examined other programs. This "introspective" ability of both Turing machines and computer programs is what enables us to prove problems undecidable.

To make the ability of a TM clearer, we shall present a number of examples of how we might think of the tape and finite control of the Turing machine. None of these tricks extend the basic model of the TM; they are only notational conveniences. Later, we shall use them to simulate extended Turing-machine models that have additional features — for instance, more than one tape — by the basic TM model.

8.3.1 Storage in the State

We can use the finite control not only to represent a position in the "program" of the Turing machine, but to hold a finite amount of data. Figure 8.13 suggests this technique (as well as another idea: multiple tracks). There, we see the finite control consisting of not only a "control" state q, but three data elements A, B, and C. The technique requires no extension to the TM model; we merely think of the state as a tuple. In the case of Fig. 8.13, we should think of the state as $[q, A, B, C]$. Regarding states this way allows us to describe transitions in a more systematic way, often making the strategy behind the TM program more transparent.

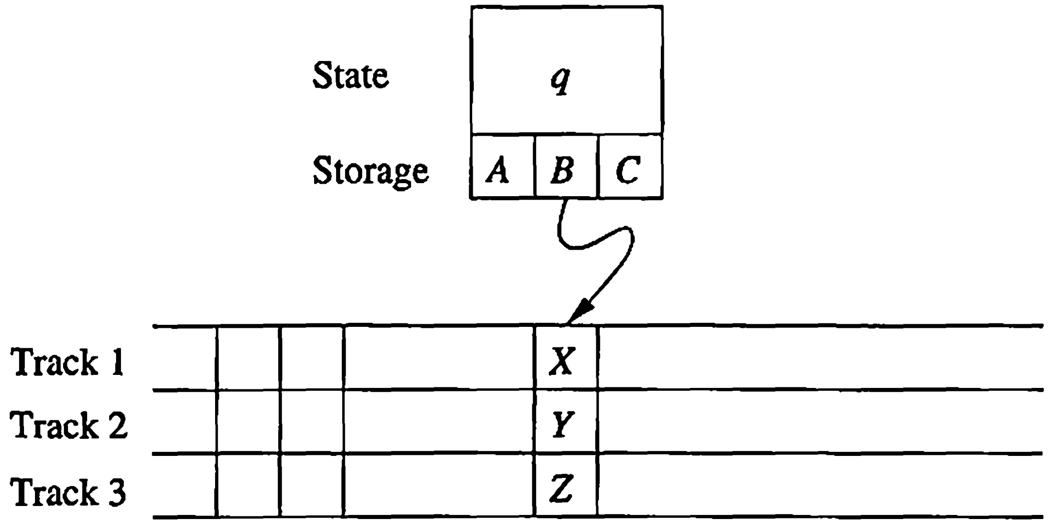

Figure 8.13: A Turing machine viewed as having finite-control storage and multiple tracks

Example 8.6: We shall design a TM

$$M = (Q, \{0,1\}, \{0,1,B\}, \delta, [q_0, B], \{[q_1, B]\})$$

that remembers in its finite control the first symbol (0 or 1) that it sees, and checks that it does not appear elsewhere on its input. Thus, M accepts the language $\mathbf{01^*+10^*}$. Accepting regular languages such as this one does not stress the ability of Turing machines, but it will serve as a simple demonstration.

The set of states Q is $\{q_0, q_1\} \times \{0, 1, B\}$. That is, the states may be thought of as pairs with two components:

a) A control portion, q_0 or q_1, that remembers what the TM is doing. Control state q_0 indicates that M has not yet read its first symbol, while q_1 indicates that it *has* read the symbol, and is checking that it does not appear elsewhere, by moving right and hoping to reach a blank cell.

b) A data portion, which remembers the first symbol seen, which must be 0 or 1. The symbol B in this component means that no symbol has been read.

The transition function δ of M is as follows:

1. $\delta([q_0, B], a) = ([q_1, a], a, R)$ for $a = 0$ or $a = 1$. Initially, q_0 is the control state, and the data portion of the state is B. The symbol scanned is copied into the second component of the state, and M moves right, entering control state q_1 as it does so.

2. $\delta([q_1, a], \overline{a}) = ([q_1, a], \overline{a}, R)$ where $\overline{a}$ is the "complement" of a, that is, 0 if $a = 1$ and 1 if $a = 0$. In state q_1, M skips over each symbol 0 or 1 that is different from the one it has stored in its state, and continues moving right.

3. $\delta([q_1, a], B) = ([q_1, B], B, R)$ for $a = 0$ or $a = 1$. If M reaches the first blank, it enters the accepting state $[q_1, B]$.

Notice that M has no definition for $\delta([q_1, a], a)$ for $a = 0$ or $a = 1$. Thus, if M encounters a second occurrence of the symbol it stored initially in its finite control, it halts without having entered the accepting state. □

8.3.2 Multiple Tracks

Another useful "trick" is to think of the tape of a Turing machine as composed of several tracks. Each track can hold one symbol, and the tape alphabet of the TM consists of tuples, with one component for each "track." Thus, for instance, the cell scanned by the tape head in Fig. 8.13 contains the symbol $[X, Y, Z]$. Like the technique of storage in the finite control, using multiple tracks does not extend what the Turing machine can do. It is simply a way to view tape symbols and to imagine that they have a useful structure.

Example 8.7: A common use of multiple tracks is to treat one track as holding the data and a second track as holding a mark. We can check off each symbol as we "use" it, or we can keep track of a small number of positions within the data by marking only those positions. Examples 8.2 and 8.4 were two instances of this technique, but in neither example did we think explicitly of the tape as if it were composed of tracks. In the present example, we shall use a second track explicitly to recognize the non-context-free language

$$L_{wcw} = \{wcw \mid w \text{ is in } (\mathbf{0} + \mathbf{1})^+\}$$

The Turing machine we shall design is:

$$M = (Q, \Sigma, \Gamma, \delta, [q_1, B], [B, B], \{[q_9, B]\})$$

where:

Q: The set of states is $\{q_1, q_2, \ldots, q_9\} \times \{0, 1, B\}$, that is, pairs consisting of a control state q_i and a data component: 0, 1, or blank. We again use the technique of storage in the finite control, as we allow the state to remember an input symbol 0 or 1.

Γ: The set of tape symbols is $\{B, *\} \times \{0, 1, c, B\}$. The first component, or track, can be either blank or "checked," represented by the symbols B and $*$, respectively. We use the $*$ to check off symbols of the first and second groups of 0's and 1's, eventually confirming that the string to the left of the center marker c is the same as the string to its right. The second component of the tape symbol is what we think of as the tape symbol itself. That is, we may think of the symbol $[B, X]$ as if it were the tape symbol X, for $X = 0, 1, c, B$.

Σ: The input symbols are $[B, 0]$ and $[B, 1]$, which, as just mentioned, we identify with 0 and 1, respectively.

δ: The transition function δ is defined by the following rules, in which a and b each may stand for either 0 or 1.

1. $\delta([q_1, B], [B, a]) = ([q_2, a], [*, a], R)$. In the initial state, M picks up the symbol a (which can be either 0 or 1), stores it in its finite control, goes to control state q_2, "checks off" the symbol it just scanned, and moves right. Notice that by changing the first component of the tape symbol from B to $*$, it performs the check-off.
2. $\delta([q_2, a], [B, b]) = ([q_2, a], [B, b], R)$. M moves right, looking for the symbol c. Remember that a and b can each be either 0 or 1, independently, but cannot be c.
3. $\delta([q_2, a], [B, c]) = ([q_3, a], [B, c], R)$. When M finds the c, it continues to move right, but changes to control state q_3.
4. $\delta([q_3, a], [*, b]) = ([q_3, a], [*, b], R)$. In state q_3, M continues past all checked symbols.
5. $\delta([q_3, a], [B, a]) = ([q_4, B], [*, a], L)$. If the first unchecked symbol that M finds is the same as the symbol in its finite control, it checks this symbol, because it has matched the corresponding symbol from the first block of 0's and 1's. M goes to control state q_4, dropping the symbol from its finite control, and starts moving left.
6. $\delta([q_4, B], [*, a]) = ([q_4, B], [*, a], L)$. M moves left over checked symbols.
7. $\delta([q_4, B], [B, c]) = ([q_5, B], [B, c], L)$. When M encounters the symbol c, it switches to state q_5 and continues left. In state q_5, M must make a decision, depending on whether or not the symbol immediately to the left of the c is checked or unchecked. If checked, then we have already considered the entire first block of 0's and 1's — those to the left of the c. We must make sure that all the 0's and 1's to the right of the c are also checked, and accept if no unchecked symbols remain to the right of the c. If the symbol immediately to the left of the c is unchecked, we find the leftmost unchecked symbol, pick it up, and start the cycle that began in state q_1.

8. $\delta([q_5, B], [B, a]) = ([q_6, B], [B, a], L)$. This branch covers the case where the symbol to the left of c is unchecked. M goes to state q_6 and continues left, looking for a checked symbol.

9. $\delta([q_6, B], [B, a]) = ([q_6, B], [B, a], L)$. As long as symbols are unchecked, M remains in state q_6 and proceeds left.

10. $\delta([q_6, B], [*, a]) = ([q_1, B], [*, a], R)$. When the checked symbol is found, M enters state q_1 and moves right to pick up the first unchecked symbol.

11. $\delta([q_5, B], [*, a]) = ([q_7, B], [*, a], R)$. Now, let us pick up the branch from state q_5 where we have just moved left from the c and find a checked symbol. We start moving right again, entering state q_7.

12. $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$. In state q_7 we shall surely see the c. We enter state q_8 as we do so, and proceed right.

13. $\delta([q_8, B], [*, a]) = ([q_8, B], [*, a], R)$. M moves right in state q_8, skipping over any checked 0's or 1's that it finds.

14. $\delta([q_8, B], [B, B]) = ([q_9, B], [B, B], R)$. If M reaches a blank cell is state q_8 without encountering any unchecked 0 or 1, then M accepts. If M first finds an unchecked 0 or 1, then the blocks before and after the c do not match, and M halts without accepting.

□

8.3.3 Subroutines

As with programs in general, it helps to think of Turing machines as built from a collection of interacting components, or "subroutines." A Turing-machine subroutine is a set of states that perform some useful process. This set of states includes a start state and another state that temporarily has no moves, and that serves as the "return" state to pass control to whatever other set of states called the subroutine. The "call" of a subroutine occurs whenever there is a transition to its initial state. Since the TM has no mechanism for remembering a "return address," that is, a state to go to after it finishes, should our design of a TM call for one subroutine to be called from several states, we can make copies of the subroutine, using a new set of states for each copy. The "calls" are made to the start states of different copies of the subroutine, and each copy "returns" to a different state.

Example 8.8: We shall design a TM to implement the function "multiplication." That is, our TM will start with 0^m10^n1 on its tape, and will end with 0^{mn} on the tape. An outline of the strategy is:

1. The tape will, in general, have one nonblank string of the form $0^i10^n10^{kn}$ for some k.

2. In one basic step, we change a 0 in the first group to B and add n 0's to the last group, giving us a string of the form $0^{i-1}10^n10^{(k+1)n}$.

3. As a result, we copy the group of n 0's to the end m times, once each time we change a 0 in the first group to B. When the first group of 0's is completely changed to blanks, there will be mn 0's in the last group.

4. The final step is to change the leading 10^n1 to blanks, and we are done.

The heart of this algorithm is a subroutine, which we call Copy. This subroutine implements step (2) above, copying the block of n 0's to the end. More precisely, Copy converts an ID of the form $0^{m-k}1q_10^n10^{(k-1)n}$ to ID $0^{m-k}1q_50^n10^{kn}$. Figure 8.14 shows the transitions of subroutine Copy. This subroutine marks the first 0 with an X, moves right in state q_2 until it finds a blank, copies the 0 there, and moves left in state q_3 to find the marker X. It repeats this cycle until in state q_1 it finds a 1 instead of a 0. At that point, it uses state q_4 to change the X's back to 0's, and ends in state q_5.

The complete multiplication Turing machine starts in state q_0. The first thing it does is go, in several steps, from ID $q_00^m10^n$ to ID $0^{m-1}1q_10^n1$. The transitions needed are shown in the portion of Fig. 8.15 to the left of the subroutine call; these transitions involve states q_0 and q_6 only.

Then, to the right of the subroutine call in Fig. 8.15 we see states q_7 through q_{12}. The purpose of states q_7, q_8, and q_9 is to take control after Copy has just copied a block of n 0's, and is in ID $0^{m-k}1q_50^n10^{kn}$. Eventually, these states bring us to state $q_00^{m-k}10^n10^{kn}$. At that point, the cycle starts again, and Copy is called to copy the block of n 0's again.

As an exception, in state q_8 the TM may find that all m 0's have been changed to blanks (i.e., $k = m$). In that case, a transition to state q_{10} occurs. This state, with the help of state q_{11}, changes the leading 10^n1 to blanks and enters the halting state q_{12}. At this point, the TM is in ID $q_{12}0^{mn}$, and its job is done. □

8.3.4 Exercises for Section 8.3

! **Exercise 8.3.1:** Redesign your Turing machines from Exercise 8.2.2 to take advantage of the programming techniques discussed in Section 8.3.

! **Exercise 8.3.2:** A common operation in Turing-machine programs involves "shifting over." Ideally, we would like to create an extra cell at the current head position, in which we could store some character. However, we cannot edit the tape in this way. Rather, we need to move the contents of each of the cells to the right of the current head position one cell right, and then find our way back to the current head position. Show how to perform this operation. *Hint*: Leave a special symbol to mark the position to which the head must return.

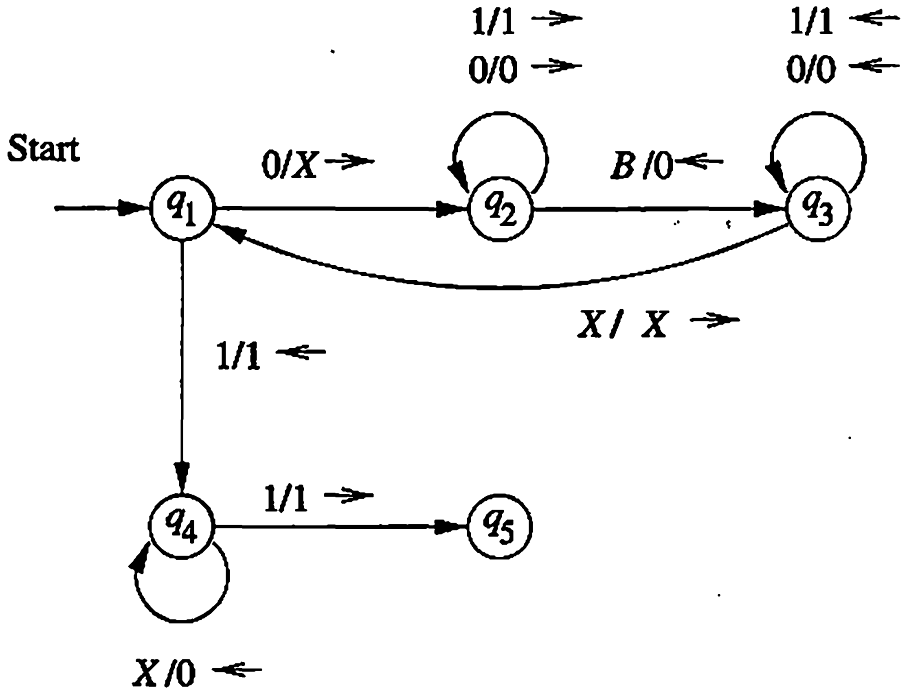

Figure 8.14: The subroutine Copy

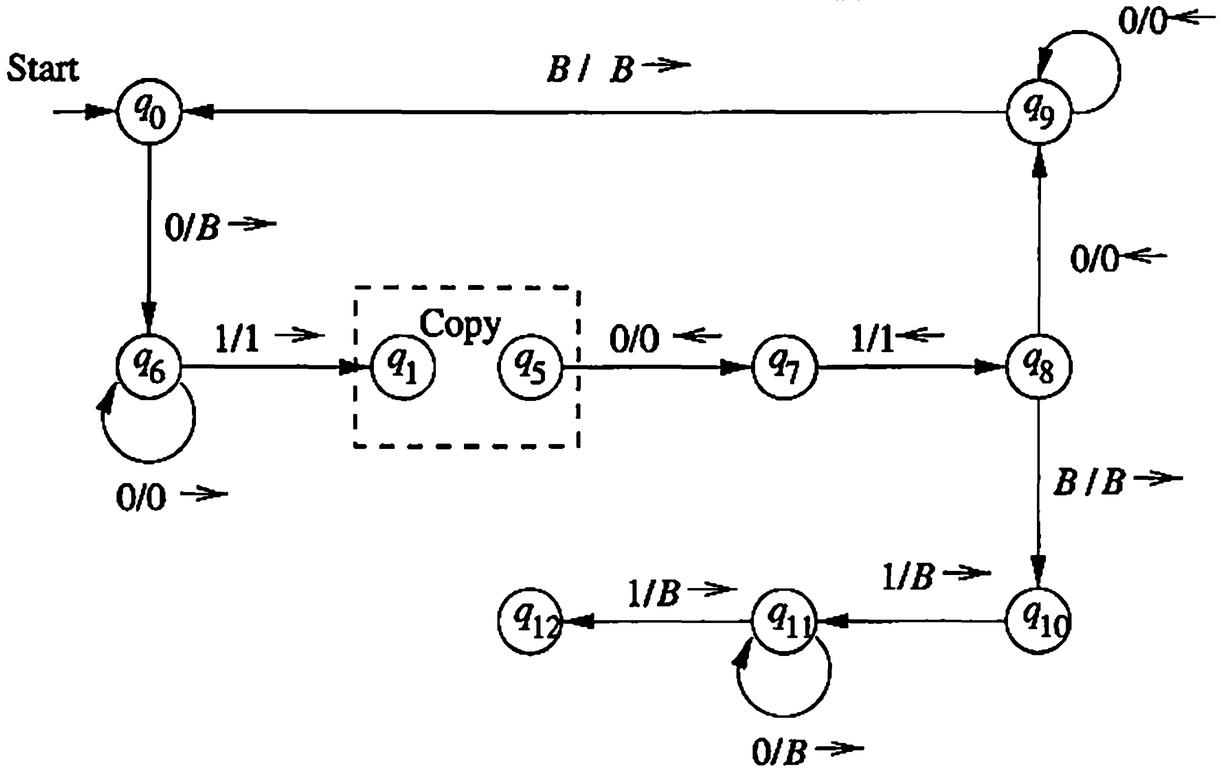

Figure 8.15: The complete multiplication program uses the subroutine Copy