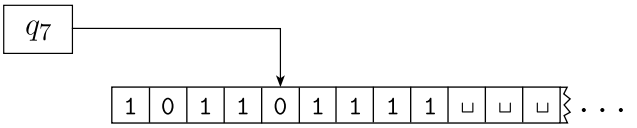

FIGURE 3.4
A Turing machine with configuration $1011q_701111$

Here we formalize our intuitive understanding of the way that a Turing machine computes. Say that configuration C_1 ***yields*** configuration C_2 if the Turing machine can legally go from C_1 to C_2 in a single step. We define this notion formally as follows.

Suppose that we have a, b, and c in Γ, as well as u and v in Γ^* and states q_i and q_j. In that case, $ua\,q_i\,bv$ and $u\,q_j\,acv$ are two configurations. Say that

$$ua\,q_i\,bv \quad \text{yields} \quad u\,q_j\,acv$$

if in the transition function $\delta(q_i, b) = (q_j, c, \mathrm{L})$. That handles the case where the Turing machine moves leftward. For a rightward move, say that

$$ua\,q_i\,bv \quad \text{yields} \quad uac\,q_j\,v$$

if $\delta(q_i, b) = (q_j, c, \mathrm{R})$.

Special cases occur when the head is at one of the ends of the configuration. For the left-hand end, the configuration $q_i\,bv$ yields $q_j\,cv$ if the transition is left-moving (because we prevent the machine from going off the left-hand end of the tape), and it yields $c\,q_j v$ for the right-moving transition. For the right-hand end, the configuration $ua\,q_i$ is equivalent to $ua\,q_i\,\sqcup$ because we assume that blanks follow the part of the tape represented in the configuration. Thus we can handle this case as before, with the head no longer at the right-hand end.

The ***start configuration*** of M on input w is the configuration $q_0\,w$, which indicates that the machine is in the start state q_0 with its head at the leftmost position on the tape. In an ***accepting configuration***, the state of the configuration is q_{accept}. In a ***rejecting configuration***, the state of the configuration is q_{reject}. Accepting and rejecting configurations are ***halting configurations*** and do not yield further configurations. Because the machine is defined to halt when in the states q_{accept} and q_{reject}, we equivalently could have defined the transition function to have the more complicated form $\delta\colon Q' \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$, where Q' is Q without q_{accept} and q_{reject}. A Turing machine M ***accepts*** input w if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists, where

1. C_1 is the start configuration of M on input w,
2. each C_i yields C_{i+1}, and
3. C_k is an accepting configuration.

The collection of strings that M accepts is ***the language of M***, or ***the language recognized by M***, denoted $L(M)$.

DEFINITION 3.5

Call a language ***Turing-recognizable*** if some Turing machine recognizes it.[1]

When we start a Turing machine on an input, three outcomes are possible. The machine may *accept*, *reject*, or *loop*. By ***loop*** we mean that the machine simply does not halt. Looping may entail any simple or complex behavior that never leads to a halting state.

A Turing machine M can fail to accept an input by entering the q_{reject} state and rejecting, or by looping. Sometimes distinguishing a machine that is looping from one that is merely taking a long time is difficult. For this reason, we prefer Turing machines that halt on all inputs; such machines never loop. These machines are called ***deciders*** because they always make a decision to accept or reject. A decider that recognizes some language also is said to ***decide*** that language.

DEFINITION 3.6

Call a language ***Turing-decidable*** or simply ***decidable*** if some Turing machine decides it.[2]

Next, we give examples of decidable languages. Every decidable language is Turing-recognizable. We present examples of languages that are Turing-recognizable but not decidable after we develop a technique for proving undecidability in Chapter 4.

EXAMPLES OF TURING MACHINES

As we did for finite and pushdown automata, we can formally describe a particular Turing machine by specifying each of its seven parts. However, going to that level of detail can be cumbersome for all but the tiniest Turing machines. Accordingly, we won't spend much time giving such descriptions. Mostly we

[1] It is called a ***recursively enumerable language*** in some other textbooks.
[2] It is called a ***recursive language*** in some other textbooks.

will give only higher level descriptions because they are precise enough for our purposes and are much easier to understand. Nevertheless, it is important to remember that every higher level description is actually just shorthand for its formal counterpart. With patience and care we could describe any of the Turing machines in this book in complete formal detail.

To help you make the connection between the formal descriptions and the higher level descriptions, we give state diagrams in the next two examples. You may skip over them if you already feel comfortable with this connection.

EXAMPLE 3.7

Here we describe a Turing machine (TM) M_2 that decides $A = \{0^{2^n} \mid n \geq 0\}$, the language consisting of all strings of 0s whose length is a power of 2.

M_2 = "On input string w:

1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, *accept*.
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, *reject*.
4. Return the head to the left-hand end of the tape.
5. Go to stage 1."

Each iteration of stage 1 cuts the number of 0s in half. As the machine sweeps across the tape in stage 1, it keeps track of whether the number of 0s seen is even or odd. If that number is odd and greater than 1, the original number of 0s in the input could not have been a power of 2. Therefore, the machine rejects in this instance. However, if the number of 0s seen is 1, the original number must have been a power of 2. So in this case, the machine accepts.

Now we give the formal description of $M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$:

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0\}$, and
- $\Gamma = \{0, \text{x}, \sqcup\}$.
- We describe δ with a state diagram (see Figure 3.8).
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}, respectively.

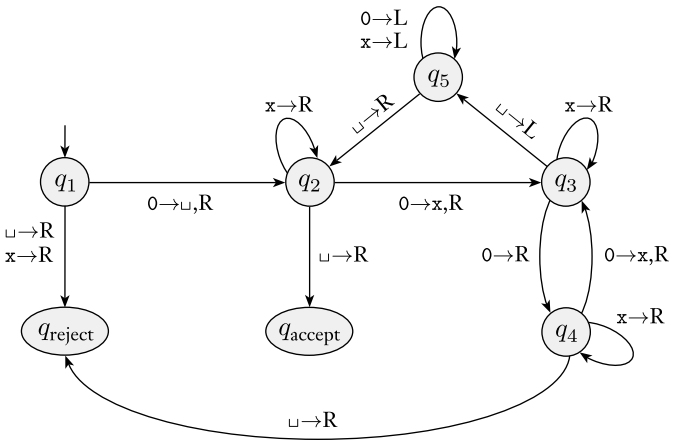

FIGURE 3.8
State diagram for Turing machine M_2

In this state diagram, the label 0→␣,R appears on the transition from q_1 to q_2. This label signifies that when in state q_1 with the head reading 0, the machine goes to state q_2, writes ␣, and moves the head to the right. In other words, $\delta(q_1,0) = (q_2,␣,\text{R})$. For clarity we use the shorthand 0→R in the transition from q_3 to q_4, to mean that the machine moves to the right when reading 0 in state q_3 but doesn't alter the tape, so $\delta(q_3,0) = (q_4,0,\text{R})$.

This machine begins by writing a blank symbol over the leftmost 0 on the tape so that it can find the left-hand end of the tape in stage 4. Whereas we would normally use a more suggestive symbol such as # for the left-hand end delimiter, we use a blank here to keep the tape alphabet, and hence the state diagram, small. Example 3.11 gives another method of finding the left-hand end of the tape.

Next we give a sample run of this machine on input 0000. The starting configuration is $q_1$0000. The sequence of configurations the machine enters appears as follows; read down the columns and left to right.

$q_1$0000	␣q_5x0x␣	␣x$q_5$xx␣
␣$q_2$000	q_5␣x0x␣	␣$q_5$xxx␣
␣x$q_3$00	␣q_2x0x␣	q_5␣xxx␣
␣x0$q_4$0	␣x$q_2$0x␣	␣$q_2$xxx␣
␣x0xq_3␣	␣xxq_3x␣	␣x$q_2$xx␣
␣x0q_5x␣	␣xxxq_3␣	␣xxq_2x␣
␣x$q_5$0x␣	␣xxq_5x␣	␣xxxq_2␣
		␣xxx␣q_{accept}

EXAMPLE 3.9

The following is a formal description of $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$, the Turing machine that we informally described (page 167) for deciding the language $B = \{w\#w \mid w \in \{0,1\}^*\}$.

- $Q = \{q_1, \ldots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0,1,\#\}$, and $\Gamma = \{0,1,\#,x,\sqcup\}$.
- We describe δ with a state diagram (see the following figure).
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}, respectively.

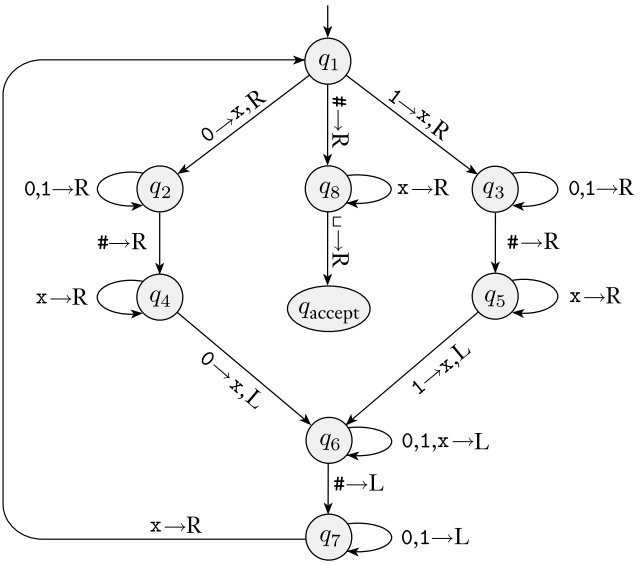

FIGURE 3.10
State diagram for Turing machine M_1

In Figure 3.10, which depicts the state diagram of TM M_1, you will find the label 0,1→R on the transition going from q_3 to itself. That label means that the machine stays in q_3 and moves to the right when it reads a 0 or a 1 in state q_3. It doesn't change the symbol on the tape.

Stage 1 is implemented by states q_1 through q_7, and stage 2 by the remaining states. To simplify the figure, we don't show the reject state or the transitions going to the reject state. Those transitions occur implicitly whenever a state lacks an outgoing transition for a particular symbol. Thus because in state q_5 no outgoing arrow with a # is present, if a # occurs under the head when the machine is in state q_5, it goes to state q_{reject}. For completeness, we say that the head moves right in each of these transitions to the reject state. ■

EXAMPLE 3.11

Here, a TM M_3 is doing some elementary arithmetic. It decides the language $C = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i \times j = k \text{ and } i, j, k \geq 1\}$.

M_3 = "On input string w:

1. Scan the input from left to right to determine whether it is a member of $\mathtt{a}^+\mathtt{b}^+\mathtt{c}^+$ and *reject* if it isn't.
2. Return the head to the left-hand end of the tape.
3. Cross off an a and scan to the right until a b occurs. Shuttle between the b's and the c's, crossing off one of each until all b's are gone. If all c's have been crossed off and some b's remain, *reject*.
4. Restore the crossed off b's and repeat stage 3 if there is another a to cross off. If all a's have been crossed off, determine whether all c's also have been crossed off. If yes, *accept*; otherwise, *reject*."

Let's examine the four stages of M_3 more closely. In stage 1, the machine operates like a finite automaton. No writing is necessary as the head moves from left to right, keeping track by using its states to determine whether the input is in the proper form.

Stage 2 looks equally simple but contains a subtlety. How can the TM find the left-hand end of the input tape? Finding the right-hand end of the input is easy because it is terminated with a blank symbol. But the left-hand end has no terminator initially. One technique that allows the machine to find the left-hand end of the tape is for it to mark the leftmost symbol in some way when the machine starts with its head on that symbol. Then the machine may scan left until it finds the mark when it wants to reset its head to the left-hand end. Example 3.7 illustrated this technique; a blank symbol marks the left-hand end.

A trickier method of finding the left-hand end of the tape takes advantage of the way that we defined the Turing machine model. Recall that if the machine tries to move its head beyond the left-hand end of the tape, it stays in the same place. We can use this feature to make a left-hand end detector. To detect whether the head is sitting on the left-hand end, the machine can write a special symbol over the current position while recording the symbol that it replaced in the control. Then it can attempt to move the head to the left. If it is still over the special symbol, the leftward move didn't succeed, and thus the head must have been at the left-hand end. If instead it is over a different symbol, some symbols remained to the left of that position on the tape. Before going farther, the machine must be sure to restore the changed symbol to the original.

Stages 3 and 4 have straightforward implementations and use several states each. ■

EXAMPLE 3.12

Here, a TM M_4 is solving what is called the *element distinctness problem*. It is given a list of strings over {0,1} separated by #s and its job is to accept if all the strings are different. The language is

$$E = \{\#x_1\#x_2\#\cdots\#x_l \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}.$$

Machine M_4 works by comparing x_1 with x_2 through x_l, then by comparing x_2 with x_3 through x_l, and so on. An informal description of the TM M_4 deciding this language follows.

M_4 = "On input w:

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only x_1 was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
5. Go to stage 3."

This machine illustrates the technique of marking tape symbols. In stage 2, the machine places a mark above a symbol, # in this case. In the actual implementation, the machine has two different symbols, # and $\dot{\#}$, in its tape alphabet. Saying that the machine places a mark above a # means that the machine writes the symbol $\dot{\#}$ at that location. Removing the mark means that the machine writes the symbol without the dot. In general, we may want to place marks over various symbols on the tape. To do so, we merely include versions of all these tape symbols with dots in the tape alphabet. ■

We conclude from the preceding examples that the described languages A, B, C, and E are decidable. All decidable languages are Turing-recognizable, so these languages are also Turing-recognizable. Demonstrating a language that is Turing-recognizable but undecidable is more difficult. We do so in Chapter 4.