Now M scans the list of edges. For each edge, M tests whether the two underlined nodes n_1 and n_2 are the ones appearing in that edge. If they are, M dots n_1, removes the underlines, and goes on from the beginning of stage 2. If they aren't, M checks the next edge on the list. If there are no more edges, $\{n_1, n_2\}$ is not an edge of G. Then M moves the underline on n_2 to the next dotted node and now calls this node n_2. It repeats the steps in this paragraph to check, as before, whether the new pair $\{n_1, n_2\}$ is an edge. If there are no more dotted nodes, n_1 is not attached to any dotted nodes. Then M sets the underlines so that n_1 is the next undotted node and n_2 is the first dotted node and repeats the steps in this paragraph. If there are no more undotted nodes, M has not been able to find any new nodes to dot, so it moves on to stage 4.

For stage 4, M scans the list of nodes to determine whether all are dotted. If they are, it enters the accept state; otherwise, it enters the reject state. This completes the description of TM M. ■

EXERCISES

3.1 This exercise concerns TM M_2, whose description and state diagram appear in Example 3.7. In each of the parts, give the sequence of configurations that M_2 enters when started on the indicated input string.

- **a.** `0`.
- A**b.** `00`.
- **c.** `000`.
- **d.** `000000`.

3.2 This exercise concerns TM M_1, whose description and state diagram appear in Example 3.9. In each of the parts, give the sequence of configurations that M_1 enters when started on the indicated input string.

- A**a.** `11`.
- **b.** `1#1`.
- **c.** `1##1`.
- **d.** `10#11`.
- **e.** `10#10`.

A**3.3** Modify the proof of Theorem 3.16 to obtain Corollary 3.19, showing that a language is decidable iff some nondeterministic Turing machine decides it. (You may assume the following theorem about trees. If every node in a tree has finitely many children and every branch of the tree has finitely many nodes, the tree itself has finitely many nodes.)

3.4 Give a formal definition of an enumerator. Consider it to be a type of two-tape Turing machine that uses its second tape as the printer. Include a definition of the enumerated language.

[A]**3.5** Examine the formal definition of a Turing machine to answer the following questions, and explain your reasoning.

a. Can a Turing machine ever write the blank symbol ␣ on its tape?

b. Can the tape alphabet Γ be the same as the input alphabet Σ?

c. Can a Turing machine's head *ever* be in the same location in two successive steps?

d. Can a Turing machine contain just a single state?

3.6 In Theorem 3.21, we showed that a language is Turing-recognizable iff some enumerator enumerates it. Why didn't we use the following simpler algorithm for the forward direction of the proof? As before, $s_1, s_2, \ldots$ is a list of all strings in Σ^*.

```
E = "Ignore the input.
    1. Repeat the following for $i = 1, 2, 3, \ldots$.
    2.     Run $M$ on $s_i$.
    3.     If it accepts, print out $s_i$."
```

3.7 Explain why the following is not a description of a legitimate Turing machine.

```
$M_{\text{bad}}$ = "On input $\langle p \rangle$, a polynomial over variables $x_1, \ldots, x_k$:
    1. Try all possible settings of $x_1, \ldots, x_k$ to integer values.
    2. Evaluate $p$ on all of these settings.
    3. If any of these settings evaluates to 0, accept; otherwise, reject."
```

3.8 Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet {0,1}.

[A]**a.** $\{w \mid w$ contains an equal number of 0s and 1s$\}$

b. $\{w \mid w$ contains twice as many 0s as 1s$\}$

c. $\{w \mid w$ does not contain twice as many 0s as 1s$\}$

PROBLEMS

3.9 Let a k-PDA be a pushdown automaton that has k stacks. Thus a 0-PDA is an NFA and a 1-PDA is a conventional PDA. You already know that 1-PDAs are more powerful (recognize a larger class of languages) than 0-PDAs.

a. Show that 2-PDAs are more powerful than 1-PDAs.

b. Show that 3-PDAs are not more powerful than 2-PDAs.
(Hint: Simulate a Turing machine tape with two stacks.)

[A]**3.10** Say that a ***write-once Turing machine*** is a single-tape TM that can alter each tape square at most once (including the input portion of the tape). Show that this variant Turing machine model is equivalent to the ordinary Turing machine model. (Hint: As a first step, consider the case whereby the Turing machine may alter each tape square at most twice. Use lots of tape.)

3.11 A ***Turing machine with doubly infinite tape*** is similar to an ordinary Turing machine, but its tape is infinite to the left as well as to the right. The tape is initially filled with blanks except for the portion that contains the input. Computation is defined as usual except that the head never encounters an end to the tape as it moves leftward. Show that this type of Turing machine recognizes the class of Turing-recognizable languages.

3.12 A ***Turing machine with left reset*** is similar to an ordinary Turing machine, but the transition function has the form

$$\delta\colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{R}, \mathrm{RESET}\}.$$

If $\delta(q, a) = (r, b, \mathrm{RESET})$, when the machine is in state q reading an a, the machine's head jumps to the left-hand end of the tape after it writes b on the tape and enters state r. Note that these machines do not have the usual ability to move the head one symbol left. Show that Turing machines with left reset recognize the class of Turing-recognizable languages.

3.13 A ***Turing machine with stay put instead of left*** is similar to an ordinary Turing machine, but the transition function has the form

$$\delta\colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{R}, \mathrm{S}\}.$$

At each point, the machine can move its head right or let it stay in the same position. Show that this Turing machine variant is *not* equivalent to the usual version. What class of languages do these machines recognize?

3.14 A ***queue automaton*** is like a push-down automaton except that the stack is replaced by a queue. A ***queue*** is a tape allowing symbols to be written only on the left-hand end and read only at the right-hand end. Each write operation (we'll call it a *push*) adds a symbol to the left-hand end of the queue and each read operation (we'll call it a *pull*) reads and removes a symbol at the right-hand end. As with a PDA, the input is placed on a separate read-only input tape, and the head on the input tape can move only from left to right. The input tape contains a cell with a blank symbol following the input, so that the end of the input can be detected. A queue automaton accepts its input by entering a special accept state at any time. Show that a language can be recognized by a deterministic queue automaton iff the language is Turing-recognizable.

3.15 Show that the collection of decidable languages is closed under the operation of

- $^{\mathrm{A}}$**a.** union.
- **b.** concatenation.
- **c.** star.
- **d.** complementation.
- **e.** intersection.

3.16 Show that the collection of Turing-recognizable languages is closed under the operation of

- $^{\mathrm{A}}$**a.** union.
- **b.** concatenation.
- **c.** star.
- **d.** intersection.
- **e.** homomorphism.

***3.17** Let $B = \{\langle M_1\rangle, \langle M_2\rangle, \ldots\}$ be a Turing-recognizable language consisting of TM descriptions. Show that there is a decidable language C consisting of TM descriptions such that every machine described in B has an equivalent machine in C and vice versa.

***3.18** Show that a language is decidable iff some enumerator enumerates the language in the standard string order.

***3.19** Show that every infinite Turing-recognizable language has an infinite decidable subset.

***3.20** Show that single-tape TMs that cannot write on the portion of the tape containing the input string recognize only regular languages.

3.21 Let $c_1x^n + c_2x^{n-1} + \cdots + c_nx + c_{n+1}$ be a polynomial with a root at $x = x_0$. Let $c_{\max}$ be the largest absolute value of a c_i. Show that

$$|x_0| < (n+1)\frac{c_{\max}}{|c_1|}.$$

$^{\text{A}}$3.22 Let A be the language containing only the single string s, where

$$s = \begin{cases} \texttt{0} & \text{if life never will be found on Mars.} \\ \texttt{1} & \text{if life will be found on Mars someday.} \end{cases}$$

Is A decidable? Why or why not? For the purposes of this problem, assume that the question of whether life will be found on Mars has an unambiguous YES or NO answer.

SELECTED SOLUTIONS

3.1 **(b)** $q_1\texttt{00}, \sqcup q_2\texttt{0}, \sqcup\texttt{x}q_3\sqcup, \sqcup q_5\texttt{x}\sqcup, q_5\sqcup\texttt{x}\sqcup, \sqcup q_2\texttt{x}\sqcup, \sqcup\texttt{x}q_2\sqcup, \sqcup\texttt{x}\sqcup q_{\text{accept}}$.

3.2 **(a)** $q_1\texttt{11}, \texttt{x}q_3\texttt{1}, \texttt{x1}q_3\sqcup, \texttt{x1}\sqcup q_{\text{reject}}$.

3.3 We prove both directions of the iff. First, if a language L is decidable, it can be decided by a deterministic Turing machine, and that is automatically a nondeterministic Turing machine.

Second, if a language L is decided by a nondeterministic TM N, we modify the deterministic TM D that was given in the proof of Theorem 3.16 as follows.
Move stage 4 to be stage 5.
Add new stage 4: *Reject* if all branches of N's nondeterminism have rejected.

We argue that this new TM D' is a decider for L. If N accepts its input, D' will eventually find an accepting branch and accept, too. If N rejects its input, all of its branches halt and reject because it is a decider. Hence each of the branches has finitely many nodes, where each node represents one step of N's computation along that branch. Therefore, N's entire computation tree on this input is finite, by virtue of the theorem about trees given in the statement of the exercise. Consequently, D' will halt and reject when this entire tree has been explored.

3.5 **(a)** Yes. The tape alphabet Γ contains $\sqcup$. A Turing machine can write any characters in Γ on its tape.

(b) No. Σ never contains $\sqcup$, but Γ always contains $\sqcup$. So they cannot be equal.

(c) Yes. If the Turing machine attempts to move its head off the left-hand end of the tape, it remains on the same tape cell.

(d) No. Any Turing machine must contain two distinct states: q_{accept} and q_{reject}. So, a Turing machine contains at least two states.

3.8 **(a)** "On input string w:

```
1. Scan the tape and mark the first 0 that has not been marked. If
   no unmarked 0 is found, go to stage 4. Otherwise, move the
   head back to the front of the tape.
2. Scan the tape and mark the first 1 that has not been marked. If
   no unmarked 1 is found, reject.
3. Move the head back to the front of the tape and go to stage 1.
4. Move the head back to the front of the tape. Scan the tape to see
   if any unmarked 1s remain. If none are found, accept; otherwise,
   reject."
```

3.10 We first simulate an ordinary Turing machine by a write-twice Turing machine. The write-twice machine simulates a single step of the original machine by copying the entire tape over to a fresh portion of the tape to the right-hand side of the currently used portion. The copying procedure operates character by character, marking a character as it is copied. This procedure alters each tape square twice: once to write the character for the first time, and again to mark that it has been copied. The position of the original Turing machine's tape head is marked on the tape. When copying the cells at or adjacent to the marked position, the tape content is updated according to the rules of the original Turing machine.

To carry out the simulation with a write-once machine, operate as before, except that each cell of the previous tape is now represented by two cells. The first of these contains the original machine's tape symbol and the second is for the mark used in the copying procedure. The input is not presented to the machine in the format with two cells per symbol, so the very first time the tape is copied, the copying marks are put directly over the input symbols.

3.15 **(a)** For any two decidable languages L_1 and L_2, let M_1 and M_2 be the TMs that decide them. We construct a TM M' that decides the union of L_1 and L_2:

"On input w:

```
1. Run $M_1$ on $w$. If it accepts, accept.
2. Run $M_2$ on $w$. If it accepts, accept. Otherwise, reject."
```

M' accepts w if either M_1 or M_2 accepts it. If both reject, M' rejects.

3.16 **(a)** For any two Turing-recognizable languages L_1 and L_2, let M_1 and M_2 be the TMs that recognize them. We construct a TM M' that recognizes the union of L_1 and L_2:

"On input w:

```
1. Run $M_1$ and $M_2$ alternately on $w$ step by step. If either accepts,
   accept. If both halt and reject, reject."
```

If either M_1 or M_2 accepts w, M' accepts w because the accepting TM arrives to its accepting state after a finite number of steps. Note that if both M_1 and M_2 reject and either of them does so by looping, then M' will loop.

3.22 The language A is one of the two languages $\{0\}$ or $\{1\}$. In either case, the language is finite and hence decidable. If you aren't able to determine which of these two languages is A, you won't be able to describe the decider for A. However, you can give two Turing machines, one of which is A's decider.