4.2 THE HALTING PROBLEM

In this section we prove one of the most philosophically important theorems of the theory of computation: There is a specific problem that is algorithmically unsolvable. Computers appear to be so powerful that you may believe that all problems will eventually yield to them. The theorem presented here demonstrates that computers are limited in a fundamental way.

What sort of problems are unsolvable by computer? Are they esoteric, dwelling only in the minds of theoreticians? No! Even some ordinary problems that people want to solve turn out to be computationally unsolvable.

In one type of unsolvable problem, you are given a computer program and a precise specification of what that program is supposed to do (e.g., sort a list of numbers). You need to verify that the program performs as specified (i.e., that it is correct). Because both the program and the specification are mathematically precise objects, you hope to automate the process of verification by feeding these objects into a suitably programmed computer. However, you will be disappointed. The general problem of software verification is not solvable by computer.

In this section and Chapter 5 you will encounter several computationally unsolvable problems. Our objectives are to help you develop a feel for the types of problems that are unsolvable and to learn techniques for proving unsolvability.

Now we turn to our first theorem that establishes the undecidability of a specific language: the problem of determining whether a Turing machine accepts a given input string. We call it A_{TM} by analogy with A_{DFA} and A_{CFG}. But, whereas

A_{DFA} and A_{CFG} were decidable, A_{TM} is not. Let

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}.$$

THEOREM 4.11

A_{TM} is undecidable.

Before we get to the proof, let's first observe that A_{TM} is Turing-recognizable. Thus this theorem shows that recognizers *are* more powerful than deciders. Requiring a TM to halt on all inputs restricts the kinds of languages that it can recognize. The following Turing machine U recognizes A_{TM}.

```
U = "On input ⟨M, w⟩, where M is a TM and w is a string:
    1. Simulate M on input w.
    2. If M ever enters its accept state, accept; if M ever enters its
       reject state, reject."
```

Note that this machine loops on input $\langle M, w\rangle$ if M loops on w, which is why this machine does not decide A_{TM}. If the algorithm had some way to determine that M was not halting on w, it could *reject*. Hence A_{TM} is sometimes called the ***halting problem***. As we demonstrate, an algorithm has no way to make this determination.

The Turing machine U is interesting in its own right. It is an example of the *universal Turing machine* first proposed by Turing. This machine is called universal because it is capable of simulating any other Turing machine from the description of that machine. The universal Turing machine played an important early role in stimulating the development of stored-program computers.

THE DIAGONALIZATION METHOD

The proof of the undecidability of the halting problem uses a technique called *diagonalization*, discovered by mathematician Georg Cantor in 1873. Cantor was concerned with the problem of measuring the sizes of infinite sets. If we have two infinite sets, how can we tell whether one is larger than the other or whether they are of the same size? For finite sets, of course, answering these questions is easy. We simply count the elements in a finite set, and the resulting number is its size. But, if we try to count the elements of an infinite set, we will never finish! So we can't use the counting method to determine the relative sizes of infinite sets.

For example, take the set of even integers and the set of all strings over $\{0,1\}$. Both sets are infinite and thus larger than any finite set, but is one of the two larger than the other? How can we compare their relative size?

Cantor proposed a rather nice solution to this problem. He observed that two finite sets have the same size if the elements of one set can be paired with the elements of the other set. This method compares the sizes without resorting to counting. We can extend this idea to infinite sets. Let's see what it means more precisely.

DEFINITION 4.12

Assume that we have sets A and B and a function f from A to B. Say that f is ***one-to-one*** if it never maps two different elements to the same place—that is, if $f(a) \neq f(b)$ whenever $a \neq b$. Say that f is ***onto*** if it hits every element of B—that is, if for every $b \in B$ there is an $a \in A$ such that $f(a) = b$. Say that A and B are the ***same size*** if there is a one-to-one, onto function $f\colon A \longrightarrow B$. A function that is both one-to-one and onto is called a ***correspondence***. In a correspondence every element of A maps to a unique element of B and each element of B has a unique element of A mapping to it. A correspondence is simply a way of pairing the elements of A with the elements of B.

EXAMPLE 4.13

Let $\mathcal{N}$ be the set of natural numbers $\{1, 2, 3, \ldots\}$ and let $\mathcal{E}$ be the set of even natural numbers $\{2, 4, 6, \ldots\}$. Using Cantor's definition of size we can see that $\mathcal{N}$ and $\mathcal{E}$ have the same size. The correspondence f mapping $\mathcal{N}$ to $\mathcal{E}$ is simply $f(n) = 2n$. We can visualize f more easily with the help of a table.

n	$f(n)$
1	2
2	4
3	6
$\vdots$	$\vdots$

Of course, this example seems bizarre. Intuitively, $\mathcal{E}$ seems smaller than $\mathcal{N}$ because $\mathcal{E}$ is a proper subset of $\mathcal{N}$. But pairing each member of $\mathcal{N}$ with its own member of $\mathcal{E}$ is possible, so we declare these two sets to be the same size. ■

DEFINITION 4.14

A set A is ***countable*** if either it is finite or it has the same size as $\mathcal{N}$.

EXAMPLE 4.15

Now we turn to an even stranger example. If we let $\mathcal{Q} = \{\frac{m}{n} \mid m, n \in \mathcal{N}\}$ be the set of positive rational numbers, $\mathcal{Q}$ seems to be much larger than $\mathcal{N}$. Yet these two sets are the same size according to our definition. We give a correspondence with $\mathcal{N}$ to show that $\mathcal{Q}$ is countable. One easy way to do so is to list all the elements of $\mathcal{Q}$. Then we pair the first element on the list with the number 1 from $\mathcal{N}$, the second element on the list with the number 2 from $\mathcal{N}$, and so on. We must ensure that every member of $\mathcal{Q}$ appears only once on the list.

To get this list we make an infinite matrix containing all the positive rational numbers, as shown in Figure 4.16. The ith row contains all numbers with numerator i and the jth column has all numbers with denominator j. So the number $\frac{i}{j}$ occurs in the ith row and jth column.

Now we turn this matrix into a list. One (bad) way to attempt it would be to begin the list with all the elements in the first row. That isn't a good approach because the first row is infinite, so the list would never get to the second row. Instead we list the elements on the diagonals, starting from the corner, which are superimposed on the diagram. The first diagonal contains the single element $\frac{1}{1}$, and the second diagonal contains the two elements $\frac{2}{1}$ and $\frac{1}{2}$. So the first three elements on the list are $\frac{1}{1}$, $\frac{2}{1}$, and $\frac{1}{2}$. In the third diagonal a complication arises. It contains $\frac{3}{1}$, $\frac{2}{2}$, and $\frac{1}{3}$. If we simply added these to the list, we would repeat $\frac{1}{1} = \frac{2}{2}$. We avoid doing so by skipping an element when it would cause a repetition. So we add only the two new elements $\frac{3}{1}$ and $\frac{1}{3}$. Continuing in this way we obtain a list of all the elements of $\mathcal{Q}$.

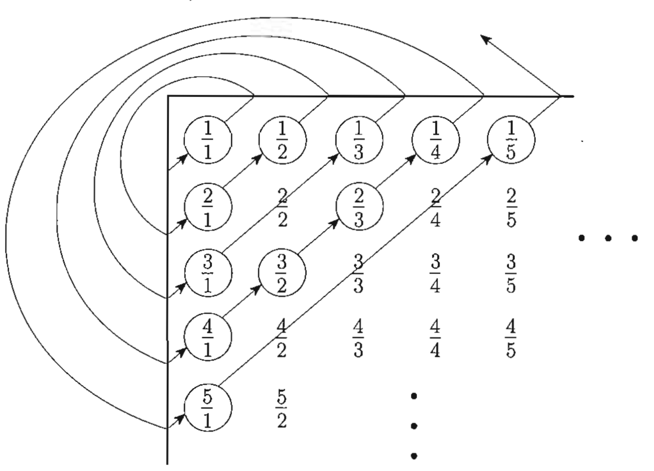

FIGURE **4.16**
A correspondence of $\mathcal{N}$ and $\mathcal{Q}$

After seeing the correspondence of $\mathcal{N}$ and $\mathcal{Q}$, you might think that any two infinite sets can be shown to have the same size. After all, you need only demonstrate a correspondence, and this example shows that surprising correspondences do exist. However, for some infinite sets no correspondence with $\mathcal{N}$ exists. These sets are simply too big. Such sets are called ***uncountable***.

The set of real numbers is an example of an uncountable set. A ***real number*** is one that has a decimal representation. The numbers π = `3.1415926...` and $\sqrt{2}$ = `1.4142135...` are examples of real numbers. Let $\mathcal{R}$ be the set of real numbers. Cantor proved that $\mathcal{R}$ is uncountable. In doing so he introduced the diagonalization method.

THEOREM **4.17**

$\mathcal{R}$ is uncountable.

PROOF In order to show that $\mathcal{R}$ is uncountable, we show that no correspondence exists between $\mathcal{N}$ and $\mathcal{R}$. The proof is by contradiction. Suppose that a correspondence f existed between $\mathcal{N}$ and $\mathcal{R}$. Our job is to show that f fails to work as it should. For it to be a correspondence, f must pair all the members of $\mathcal{N}$ with all the members of $\mathcal{R}$. But we will find an x in $\mathcal{R}$ that is not paired with anything in $\mathcal{N}$, which will be our contradiction.

The way we find this x is by actually constructing it. We choose each digit of x to make x different from one of the real numbers that is paired with an element of $\mathcal{N}$. In the end we are sure that x is different from any real number that is paired.

We can illustrate this idea by giving an example. Suppose that the correspondence f exists. Let f(`1`) = `3.14159...`, f(`2`) = `55.55555...`, f(`3`) = `...`, and so on, just to make up some values for f. Then f pairs the number `1` with `3.14159...`, the number `2` with `55.55555...`, and so on. The following table shows a few values of a hypothetical correspondence f between $\mathcal{N}$ and $\mathcal{R}$.

n	$f(n)$
1	3.14159...
2	55.55555...
3	0.12345...
4	0.50000...
⋮	⋮

We construct the desired x by giving its decimal representation. It is a number between 0 and 1, so all its significant digits are fractional digits following the decimal point. Our objective is to ensure that $x \neq f(n)$ for any n. To ensure that $x \neq f(1)$ we let the first digit of x be anything different from the first fractional digit `1` of f(`1`) = `3.`$\underline{\texttt{1}}$`4159...`. Arbitrarily, we let it be `4`. To ensure that $x \neq f(2)$ we let the second digit of x be anything different from the second fractional digit `5` of f(`2`) = `55.5`$\underline{\texttt{5}}$`5555...`. Arbitrarily, we let it be `6`. The third fractional digit of f(`3`) = `0.12`$\underline{\texttt{3}}$`45...` is `3`, so we let x be anything different—say, `4`. Continuing in this way down the diagonal of the table for f, we obtain all the digits of x, as shown in the following table. We know that x is not $f(n)$ for any n because it differs from $f(n)$ in the nth fractional digit. (A slight problem arises because certain numbers, such as `0.1999...` and `0.2000...`, are equal even though their decimal representations are different. We avoid this problem by never selecting the digits `0` or `9` when we construct x.)

n	$f(n)$
1	$3.\underline{1}4159\ldots$
2	$55.5\underline{5}555\ldots$
3	$0.12\underline{3}45\ldots$
4	$0.500\underline{0}0\ldots$
$\vdots$	$\vdots$

$x = 0.4641\ldots$

The preceding theorem has an important application to the theory of computation. It shows that some languages are not decidable or even Turing-recognizable, for the reason that there are uncountably many languages yet only countably many Turing machines. Because each Turing machine can recognize a single language and there are more languages than Turing machines, some languages are not recognized by any Turing machine. Such languages are not Turing-recognizable, as we state in the following corollary.

COROLLARY 4.18

Some languages are not Turing-recognizable.

PROOF To show that the set of all Turing machines is countable we first observe that the set of all strings Σ^* is countable, for any alphabet Σ. With only finitely many strings of each length, we may form a list of Σ^* by writing down all strings of length 0, length 1, length 2, and so on.

The set of all Turing machines is countable because each Turing machine M has an encoding into a string $\langle M\rangle$. If we simply omit those strings that are not legal encodings of Turing machines, we can obtain a list of all Turing machines.

To show that the set of all languages is uncountable we first observe that the set of all infinite binary sequences is uncountable. An ***infinite binary sequence*** is an unending sequence of 0s and 1s. Let $\mathcal{B}$ be the set of all infinite binary sequences. We can show that $\mathcal{B}$ is uncountable by using a proof by diagonalization similar to the one we used in Theorem 4.17 to show that $\mathcal{R}$ is uncountable.

Let $\mathcal{L}$ be the set of all languages over alphabet Σ. We show that $\mathcal{L}$ is uncountable by giving a correspondence with $\mathcal{B}$, thus showing that the two sets are the same size. Let $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$. Each language $A \in \mathcal{L}$ has a unique sequence in $\mathcal{B}$. The ith bit of that sequence is a 1 if $s_i \in A$ and is a 0 if $s_i \notin A$, which is called the ***characteristic sequence*** of A. For example, if A were the language of all strings starting with a 0 over the alphabet $\{0,1\}$, its characteristic sequence χ_A would be

$$\begin{array}{rcccccccccccl}
\Sigma^* = \{ & \varepsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000, & 001, & \cdots & \} ; \\
A = \{ & & 0, & & 00, & 01, & & & 000, & 001, & \cdots & \} ; \\
\chi_A = & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \cdots & .
\end{array}$$

The function $f\colon \mathcal{L}\longrightarrow\mathcal{B}$, where $f(A)$ equals the characteristic sequence of A, is one-to-one and onto and hence a correspondence. Therefore, as $\mathcal{B}$ is uncountable, $\mathcal{L}$ is uncountable as well.

Thus we have shown that the set of all languages cannot be put into a correspondence with the set of all Turing machines. We conclude that some languages are not recognized by any Turing machine.

THE HALTING PROBLEM IS UNDECIDABLE

Now we are ready to prove Theorem 4.11, the undecidability of the language

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}.$$

PROOF We assume that A_{TM} is decidable and obtain a contradiction. Suppose that H is a decider for A_{TM}. On input $\langle M, w\rangle$, where M is a TM and w is a string, H halts and accepts if M accepts w. Furthermore, H halts and rejects if M fails to accept w. In other words, we assume that H is a TM, where

$$H\big(\langle M, w\rangle\big) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w. \end{cases}$$

Now we construct a new Turing machine D with H as a subroutine. This new TM calls H to determine what M does when the input to M is its own description $\langle M\rangle$. Once D has determined this information, it does the opposite. That is, it rejects if M accepts and accepts if M does not accept. The following is a description of D.

```
D = "On input ⟨M⟩, where M is a TM:
    1. Run H on input ⟨M, ⟨M⟩⟩.
    2. Output the opposite of what H outputs; that is, if H accepts,
       reject and if H rejects, accept."
```

Don't be confused by the idea of running a machine on its own description! That is similar to running a program with itself as input, something that does occasionally occur in practice. For example, a compiler is a program that translates other programs. A compiler for the language Pascal may itself be written in Pascal, so running that program on itself would make sense. In summary,

$$D\big(\langle M\rangle\big) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M\rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M\rangle. \end{cases}$$

What happens when we run D with its own description $\langle D\rangle$ as input? In that case we get

$$D\big(\langle D\rangle\big) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D\rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D\rangle. \end{cases}$$

No matter what D does, it is forced to do the opposite, which is obviously a contradiction. Thus neither TM D nor TM H can exist.

Let's review the steps of this proof. Assume that a TM H decides A_{TM}. Then use H to build a TM D that when given input $\langle M \rangle$ accepts exactly when M does not accept input $\langle M \rangle$. Finally, run D on itself. The machines take the following actions, with the last line being the contradiction.

- H accepts $\langle M, w \rangle$ exactly when M accepts w.
- D rejects $\langle M \rangle$ exactly when M accepts $\langle M \rangle$.
- D rejects $\langle D \rangle$ exactly when D accepts $\langle D \rangle$.

Where is the diagonalization in the proof of Theorem 4.11? It becomes apparent when you examine tables of behavior for TMs H and D. In these tables we list all TMs down the rows, $M_1, M_2, \ldots$ and all their descriptions across the columns, $\langle M_1 \rangle, \langle M_2 \rangle, \ldots$ The entries tell whether the machine in a given row accepts the input in a given column. The entry is *accept* if the machine accepts the input but is blank if it rejects or loops on that input. We made up the entries in the following figure to illustrate the idea.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*		*accept*		
M_2	*accept*	*accept*	*accept*	*accept*	
M_3					$\cdots$
M_4	*accept*	*accept*			
$\vdots$		$\vdots$			

FIGURE 4.19
Entry i, j is *accept* if M_i accepts $\langle M_j \rangle$

In the following figure the entries are the results of running H on inputs corresponding to Figure 4.18. So if M_3 does not accept input $\langle M_2 \rangle$, the entry for row M_3 and column $\langle M_2 \rangle$ is *reject* because H rejects input $\langle M_3, \langle M_2 \rangle \rangle$.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$		$\vdots$			

FIGURE 4.20
Entry i, j is the value of H on input $\langle M_i, \langle M_j \rangle \rangle$

In the following figure, we added D to Figure 4.19. By our assumption, H is a TM and so is D. Therefore it must occur on the list $M_1, M_2, \ldots$ of all TMs. Note that D computes the opposite of the diagonal entries. The contradiction occurs at the point of the question mark where the entry must be the opposite of itself.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
$\vdots$		$\vdots$			$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		?	
$\vdots$		$\vdots$					$\ddots$

FIGURE 4.21
If D is in the figure, a contradiction occurs at "?"

A TURING-UNRECOGNIZABLE LANGUAGE

In the preceding section we demonstrated a language—namely, A_{TM}—that is undecidable. Now we demonstrate a language that isn't even Turing-recognizable. Note that A_{TM} will not suffice for this purpose because we showed that A_{TM} is Turing-recognizable (page 174). The following theorem shows that, if both a language and its complement are Turing-recognizable, the language is decidable. Hence, for any undecidable language, either it or its complement is not Turing-recognizable. Recall that the complement of a language is the language consisting of all strings that are not in the language. We say that a language is ***co-Turing-recognizable*** if it is the complement of a Turing-recognizable language.

THEOREM 4.22

A language is decidable iff it is Turing-recognizable and co-Turing-recognizable.

In other words, a language is decidable exactly when both it and its complement are Turing-recognizable.

PROOF We have two directions to prove. First, if A is decidable, we can easily see that both A and its complement $\overline{A}$ are Turing-recognizable. Any decidable language is Turing-recognizable, and the complement of a decidable language also is decidable.

For the other direction, if both A and $\overline{A}$ are Turing-recognizable, we let M_1 be the recognizer for A and M_2 be the recognizer for $\overline{A}$. The following Turing machine M is a decider for A.

```
M = "On input w:
    1. Run both $M_1$ and $M_2$ on input $w$ in parallel.
    2. If $M_1$ accepts, accept; if $M_2$ accepts, reject."
```

Running the two machines in parallel means that M has two tapes, one for simulating M_1 and the other for simulating M_2. In this case M takes turns simulating one step of each machine, which continues until one of them accepts.

Now we show that M decides A. Every string w is either in A or $\overline{A}$. Therefore either M_1 or M_2 must accept w. Because M halts whenever M_1 or M_2 accepts, M always halts and so it is a decider. Furthermore, it accepts all strings in A and rejects all strings not in A. So M is a decider for A, and thus A is decidable.

COROLLARY 4.23

$\overline{A_{\mathsf{TM}}}$ is not Turing-recognizable.

PROOF We know that A_{TM} is Turing-recognizable. If $\overline{A_{\mathsf{TM}}}$ also were Turing-recognizable, A_{TM} would be decidable. Theorem 4.11 tells us that A_{TM} is not decidable, so $\overline{A_{\mathsf{TM}}}$ must not be Turing-recognizable.