1.63. Suppose R is a relation on a nonempty set A.

a. Define $R^s = R \cup \{(x, y) \mid yRx\}$. Show that R^s is symmetric and is the smallest symmetric relation on A containing R (i.e., for any symmetric relation R_1 with $R \subseteq R_1$, $R^s \subseteq R_1$).

b. Define R^t to be the intersection of all transitive relations on A containing R. Show that R^t is transitive and is the smallest transitive relation on A containing R.

c. Let $R^u = R \cup \{(x, y) \mid \exists z(xRz \text{ and } zRy)\}$. Is R^u equal to the set R^t in part (b)? Either prove that it is, or give an example in which it is not.

The relations R^s and R^t are called the symmetric closure and transitive closure of R, respectively.

1.64. Let R be the equivalence relation in Exercise 1.33a. Assuming that S is finite, find a function $f : 2^S \to \mathcal{N}$ so that for any $x, y \in 2^S$, xRy if and only if $f(x) = f(y)$.

1.65. Let n be a positive integer. Find a function $f : \mathcal{N} \to \mathcal{N}$ so that for any $x, y \in \mathcal{N}$, $x \equiv_n y$ if and only if $f(x) = f(y)$.

1.66. Let A be any set, and let R be any equivalence relation on A. Find a set B and a function $f : A \to B$ so that for any $x, y \in A$, xRy if and only if $f(x) = f(y)$.

1.67. Suppose R is an equivalence relation on a set A. A subset $S \subseteq A$ is *pairwise inequivalent* if no two distinct elements of S are equivalent. S is a *maximal* pairwise inequivalent set if S is pairwise inequivalent and every element of A is equivalent to some element of S. Show that a set S is a maximal pairwise inequivalent set if and only if it contains exactly one element of each equivalence class.

1.68. Suppose R_1 and R_2 are equivalence relations on a set A. As discussed in Section 1.4, the equivalence classes of R_1 and R_2 form partitions P_1 and P_2, respectively, of A. Show that $R_1 \subseteq R_2$ if and only if the partition P_1 is *finer* than P_2 (i.e., every subset in the partition P_2 is the union of one or more subsets in the partition P_1).

1.69. Suppose Σ is an alphabet. It is obviously possible for two distinct strings x and y over Σ to satisfy the condition $xy = yx$, since this condition is always satisfied if $y = \Lambda$. Is it possible under the additional restriction that x and y are both nonnull? Either prove that this cannot happen, or describe precisely the circumstances under which it can.

1.70. Show that there is no language L so that $\{aa, bb\}^*\{ab, ba\}^* = L^*$.

1.71. Consider the language $L = \{x \in \{0, 1\}^* \mid x = yy \text{ for some string } y\}$. We know that $L = L\{\Lambda\} = \{\Lambda\}L$ (because any language L has this property). Is there any other way to express L as the concatenation of two languages? Prove your answer.

CHAPTER

2

Mathematical Induction and Recursive Definitions

2.1 | PROOFS

A *proof* of a statement is essentially just a convincing argument that the statement is true. Ideally, however, a proof not only convinces but explains why the statement is true, and also how it relates to other statements and how it fits into the overall theory. A typical step in a proof is to derive some statement from (1) assumptions or hypotheses, (2) statements that have already been derived, and (3) other generally accepted facts, using general principles of logical reasoning. In a very careful, detailed proof, we might allow no "generally accepted facts" other than certain axioms that we specify initially, and we might restrict ourselves to certain specific rules of logical inference, by which each step must be justified. Being this careful, however, may not be feasible or worthwhile. We may take shortcuts ("It is obvious that ..." or "It is easy to show that ...") and concentrate on the main steps in the proof, assuming that a conscientious or curious reader could fill in the low-level details.

Usually what we are trying to prove involves a statement of the form $p \to q$. A *direct* proof assumes that the statement p is true and uses this to show q is true.

The Product of Two Odd Integers Is Odd — EXAMPLE 2.1

To prove: For any integers a and b, if a and b are odd, then ab is odd.

■ Proof

We start by saying more precisely what our assumption means. An integer n is odd if there exists an integer x so that $n = 2x + 1$. Now let a and b be any odd integers. Then according to this definition, there is an integer x so that $a = 2x + 1$, and there is an integer y so that

$b = 2y + 1$. We wish to show that there is an integer z so that $ab = 2z + 1$. Let us therefore calculate ab:

$$\begin{aligned} ab &= (2x + 1)(2y + 1) \\ &= 4xy + 2x + 2y + 1 \\ &= 2(2xy + x + y) + 1 \end{aligned}$$

Since we have shown that there is a z, namely, $2xy + x + y$, so that $ab = 2z + 1$, the proof is complete.

This is an example of a *constructive* proof. We proved the statement "There exists z such that . . ." by constructing a specific value for z that works. A nonconstructive proof shows that such a z must exist without providing any information about its value. Such a proof would not explain, it would only convince. Although in some situations this is the best we can do, people normally prefer a constructive proof if one is possible. In some cases, the method of construction is interesting in its own right. In these cases, the proof is even more valuable because it provides an algorithm as well as an explanation.

Since the statement we proved in Example 2.1 is the quantified statement "For any integers a and $b, \ldots,$" it is important to understand that it is *not* sufficient to give an example of a and b for which the statement is true. If we say "Let $a = 45$ and $b = 11$; then $a = 2(22) + 1$ and $b = 2(5) + 2$; therefore, $ab = (2 * 22 + 1)(2 * 5 + 1) = \ldots = 2 * 247 + 1$," we have proved nothing except that $45 * 11$ is odd. Finding a value of x so that the statement $P(x)$ is true is a proof of the statement "There exists x such that $P(x)$." Finding a value of x for which $P(x)$ is false *disproves* the statement "For every x, $P(x)$" (or, if you prefer, proves the statement "It is not the case that for every x, $P(x)$"); this is called a *proof by counterexample*. To prove "For every x, $P(x)$," however, requires that we give an argument in which there are no restrictions on x. (Let us return briefly to the example with 45 and 11. It is not totally unreasonable to claim that the argument beginning "Let $a = 45$ and $b = 11$" is a proof of the quantified statement—after all, the algebraic steps involved are the same as the ones we presented in our official proof. The crucial point, however, is that there is nothing special about 45 and 11. Someone who offers this as a proof should at least point out that the same argument would work in general. For an argument this simple, such an observation may be convincing; even more convincing is an argument involving a and b like the one we gave originally.)

The alternative to a direct proof is an *indirect* proof, and the simplest form of indirect proof is a *proof by contrapositive*, using the logical equivalence of $p \rightarrow q$ and $\neg q \rightarrow \neg p$.

EXAMPLE 2.2 A Proof by Contrapositive

To prove: For any positive integers i, j, and n, if $i * j = n$, then either $i \leq \sqrt{n}$ or $j \leq \sqrt{n}$.

■ Proof

The statement we wish to prove is of the general form "For every x, if $p(x)$, then $q(x)$." For each x, the statement "If $p(x)$ then $q(x)$" is logically equivalent to "If not $q(x)$ then not $p(x)$," and therefore (by a general principle of logical reasoning) the statement we want to prove is equivalent to this: For any positive integers i, j, and n, if it is not the case that $i \leq \sqrt{n}$ or $j \leq \sqrt{n}$, then $i * j \neq n$.

If it is not true that $i \leq \sqrt{n}$ or $j \leq \sqrt{n}$, then $i > \sqrt{n}$ and $j > \sqrt{n}$. A generally accepted fact from mathematics is that if a and b are numbers with $a > b$, and c is a number > 0, then $ac > bc$. Applying this to the inequality $i > \sqrt{n}$ with $c = j$, we obtain $i * j > \sqrt{n} * j$. Since $n > 0$, we know that $\sqrt{n} > 0$, and we may apply the same fact again to the inequality $j > \sqrt{n}$, this time letting $c = \sqrt{n}$, to obtain $j\sqrt{n} > \sqrt{n}\sqrt{n} = n$. We now have $i * j > j\sqrt{n} > n$, and it follows that $i * j \neq n$.

The second paragraph in this proof illustrates the fact that a complete proof, with no details left out, is usually not feasible. Even though the statement we are proving here is relatively simple, and our proof includes more detail than might normally be included, there is still a lot left out. Here are some of the details that were ignored:

1. $\neg(p \vee q)$ is logically equivalent to $\neg p \wedge \neg q$. Therefore, if it is not true that $i \leq \sqrt{n}$ or $j \leq \sqrt{n}$, then $i \nleq \sqrt{n}$ and $j \nleq \sqrt{n}$.
2. For any two real numbers a and b, exactly one of the conditions $a < b$, $a > b$, and $a = b$ holds. (This is a generally accepted fact from mathematics.) Therefore, if $i \nleq \sqrt{n}$, then $i > \sqrt{n}$, and similarly for j.
3. For any two real numbers a and b, $a * b = b * a$. Therefore, $\sqrt{n} * j = j\sqrt{n}$.
4. The $>$ relation on the set of real numbers is transitive. Therefore, from the fact that $i * j > j\sqrt{n}$ and $j\sqrt{n} > n$ it follows that $i * j > n$.

Even if we include all these details, we have not stated explicitly the rules of inference we have used to arrive at the final conclusion, and we have used a number of facts about real numbers that could themselves be proved from more fundamental axioms. In presenting a proof, one usually tries to strike a balance: enough left out to avoid having the minor details obscure the main points and put the reader to sleep, and enough left in so that the reader will be convinced.

A variation of proof by contrapositive is *proof by contradiction*. In its most general form, proving a statement p by contradiction means showing that if it is not true, some contradiction results. Formally, this means showing that the statement $\neg p \rightarrow$ *false* is true. It follows that the contrapositive statement *true* $\rightarrow p$ is true, and this statement is logically equivalent to p. If we wish to prove the statement $p \rightarrow q$ by contradiction, we assume that $p \rightarrow q$ is false. Because of the logical equivalence of $p \rightarrow q$ and $\neg p \vee q$, this means assuming that $\neg(\neg p \vee q)$, or $p \wedge \neg q$, is true. From this assumption we try to derive some statement that contradicts some statement we know to be true—possibly p, or possibly some other statement.

EXAMPLE 2.3 $\sqrt{2}$ Is Irrational

A real number x is *rational* if there are two integers m and n so that $x = m/n$. We present one of the most famous examples of proof by contradiction: the proof, known to the ancient Greeks, that $\sqrt{2}$ is irrational.

■ **Proof**

Suppose for the sake of contradiction that $\sqrt{2}$ is rational. Then there are integers m' and n' with $\sqrt{2} = m'/n'$. By dividing both m' and n' by all the factors that are common to both, we obtain $\sqrt{2} = m/n$, for some integers m and n having no common factors. Since $m/n = \sqrt{2}$, $m = n\sqrt{2}$. Squaring both sides of this equation, we obtain $m^2 = 2n^2$, and therefore m^2 is even (divisible by 2). The result proved in Example 2.1 is that for any integers a and b, if a and b are odd, then ab is odd. Since a conditional statement is logically equivalent to its contrapositive, we may conclude that for any a and b, if ab is not odd, then either a is not odd or b is not odd. However, an integer is not odd if and only if it is even (Exercise 2.21), and so for any a and b, if ab is even, then a or b is even. If we apply this when $a = b = m$, we conclude that since m^2 is even, m must be even. This means that for some k, $m = 2k$. Therefore, $(2k)^2 = 2n^2$. Simplifying this and canceling 2 from both sides, we obtain $2k^2 = n^2$. Therefore, n^2 is even. The same argument that we have already used shows that n must be even, and so $n = 2j$ for some j. We have shown that m and n are both divisible by 2. This contradicts the previous statement that m and n have no common factor. The assumption that $\sqrt{2}$ is rational therefore leads to a contradiction, and the conclusion is that $\sqrt{2}$ is irrational.

EXAMPLE 2.4 Another Proof by Contradiction

To prove: For any sets A, B, and C, if $A \cap B = \emptyset$ and $C \subseteq B$, then $A \cap C = \emptyset$.

■ **Proof**

Again we try a proof by contradiction. Suppose that A, B, and C are sets for which the conditional statement is false. Then $A \cap B = \emptyset$, $C \subseteq B$, and $A \cap C \neq \emptyset$. Therefore, there exists x with $x \in A \cap C$, so that $x \in A$ and $x \in C$. Since $C \subseteq B$ and $x \in C$, it follows that $x \in B$. Therefore, $x \in A \cap B$, which contradicts the assumption that $A \cap B = \emptyset$. Since the assumption that the conditional statement is false leads to a contradiction, the statement is proved.

There is not always a clear line between a proof by contrapositive and one by contradiction. Any proof by contrapositive that $p \rightarrow q$ is true can easily be reformulated as a proof by contradiction. Instead of assuming that $\neg q$ is true and trying to show $\neg p$, assume that p and $\neg q$ are true and derive $\neg p$; then the contradiction is that p and $\neg p$ are both true. In the last example it seemed slightly easier to argue by contradiction, since we wanted to use the assumption that $C \subseteq B$. A proof by contrapositive would assume that $A \cap C \neq \emptyset$ and would try to show that

$$\neg((A \cap B = \emptyset) \wedge (C \subseteq B))$$

This approach seems a little more complicated, just because the formula we are trying to obtain is more complicated.

It is often convenient (or necessary) to use several different proof techniques within a single proof. Although the overall proof in the following example is not a proof by contradiction, this technique is used twice within the proof.

EXAMPLE 2.5 There Must Be a Prime Between n and $n!$

For a positive integer n the number $n!$ is defined to be the product $n * (n-1) * \cdots * 2 * 1$ of all the positive integers less than or equal to n. *To prove:* For any integer $n > 2$, there is a prime p satisfying $n < p < n!$.

■ **Proof**

Since $n > 2$, two distinct factors in $n!$ are n and 2. Therefore, $n! \geq 2n = n + n > n + 1$, and thus $n! - 1 > n$. The number $n! - 1$ must have a factor p that is a prime. (See Example 1.2 for the definition of a prime. The fact that every integer greater than 1 has a prime factor is a basic fact about positive integers, which we will prove in Example 2.11.) Since p is a divisor of $n! - 1$, $p \leq n! - 1 < n!$. This gives us one of the inequalities we need. To show the other one, suppose for the sake of contradiction that $p \leq n$. Then since p is one of the positive integers less than or equal to n, p is a factor of $n!$. However, p cannot be a factor of both $n!$ and $n! - 1$; if it were, it would be a factor of 1, their difference, and this is impossible. Therefore, the assumption that $p \leq n$ leads to a contradiction, and we may conclude that $n < p < n!$.

Another useful technique is to divide the proof into separate cases; this is illustrated by the next example.

EXAMPLE 2.6 Strings of Length 4 Contain Substrings yy

To prove: Every string x in $\{0, 1\}^*$ of length 4 contains a nonnull substring of the form yy.

■ **Proof**

We can show the result by considering two separate cases. If x contains two consecutive 0's or two consecutive 1's, then the statement is true for a string y of length 1. In the other case, any symbol that follows a 0 must be a 1, and vice versa, so that x must be either 0101 or 1010. The statement is therefore true for a string y of length 2.

Even though the argument is simple, let us state more explicitly the logic on which it depends. We want to show that some proposition P is true. The statement P is logically equivalent to $true \rightarrow P$. If we denote by p the statement that x contains two consecutive 0's or two consecutive 1's, then $p \vee \neg p$ is true. This means $true \rightarrow P$ is logically equivalent to

$$(p \vee \neg p) \rightarrow P$$

which in turn is logically equivalent to

$$(p \rightarrow P) \wedge (\neg p \rightarrow P)$$

This last statement is what we actually prove, by showing that each of the two separate conditional statements is true.

In this proof, there was some choice as to which cases to consider. A less efficient approach would have been to divide our two cases into four subcases: (i) x contains two consecutive 0's; (and so forth). An even more laborious proof would be to consider the 16 strings of length 4 individually, and to show that the result is true in each case. Any of these approaches is valid, as long as our cases cover all the possibilities and we can complete the proof in each case.

The examples in this section provide only a very brief introduction to proofs. Learning to read proofs takes a lot of practice, and creating your own is even harder. One thing that does help is to develop a critical attitude. Be skeptical. When you read a step in a proof, ask yourself, "Am I convinced by this?" When you have written a proof, read it over as if someone else had written it (it is best to read aloud if circumstances permit), and as you read each step ask yourself the same question.

2.2 THE PRINCIPLE OF MATHEMATICAL INDUCTION

Very often, we wish to prove that some statement involving a natural number n is true for every sufficiently large value of n. The statement might be a numerical equality:

$$\sum_{i=1}^{n} i = n(n+1)/2$$

The number of subsets of $\{1, 2, \ldots, n\}$ is 2^n.

It might be an inequality:

$$n! > 2^n$$

It might be some other assertion about n, or about a set with n elements, or a string of length n:

There exist positive integers j and k so that $n = 3j + 7k$.

Every language with exactly n elements is regular.

If $x \in \{0, 1\}^*$, $|x| = n$, and $x = 0y1$, then x contains the substring 01.

(The term *regular* is defined in Chapter 3.) In this section, we discuss a common approach to proving statements of this type.

In both the last two examples, it might seem as though the explicit mention of n makes the statement slightly more awkward. It would be simpler to say, "Every finite language is regular," and this statement is true; it would also be correct to let the last statement begin, "For any x and y in $\{0, 1\}^*$, if $x = 0y1, \ldots$." However, in both cases the simpler statement is equivalent to the assertion that the original statement is true for every nonnegative value of n, and formulating the statement so that it involves n will allow us to apply the proof technique we are about to discuss.

The Sum of the First n Positive Integers — EXAMPLE 2.7

We begin with the first example above, expressed without the summation notation:

$$1 + 2 + \cdots + n = n(n+1)/2$$

This formula is supposed to hold for every $n \geq 1$; however, it makes sense to consider it for $n = 0$ as well if we interpret the left side in that case to be the empty sum, which by definition is 0. Let us therefore try to prove that the statement is true for every $n \geq 0$.

How do we start? Unless we have any better ideas, we might very well begin by writing out the formula for the first few values of n, to see if we can spot a pattern.

$$\begin{aligned}
n = 0: &\quad 0 = 0(0+1)/2 \\
n = 1: &\quad 0 + 1 = 1(1+1)/2 \\
n = 2: &\quad 0 + 1 + 2 = 2(2+1)/2 \\
n = 3: &\quad 0 + 1 + 2 + 3 = 3(3+1)/2 \\
n = 4: &\quad 0 + 1 + 2 + 3 + 4 = 4(4+1)/2
\end{aligned}$$

As we are verifying these formulas, we probably realize after a few lines that in checking a specific case, say $n = 4$, it is not necessary to do all the arithmetic on the left side: $0+1+2+3+4$. We can take the left side of the previous formula, which we have already calculated, and add 4. When we calculated $0 + 1 + 2 + 3$, we obtained $3(3 + 1)/2$. So our answer for $n = 4$ is

$$3(3+1)/2 + 4 = 4(3/2 + 1) = 4(3+2)/2 = 4(4+1)/2$$

which is the one we wanted. Now that we have done this step, we can take care of $n = 5$ the same way, by taking the sum we just obtained for $n = 4$ and adding 5:

$$4(4+1)/2 + 5 = 5(4/2 + 1) = 5(4+2)/2 = 5(5+1)/2$$

These two calculations are similar—in fact, this is the pattern we were looking for, and we can probably see at this point that it will continue. Are we ready to write our proof?

■ Example 2.7. Proof Number 1

To show

$$0 + 1 + 2 + \cdots + n = n(n+1)/2 \qquad \text{for every } n \geq 0$$

$n = 0$: $0 = 0(0+1)/2$

$n = 1$: $0 + 1 = 0(0+1)/2 + 1$ (by using the result for $n = 0$)
$= 1(0/2 + 1)$
$= 1(0+2)/2$
$= 1(1+1)/2$

$n = 2$: $0 + 1 + 2 = 1(1+1)/2 + 2$ (by using the result for $n = 1$)
$= 2(1/2 + 1)$
$= 2(1+2)/2$
$= 2(2+1)/2$

$$
\begin{aligned}
n = 3: \quad 0+1+2+3 &= 2(2+1)/2+3 \quad \text{(by using the result for } n=2\text{)} \\
&= 3(2/2+1) \\
&= 3(2+2)/2 \\
&= 3(3+1)/2
\end{aligned}
$$

Since this pattern continues indefinitely, the formula is true for every $n \geq 0$.

Now let us criticize this proof. The conclusion, "the formula is true for every $n \geq 0$," is supposed to follow from the fact that "this pattern continues indefinitely." The phrase "this pattern" refers to the calculation that we have done three times, to derive the formula for $n = 1$ from $n = 0$, for $n = 2$ from $n = 1$, and for $n = 3$ from $n = 2$. There are at least two clear deficiencies in the proof. One is that we have not said explicitly what "this pattern" is. The second, which is more serious, is that we have not made any attempt to justify the assertion that it continues indefinitely. In this example, the pattern is obvious enough that people might accept the assertion without much argument. However, it would be fair to say that the most important statement in the proof is the one for which no reasons are given!

Our second version of the proof tries to correct both these problems at once: to describe the pattern precisely by doing the calculation, not just for three particular values of n but for an *arbitrary* value of n, and in the process, to demonstrate that the pattern does not depend on the value of n and therefore *does* continue indefinitely.

■ Example 2.7. Proof Number 2

To show

$$0+1+2+\cdots+n = n(n+1)/2 \qquad \text{for every } n \geq 0$$

$$
\begin{aligned}
n = 0: \quad & 0 = 0(0+1)/2 \\
n = 1: \quad & 0+1 = 0(0+1)/2+1 \quad \text{(by using the result for } n=0\text{)} \\
& = 1(0/2+1) \\
& = 1(0+2)/2 \\
& = 1(1+1)/2 \\
n = 2: \quad & 0+1+2 = 1(1+1)/2+2 \quad \text{(by using the result for } n=1\text{)} \\
& = 2(1/2+1) \\
& = 2(1+2)/2 \\
& = 2(2+1)/2 \\
n = 3: \quad & 0+1+2+3 = 2(2+1)/2+3 \quad \text{(by using the result for } n=2\text{)} \\
& = 3(2/2+1) \\
& = 3(2+2)/2 \\
& = 3(3+1)/2
\end{aligned}
$$

In general, for any value of $k \geq 0$, the formula for $n = k+1$ can be derived from the one for $n = k$ as follows:

$$
\begin{aligned}
0+1+2+\cdots+(k+1) &= (0+1+\cdots+k)+(k+1) \\
&= k(k+1)/2+(k+1) \quad \text{(from the result for } n=k\text{)} \\
&= (k+1)(k/2+1) \\
&= (k+1)(k+2)/2 \\
&= (k+1)((k+1)+1)/2
\end{aligned}
$$

Therefore, the formula holds for every $n \geq 0$.

We might now say that the proof has more than it needs. Presenting the calculations for three specific values of n originally made it easier for the reader to spot the pattern; now, however, the pattern has been stated explicitly. To the extent that the argument for these three specific cases is taken to be part of the proof, it obscures the two *essential* parts of the proof: (1) checking the formula for the initial value of n, $n = 0$, and (2) showing in general that once we have obtained the formula for one value of n ($n = k$), we can derive it for the next value ($n = k+1$). These two facts together are what allow us to conclude that the formula holds for every $n \geq 0$. Neither by itself would be enough. (On one hand, the formula for $n = 0$, or even for the first million values of n, might be true just by accident. On the other hand, it would not help to know that we can always derive the formula for the case $n = k+1$ from the one for the case $n = k$, if we could never get off the ground by showing that it *is* actually true for some starting value of k.)

The principle that we have used in this example can now be formulated in general.

The Principle of Mathematical Induction

Suppose $P(n)$ is a statement involving an integer n. Then to prove that $P(n)$ is true for every $n \geq n_0$, it is sufficient to show these two things:

1. $P(n_0)$ is true.
2. For any $k \geq n_0$, if $P(k)$ is true, then $P(k+1)$ is true.

A *proof by induction* is an application of this principle. The two parts of such a proof are called the *basis step* and the *induction* step. In the induction step, we *assume* that k is a number $\geq n_0$ and that the statement $P(n)$ is true in the case $n = k$; we call this assumption the *induction hypothesis*. Let us return to our example one last time in order to illustrate the format of a proof by induction.

■ Example 2.7. Proof Number 3 (by induction)

Let $P(n)$ be the statement

$$1+2+3+\cdots+n = n(n+1)/2$$

To show that $P(n)$ is true for every $n \geq 0$.

Basis step. We must show that $P(0)$ is true. $P(0)$ is the statement $0 = 0(0+1)/2$, and this is obviously true.

Induction hypothesis.

$$k \geq 0 \quad \text{and} \quad 1+2+3+\cdots+k = k(k+1)/2$$

Statement to be shown in induction step.

$$1+2+3+\cdots+(k+1) = (k+1)((k+1)+1)/2$$

Proof of induction step.

$$\begin{aligned} 1+2+3+\cdots+(k+1) &= (1+2+\cdots+k)+(k+1) \\ &= k(k+1)/2+(k+1) \quad \text{(by the induction hypothesis)} \\ &= (k+1)(k/2+1) \\ &= (k+1)(k+2)/2 \\ &= (k+1)((k+1)+1)/2 \end{aligned}$$

Whether or not you follow this format exactly, it is advisable always to include in your proof explicit statements of the following:

- The general statement involving n that is to be proved.
- The statement to which it reduces in the basis step (the general statement, but with n_0 substituted for n).
- The induction hypothesis (the general statement, with k substituted for n, and preceded by "$k \geq n_0$, and").
- The statement to be shown in the induction step (with $k+1$ substituted for n).
- The point during the induction step at which the induction hypothesis is used.

The advantage of formulating a general principle of induction is that it supplies a general framework for proofs of this type. If you read in a journal article the phrase "It can be shown by induction that . . . ," even if the details are missing, you can supply them. Although including these five items explicitly may seem laborious at first, the advantage is that it can help you to clarify for yourself exactly what you are trying to do in the proof. Very often, once you have gotten to this point, filling in the remaining details is a straightforward process.

EXAMPLE 2.8 Strings of the Form 0y1 Must Contain the Substring 01

Let us prove the following statement: For any $x \in \{0, 1\}^*$, if x begins with 0 and ends with 1 (i.e., $x = 0y1$ for some string y), then x must contain the substring 01.

You may wonder whether this statement requires an induction proof; let us begin with an argument that does not involve induction, at least explicitly. If $x = 0y1$ for some string $y \in \{0, 1\}^*$, then x must contain at least one 1. The *first* 1 in x cannot occur at the beginning, since x starts with 0; therefore, the first 1 must be immediately preceded by a 0, which means that x contains the substring 01. It would be hard to imagine a proof much simpler than this, and it seems convincing. It is interesting to observe, however, that this proof uses a fact about natural numbers (every nonempty subset has a smallest element) that is equivalent to

the principle of mathematical induction. We will return to this statement later, when we have a slightly modified version of the induction principle. See Example 2.12 and the discussion before that example.

In any case, we are interested in illustrating the principle of induction at least as much as in the result itself. Let us try to construct an induction proof. Our initial problem is that mathematical induction is a way of proving statements of the form "For every $n \geq n_0, \ldots,$" and our statement is not of this form. This is easy to fix, and the solution was suggested at the beginning of this section. Consider the statement $P(n)$: If $|x| = n$ and $x = 0y1$ for some string $y \in \{0, 1\}^*$, then x contains the substring 01. In other words, we are introducing an integer n into our statement, specifically in order to use induction. If we can prove that $P(n)$ is true for every $n \geq 2$, it will follow that the original statement is true. (The integer we choose is the length of the string, and we could describe the method of proof as *induction on the length of the string*. There are other possible choices; see Exercise 2.6.)

In the basis step, we wish to prove the statement "If $|x| = 2$ and $x = 0y1$ for some string $y \in \{0, 1\}^*$, then x contains the substring 01." This statement is true, because if $|x| = 2$ and $x = 0y1$, then y must be the null string Λ, and we may conclude that $x = 01$. Our induction hypothesis will be the statement: $k \geq 2$, and if $|x| = k$ and $x = 0y1$ for some string $y \in \{0, 1\}^*$, then x contains the substring 01. In the induction step, we must show: if $|x| = k+1$ and $x = 0y1$ for some $y \in \{0, 1\}^*$, then x contains the substring 01. (These three statements are obtained from the original statement $P(n)$ very simply: first, by substituting 0 for n; second, by substituting k for n, and adding the phrase "$k \geq 2$, and" at the beginning; third, by substituting $k+1$ for n. These three steps are always the same, and the basis step is often as easy to prove as it is here. Now the mechanical part is over, and we must actually think about how to continue the proof!)

We have a string x of length $k+1$, about which we want to prove something. We have an induction hypothesis that tells us something about certain strings of length k, the ones that begin with 0 and end with 1. In order to apply the induction hypothesis, we need a string of length k to apply it to. We can get a string of length k from x by leaving out one symbol. Let us try deleting the initial 0. (See Exercise 2.5.) The remainder, $y1$, is certainly a string of length k, and we know that it ends in 1, but it may not begin with 0—and we can apply the induction hypothesis only to strings that do. However, if $y1$ does not begin with 0, it must begin with 1, and in this case x starts with the substring 01! If $y1$ does begin with 0, then the induction hypothesis tells us that it must contain the substring 01, so that $x = 0y1$ must contain the substring too.

Now that we have figured out the crucial steps, we can afford to be a little more concise in our official proof. We are trying to prove that for every $n \geq 2$, $P(n)$ is true, where $P(n)$ is the statement: If $|x| = n$ and $x = 0y1$ for some string $y \in \{0, 1\}^*$, then x contains the substring 01.

Basis step. We must show that the statement $P(2)$ is true. $P(2)$ says that if $|x| = 2$ and $x = 0y$ for some $y \in \{0, 1\}^*$, then x contains the substring 01. $P(2)$ is true, because if $|x| = 2$ and $x = 0y1$ for some y, then $x = 01$.

Induction hypothesis. $k \geq 2$ and $P(k)$; in other words, if $|x| = k$ and $x = 0y1$ for some $y \in \{0, 1\}^*$, then x contains the substring 01.

Statement to be shown in induction step. $P(k+1)$; that is, if $|x| = k+1$ and $x = 0y1$ for some $y \in \{0, 1\}^*$, then x contains the substring 01.

Proof of induction step. Since $|x| = k + 1$ and $x = 0y1$, $|y1| = k$. If y begins with 1, then x begins with the substring 01. If y begins with 0, then $y1$ begins with 0 and ends with 1; by the induction hypothesis, y contains the substring 01, and therefore x does also.

EXAMPLE 2.9 Verifying a Portion of a Program

The program fragment below is written in pseudocode. Lowercase letters represent constants, uppercase letters represent variables, and the constant n is assumed to be nonnegative:

```
Y = 1;
for I = 1 to n
   Y = Y * x;
write(Y);
```

We would like to show that when this code is executed, the value printed out is x^n. We do this in a slightly roundabout way, by introducing a new integer j, the number of iterations of the loop that have been performed. Let $P(j)$ be the statement that the value of Y after j iterations is x^j. The result we want will follow from the fact that $P(j)$ is true for any $j \geq 0$, and the fact that "For $I = 1$ to n" results in n iterations of the loop.

Basis step. $P(0)$ is the statement that after 0 iterations of the loop, Y has the value x^0. This is true because Y receives the initial value 1 and after 0 iterations of the loop its value is unchanged.

Inductive hypothesis. $k \geq 0$, and after k iterations of the loop the value of Y is x^k.

Statement to be proved in induction step. After $k + 1$ iterations of the loop, the value of Y is x^{k+1}.

Proof of induction step. The effect of the assignment statement $Y = Y * x$ is to replace the old value of Y by that value times x; therefore, the value of Y after any iteration is x times the value before that iteration. Since $x * x^k = x^{k+1}$, the proof is complete.

Although the program fragment in this example is very simple, the example should suggest that the principle of mathematical induction can be a useful technique for verifying the correctness of programs. For another example, see Exercise 2.56.

You may occasionally find the principle of mathematical induction in a disguised form, which we could call the *minimal counterexample principle*. The last example in this section illustrates this.

EXAMPLE 2.10 A Proof Using the Minimal Counterexample Principle

To show: For every integer $n \geq 0$, $5^n - 2^n$ is divisible by 3.

Just as in an ordinary induction proof, we begin by checking that $P(n)$ is true for the starting value of n. This is true here, since $5^0 - 2^0 = 1 - 1 = 0$, and 0 is divisible by 3. Now if it is *not* true that $P(n)$ is true for every $n \geq 0$, then there are values of n greater than or equal to 0 for which $P(n)$ is false, and therefore there must be a smallest such value, say $n = k$.

(See Example 2.12.) Since we have verified $P(0)$, k must be at least 1. Therefore, $k - 1$ is at least 0, and since k is the smallest value for which P fails, $P(k - 1)$ is true. This means that $5^{k-1} - 2^{k-1}$ is a multiple of 3, say $3j$. Then, however,

$$5^k - 2^k = 5 * 5^{k-1} - 2 * 2^{k-1} = 3 * 5^{k-1} + 2 * (5^{k-1} - 2^{k-1}) = 3 * 5^{k-1} + 2 * 3j$$

This expression is divisible by 3. We have derived a contradiction, which allows us to conclude that our original assumption is false. Therefore, $P(n)$ is true for every $n \geq 0$.

You can probably see the similarity between this proof and one that uses the principle of mathematical induction. Although an induction proof has the advantage that it does not involve proof by contradiction, both approaches are equally valid.

Not every statement involving an integer n is appropriate for mathematical induction. Using this technique on the statement

$$(2^n + 1)(2^n - 1) = 2^{2n} - 1$$

would be silly because the proof of the induction step would not require the induction hypothesis at all. The formula for $n = k + 1$, or for any other value, can be obtained immediately by expanding the left side of the formula and using laws of exponents. The proof would not be a *real* induction proof, and it would be misleading to classify it as one.

A general rule of thumb is that if you are tempted to use a phrase like "Repeat this process for each n," or "Since this pattern continues indefinitely" in a proof, there is a good chance that the proof can be made more precise by using mathematical induction. When you encounter one of these phrases while reading a proof, it is very likely a substitute for an induction argument. In this case, supplying the details of the induction may help you to understand the proof better.

2.3 THE STRONG PRINCIPLE OF MATHEMATICAL INDUCTION

Sometimes, as in our first example, a proof by mathematical induction is called for, but the induction principle in Section 2.2 is not the most convenient tool.

EXAMPLE 2.11 Integers Bigger Than 2 Have Prime Factorizations

Recall that a *prime* is a positive integer, 2 or bigger, that has no positive integer divisors except itself and 1. Part of the fundamental theorem of arithmetic is that every integer can be factored into primes. More precisely, let $P(n)$ be the statement that n is either prime or the product of two or more primes; we will try to prove that $P(n)$ is true for every $n \geq 2$.

The basis step does not present any problems. $P(2)$ is true, since 2 is a prime. If we proceed as usual, then we take as the induction hypothesis the statement that $k \geq 2$ and k is either prime or the product of two or more primes. We would like to show that $k + 1$ is either prime or the product of primes. If $k + 1$ happens to be prime, there is nothing left to prove. Otherwise, by the definition of prime, $k + 1$ has some positive integer divisor other than itself

and 1. This means $k + 1 = r * s$ for some positive integers r and s, neither of which is 1 or $k + 1$. It follows that r and s must both be greater than 1 and less than $k + 1$.

In order to finish the induction step, we would like to show that r and s are both either primes or products of primes; it would then follow, since $k + 1$ is the product of r and s, that $k + 1$ is a product of two or more primes. Unfortunately, the only information our induction hypothesis gives us is that k is a prime or a product of primes, and this tells us nothing about r or s.

Consider, however, the following intuitive argument, in which we set about verifying the statement $P(n)$ one value of n at a time:

2 is a prime.

3 is a prime.

$4 = 2 * 2$, which is a product of primes since $P(2)$ is known to be true.

5 is a prime.

$6 = 2 * 3$, which is a product of primes since $P(2)$ and $P(3)$ are known to be true.

7 is a prime.

$8 = 2 * 4$, which is a product of primes since $P(2)$ and $P(4)$ are known to be true.

$9 = 3 * 3$, which is a product of primes since $P(3)$ is known to be true.

$10 = 2 * 5$, which is a product of primes since $P(2)$ and $P(5)$ are known to be true.

11 is a prime.

$12 = 2 * 6$, which is a product of primes since $P(2)$ and $P(6)$ are known to be true.

...

This seems as convincing as the intuitive argument given at the start of Example 2.7. Furthermore, we can describe explicitly the pattern illustrated by the first 11 steps: For each $k \geq 2$, either $k + 1$ is prime or it is the product of two numbers r and s for which the proposition P has already been shown to hold.

The difference between the pattern appearing here and the one we saw in Example 2.7 is this: At each step in the earlier example we were able to obtain the truth of $P(k + 1)$ by knowing that $P(k)$ was true, and here we need to know that P holds, not only for k but also for all the values up to k. The following modified version of the induction principle will allow our proof to proceed.

The Strong Principle of Mathematical Induction

Suppose $P(n)$ is a statement involving an integer n. Then to prove that $P(n)$ is true for every $n \geq n_0$, it is sufficient to show these two things:

1. $P(n_0)$ is true.
2. For any $k \geq n_0$, if $P(n)$ is true for every n satisfying $n_0 \leq n \leq k$, then $P(k + 1)$ is true.

To use this principle in a proof, we follow the same steps as before except for the way we state the induction hypothesis. The statement here is that k is some integer $\geq n_0$ and that *all* the statements $P(n_0)$, $P(n_0 + 1)$, ..., $P(k)$ are true. With this change, we can finish the proof we began earlier.

■ Example 2.11. Proof by induction.

To show: $P(n)$ is true for every $n \geq 2$, where $P(n)$ is the statement: n is either a prime or a product of two or more primes.

Basis step. $P(2)$ is the statement that 2 is either a prime or a product of two or more primes. This is true because 2 is a prime.

Induction hypothesis. $k \geq 2$, and for every n with $2 \leq n \leq k$, n is either prime or a product of two or more primes.

Statement to be shown in induction step. $k + 1$ is either prime or a product of two or more primes.

Proof of induction step. We consider two cases. If $k + 1$ is prime, the statement $P(k + 1)$ is true. Otherwise, by definition of a prime, $k + 1 = r * s$, for some positive integers r and s, neither of which is 1 or $k + 1$. It follows that $2 \leq r \leq k$ and $2 \leq s \leq k$. Therefore, by the induction hypothesis, both r and s are either prime or the product of two or more primes. Therefore, their product $k + 1$ is the product of two or more primes, and $P(k + 1)$ is true.

The strong principle of induction is also referred to as the principle of *complete* induction, or *course-of-values* induction. The first example suggests that it is as plausible intuitively as the ordinary induction principle, and in fact the two are equivalent. As to whether they are *true*, the answer may seem a little surprising. Neither can be proved using other standard properties of the natural numbers. (Neither can be disproved, either!) This means, in effect, that in order to use the induction principle, we must adopt it as an axiom. A well-known set of axioms for the natural numbers, the *Peano* axioms, includes one similar to the induction principle.

Twice in Section 2.2 we had occasion to use the *well-ordering* principle for the natural numbers, which says that every nonempty subset of $\mathcal{N}$ has a smallest element. As obvious as this statement probably seems, it is also impossible to prove without using induction or something comparable. In the next example, we show that it follows from the strong principle of induction. (It can be shown to be equivalent.)

The Well-ordering Principle for the Natural Numbers EXAMPLE 2.12

To prove: Every nonempty subset of $\mathcal{N}$, the set of natural numbers, has a smallest element. (What we are actually proving is that if the strong principle of mathematical induction is true, then every nonempty subset of $\mathcal{N}$ has a smallest element.)

First we need to find a way to express the result in the form "For every $n \geq n_0$, $P(n)$." Every nonempty subset A of $\mathcal{N}$ contains a natural number, say n. If every subset of $\mathcal{N}$

containing n has a smallest element, then A does. With this in mind, we let $P(n)$ be the statement "Every subset of $\mathcal{N}$ containing n has a smallest element." We prove that $P(n)$ is true for every $n \geq 0$. (See Exercise 2.7.)

Basis step. $P(0)$ is the statement that every subset of $\mathcal{N}$ containing 0 has a smallest element. This is true because 0 is the smallest natural number and therefore the smallest element of the subset.

Induction hypothesis. $k \geq 0$, and for every n with $0 \leq n \leq k$, every subset of $\mathcal{N}$ containing n has a smallest element. (Put more simply, $k \geq 0$ and every subset of $\mathcal{N}$ containing an integer less than or equal to k has a smallest element.)

Statement to be shown in induction step. Every subset of $\mathcal{N}$ containing $k+1$ has a smallest element.

Proof of induction step. Let A be any subset of $\mathcal{N}$ containing $k+1$. We consider two cases. If A contains no natural number less than $k+1$, then $k+1$ is the smallest element of A. Otherwise, A contains some natural number n with $n \leq k$. In this case, by the induction hypothesis, A contains a smallest element.

The strong principle of mathematical induction is more appropriate here, since when we come up with an n to which we want to apply the induction hypothesis, all we know about n is that $n \leq k$. We do *not* know that $n = k$. It may not be obvious at the beginning of an induction proof whether the strong induction principle is required or whether you can get by with the original version. You can avoid worrying about this by *always* using the strong version. It allows you to adopt a stronger induction hypothesis, and so if an induction proof is possible at all, it will certainly be possible with the strong version. In any case, you can put off the decision until you reach the point where you have to prove $P(k+1)$. If you can do this with only the assumption that $P(k)$ is true, then the original principle of induction is sufficient. If you need information about earlier values of n as well, the strong version is needed.

We will see more examples of how the strong principle of mathematical induction is applied once we have discussed recursive definitions and the close relationship between them and mathematical induction.

2.4 | RECURSIVE DEFINITIONS

2.4.1 Recursive Definitions of Functions with Domain $\mathcal{N}$

The chances are that in a programming course you have seen a translation into some high-level programming language of the following definition:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1)! & \text{if } n > 0 \end{cases}$$

This is one of the simplest examples of a recursive, or inductive, definition. It defines the factorial function on the set of natural numbers, first by defining the value at 0,

and then by defining the value at any larger natural number in terms of its value at the previous one. There is an obvious analogy here to the basis step and the induction step in a proof by mathematical induction. The intuitive reason this is a valid definition is the same as the intuitive reason the principle of induction should be true: If we think of defining $n!$ for all the values of n in order, beginning with $n = 0$, then for any $k \geq 0$, eventually we will have defined the value of $k!$, and at that point the definition will tell us how to obtain $(k+1)!$.

In this section, we will look at a number of examples of recursive definitions of functions and examine more closely the relationship between recursive definitions and proofs by induction. We begin with more examples of functions on the set of natural numbers.

EXAMPLE 2.13 The Fibonacci Function

The Fibonacci function f is usually defined as follows:

$$f(0) = 1$$
$$f(1) = 1$$
$$\text{for every } n \geq 1,\ f(n+1) = f(n) + f(n-1)$$

To evaluate $f(4)$, for example, we can use the definition in either a top-down fashion:

$$\begin{aligned} f(4) &= f(3) + f(2) \\ &= (f(2) + f(1)) + f(2) \\ &= ((f(1) + f(0)) + f(1)) + (f(1) + f(0)) \\ &= ((1+1)+1) + (1+1) \\ &= 5 \end{aligned}$$

or a bottom-up fashion:

$$\begin{aligned} f(0) &= 1 \\ f(1) &= 1 \\ f(2) &= f(1) + f(0) = 1 + 1 = 2 \\ f(3) &= f(2) + f(1) = 2 + 1 = 3 \\ f(4) &= f(3) + f(2) = 3 + 2 = 5 \end{aligned}$$

It is possible to give a nonrecursive algebraic formula for the number $f(n)$; see Exercise 2.53. However, the recursive definition is the one that most people remember and prefer to use.

If the definition of the factorial function is reminiscent of the principle of mathematical induction, then the definition of the Fibonacci function suggests the strong principle of induction. This is because the definition of $f(n+1)$ involves not only $f(n)$ but $f(n-1)$. This observation is useful in proving facts about f. For example, let us prove that

$$\text{for every } n \geq 0,\ f(n) \leq (5/3)^n$$

Basis step. We must show that $f(0) \leq (5/3)^0$; this is true, since $f(0)$ and $(5/3)^0$ are both 1.

Induction hypothesis. $k \geq 0$, and for every n with $0 \leq n \leq k$, $f(n) \leq (5/3)^n$.

Statement to show in induction step. $f(k+1) \leq (5/3)^{k+1}$.

Proof of induction step. Because of the way f is defined, we consider two cases in order to make sure that our proof is valid for *every* value of $k \geq 0$. If $k = 0$, then $f(k+1) = f(1) = 1$, by definition, and in this case the inequality is clearly true. If $k > 0$, then we must use the recursive part of the definition, which is $f(k+1) = f(k) + f(k-1)$. Since both k and $k-1$ are $\leq k$, we may apply the induction hypothesis to both terms, obtaining

$$\begin{aligned} f(k+1) &= f(k) + f(k-1) \\ &\leq (5/3)^k + (5/3)^{k-1} \\ &= (5/3)^{k-1}(5/3+1) \\ &= (5/3)^{k-1}(8/3) \\ &= (5/3)^{k-1}(24/9) \\ &< (5/3)^{k-1}(25/9) \;=\; (5/3)^{k+1} \end{aligned}$$

EXAMPLE 2.14 The Union of n Sets

Suppose $A_1, A_2, \ldots$ are subsets of some universal set U. For each $n \geq 0$, we may define $\bigcup_{i=1}^{n} A_i$ as follows:

$$\bigcup_{i=1}^{0} A_i = \emptyset$$

$$\text{for every } n \geq 0,\ \bigcup_{i=1}^{n+1} A_i = \left(\bigcup_{i=1}^{n} A_i\right) \cup A_{n+1}$$

For each n, $\bigcup_{i=1}^{n} A_i$ is a set, in particular a subset of U; therefore, it makes sense to view the recursive definition as defining a function from the set of natural numbers to the set of subsets of U. What we have effectively done in this definition is to extend the binary operation of union so that it can be applied to n operands. (We discussed this possibility for associative operations like union near the end of Section 1.1.) This procedure is familiar to you in other settings, although you may not have encountered a formal recursive definition like this one before. When you add n numbers in an expression like $\sum_{i=1}^{n} a_i$, for example, you are extending the binary operation of addition to n operands. The notational device used to do this is the summation sign: Σ bears the same relation to the binary operation $+$ that $\bigcup$ does to the binary operation $\cup$. A recursive definition of Σ would follow the one above in every detail:

$$\sum_{i=1}^{0} a_i = 0$$

$$\text{for every } n \geq 0,\ \sum_{i=1}^{n+1} a_i = \left(\sum_{i=1}^{n} a_i\right) + a_{n+1}$$

Similarly, we could define the intersection of n sets, the product of n numbers, the concatenation of n strings, the concatenation of n languages, and so forth. (The only difficulty we would have

with the last two is that we have not introduced a notation for the concatenation operator—we have written the concatenation of x and y as simply xy.) We are also free to use one of these general definitions in a special case. For example, we might consider the concatenation of n languages, all of which are the same language L:

$$L^0 = \{\Lambda\}$$

$$\text{for every } n \geq 0,\ L^{n+1} = L^n\, L$$

Of course, we already have nonrecursive definitions of many of these things. Our definition of $\bigcup_{i=1}^{n} A_i$ in Section 1.1 is

$$\bigcup_{i=1}^{n} A_i = \{x \mid x \in A_i \text{ for at least one } i \text{ with } 1 \leq i \leq n\}$$

The definition of L^n in Section 1.5 is

$$L^n = LL\cdots L \ (n \text{ factors in all})$$

and we also described it as the set of all strings that can be obtained by concatenating n elements of L. It may not be obvious that we have gained anything by introducing the recursive definition. The nonrecursive definition of the n-fold union is clear enough, and even with the ellipses (...) it is not difficult to determine which strings are included in L^n. However, the recursive definition has the same advantage over the ellipses that we discussed in the first proof in Example 2.7; rather than *suggesting* what the general step is in this n-fold concatenation, it comes right out and says it. After all, when you construct an element of L^n, you concatenate two strings, not n strings at once. The recursive definition is more consistent with the *binary* nature of concatenation, and more explicit about how the concatenation is done: An $(n+1)$-fold concatenation is obtained by concatenating an element of L^k and an element of L. The recursive definition has a dynamic, or algorithmic, quality that the other one lacks. Finally, and probably most important, the recursive definition has a practical advantage. It gives us a natural way of constructing proofs using mathematical induction.

Suppose we want to prove the generalized De Morgan law:

$$\text{for every } n \geq 0,\ \left(\bigcup_{i=1}^{n} A_i\right)' = \bigcap_{i=1}^{n} A_i'$$

In the induction step, we must show something about

$$\left(\bigcup_{i=1}^{k+1} A_i\right)'$$

Using the recursive definition, we begin by replacing this with

$$\left(\left(\bigcup_{i=1}^{k} A_i\right) \cup A_{k+1}\right)'$$

at which point we can use the original De Morgan law to complete the proof, since we have expressed the $(k+1)$-fold union as a *two*-fold union.

This example illustrates again the close relationship between the principle of mathematical induction and the idea of recursive definitions. Not only are the two ideas similar in principle, they are almost inseparable in practice. Recursive definitions are useful in constructing induction proofs, and induction is the natural proof technique to use on objects defined recursively.

The relationship is so close, in fact, that in induction proofs we might use recursive definitions without realizing it. In Example 2.7, we proved that

$$1 + 2 + \cdots + n = n(n+1)/2$$

and the crucial observation in the induction step was that

$$(1 + 2 + \cdots + (k+1)) = (1 + 2 + \cdots + k) + (k+1)$$

This is exactly the formula we would have obtained from the recursive definition of the summation operator Σ in Example 2.14. In other words, although we had not formally adopted this definition at the time, the property of summation that we needed for the induction argument was the one the definition provides.

2.4.2 Recursive Definitions of Sets

We can also define a single set recursively. Although such a definition may not involve an integer n explicitly, the principle is similar. We specify certain objects that are in the set to start with, and we describe one or more general methods for obtaining new elements of the set from existing ones.

EXAMPLE 2.15 Recursive Definition of L^*

Suppose L is a language over some alphabet Σ. We have previously defined L^* as the union of the sets L^n for $n \geq 0$. From our recursive definition of L^n it follows that for any n, any $x \in L^n$, and any $y \in L$, the string xy is an element of L^{n+1} and therefore of L^*. Furthermore, every element of L^* can be obtained this way except Λ, which comes from L^0. This suggests the following more direct recursive definition of L^*.

1. $\Lambda \in L^*$.
2. For any $x \in L^*$ and any $y \in L$, $xy \in L^*$.
3. No string is in L^* unless it can be obtained by using rules 1 and 2.

To illustrate, let $L = \{a, ab\}$. According to rule 1, $\Lambda \in L^*$. One application of rule 2 adds the strings in $L^1 = L$, which are $\Lambda a = a$ and $\Lambda ab = ab$. Another application of rule 2 adds the strings in L^2, which are Λa, Λab, aa, aab, aba, and $abab$. For any $k \geq 0$, a string obtained by concatenating k elements of L can be produced from the definition by using k applications of rule 2.

An even simpler illustration is to let L be Σ, which is itself a language over Σ. Then a string of length k in Σ^*, which is a concatenation of k symbols belonging to Σ, can be produced by k applications of rule 2.

This way of defining L^* recursively is not the only way. Here is another possibility.

1. $\Lambda \in L^*$.
2. For any $x \in L$, $x \in L^*$.
3. For any two elements x and y of L^*, $xy \in L^*$.
4. No string is in L^* unless it can be obtained by using rules 1, 2, and 3.

In this approach, rules 1 and 2 are both necessary in the basis part of the definition, since rule 1 by itself would provide no strings other than Λ to concatenate in the recursive part.

The first definition is a little closer to our original definition of L^*, and perhaps a little easier to work with, because there is a direct correspondence between the applications of rule 2 needed to generate an element of L^* and the strings of L that are being concatenated. The second definition allows more flexibility as to how to produce a string in the language. There is a sense, however, in which both definitions capture the idea of all possible concatenations of strings of L, and for this reason it may not be too difficult to convince yourself that both definitions really do work—that the set being defined is L^* in each case. Exercise 2.63 asks you to consider the question in more detail.

EXAMPLE 2.16 Palindromes

Let Σ be any alphabet. The language *pal* of *palindromes* over Σ can be defined as follows:

1. $\Lambda \in pal$.
2. For any $a \in \Sigma$, $a \in pal$.
3. For any $x \in pal$ and any $a \in \Sigma$, $axa \in pal$.
4. No string is in *pal* unless it can be obtained by using rules 1, 2, and 3.

The strings that can be obtained by using rules 1 and 2 exclusively are the elements of *pal* of even length, and those obtained by using rules 2 and 3 are of odd length. A simple nonrecursive definition of *pal* is that it is the set of strings that read the same backwards as forwards. (See Exercise 2.60.)

In Example 2.14, we mentioned the algorithmic, or constructive, nature of the recursive definition of L^n. In the case of a language such as *pal*, this aspect of the recursive definition can be useful both from the standpoint of generating elements of the language and from the standpoint of recognizing elements of the language. The definition says, on the one hand, that we can construct palindromes by starting with either Λ or a single symbol, and continuing to concatenate a symbol onto both ends of the current string. On the other hand, it says that if we wish to test a string to see if it's a palindrome, we may first compare the leftmost and rightmost symbols, and if they are equal, reduce the problem to testing the remaining substring.

EXAMPLE 2.17 Fully Parenthesized Algebraic Expressions

Let Σ be the alphabet $\{i, (,), +, -\}$. Below is a recursive definition of the language AE of fully parenthesized algebraic expressions involving the binary operators $+$ and $-$ and the identifier i. The term *fully parenthesized* means exactly one pair of parentheses for every operator.

1. $i \in AE$.
2. For any $x, y \in AE$, both $(x + y)$ and $(x - y)$ are elements of AE.
3. No string is in AE unless it can be obtained by using rules (1) and (2).

Some of the strings in AE are i, $(i + i)$, $(i - i)$, $((i + i) - i)$, and $((i - (i - i)) + i)$.

EXAMPLE 2.18 Finite Subsets of the Natural Numbers

We define a set $\mathcal{F}$ of subsets of the natural numbers as follows:

1. $\emptyset \in \mathcal{F}$.
2. For any $n \in \mathcal{N}$, $\{n\} \in \mathcal{F}$.
3. For any A and B in $\mathcal{F}$, $A \cup B \in \mathcal{F}$.
4. Nothing is in $\mathcal{F}$ unless it can be obtained by using rules 1, 2, and 3.

We can obtain any two-element subset of $\mathcal{N}$ by starting with two one-element sets and using rule 3. Because we can then apply rule 3 to any two-element set A and any one-element set B, we obtain all the three-element subsets of $\mathcal{N}$. It is easy to show using mathematical induction that for any natural number n, any n-element subset of $\mathcal{N}$ is an element of $\mathcal{F}$; we may conclude that $\mathcal{F}$ is the collection of all finite subsets of $\mathcal{N}$.

Let us consider the last statement in each of the recursive definitions in this section. In each case, the previous statements describe ways of producing new elements of the set being defined. The last statement is intended to remove any ambiguity: Unless an object can be shown using these previous rules to belong to the set, it does *not* belong to the set.

We might choose to be even a little more explicit. In Example 2.17, we might say, "No string is in AE unless it can be obtained by *a finite number of applications of* rules 1 and 2." Here this extra precision hardly seems necessary, as long as it is understood that "string" means something of finite length; in Example 2.18 it is easier to see that it might be appropriate. One might ask whether there are any infinite sets in $\mathcal{F}$. We would hope that the answer is "no" because we have already agreed that the definition is a reasonable way to define the collection of *finite* subsets of $\mathcal{N}$. On the one hand, we could argue that the only way rule 3 could produce an infinite set would be for one of the sets A or B to be infinite already. On the other hand, think about using the definition to show that an infinite set C is not in $\mathcal{F}$. This means showing that C cannot be obtained by using rule 3. For an infinite set C to be obtained this way, either A or B (both of which must be elements of $\mathcal{F}$) would have to be infinite—but how do we know that cannot happen? The definition is not really precise unless it makes it clear that rules 1 to 3 can be used only a finite number of times in obtaining an element of $\mathcal{F}$. Remember also that we think of a recursive definition as a constructive, or algorithmic, definition, and that in any actual construction we would be able to apply any of the rules in the definition only a finite number of times.

Let us describe even more carefully the steps that would be involved in "a finite number of applications" of rules 1 to 3 in Example 2.18. Take a finite subset A of $\mathcal{N}$ that we might want to obtain from the definition of $\mathcal{F}$, say $A = \{2, 3, 7, 11, 14\}$. There are a number of ways we can use the definition to show that $A \in \mathcal{F}$. One obvious approach is to start with $\{2\}$ and use rule 3 four times, adding one more element each time, so that the four steps give us the subsets $\{2, 3\}$, $\{2, 3, 7\}$, $\{2, 3, 7, 11\}$, and $\{2, 3, 7, 11, 14\}$. Each time, the new subset is obtained by applying rule 3 to two sets, one a set we had obtained earlier by using rule 3, the other a one-element set

(one of the elements of $\mathcal{F}$ specified explicitly by rule 2). Another approach would be to start with two one-element sets, say $\{2\}$ and $\{7\}$, to add elements one at a time to each so as to obtain the sets $\{2, 3\}$ and $\{7, 11, 14\}$, and then to use rule 3 once more to obtain their union. In both of these approaches, we can write a *sequence* of sets representing the preliminary steps we take to obtain the one we want. We might include in our sequence all the one-element sets we use, in addition to the two-, three-, or four-element subsets we obtain along the way. In the first case, therefore, the sequence might look like this:

1. $\{2\}$ 2. $\{3\}$ 3. $\{2, 3\}$ 4. $\{7\}$
5. $\{2, 3, 7\}$ 6. $\{11\}$ 7. $\{2, 3, 7, 11\}$ 8. $\{14\}$
9. $\{2, 3, 7, 11, 14\}$

and in the second case the sequence might be

1. $\{2\}$ 2. $\{3\}$ 3. $\{2, 3\}$ 4. $\{7\}$
5. $\{11\}$ 6. $\{7, 11\}$ 7. $\{14\}$ 8. $\{7, 11, 14\}$
9. $\{2, 3, 7, 11, 14\}$

In both cases, there is considerable flexibility as to the order of the terms. The significant feature of both sequences is that every term is either one of the specific sets mentioned in statements 1 and 2 of the definition, or it is obtained from two terms appearing earlier in the sequence by using statement 3.

A precise way of expressing statement 4 in the definition would therefore be to say:

> No set A is in $\mathcal{F}$ unless there is a positive integer n and a sequence $A_1, A_2, \ldots, A_n$, so that $A_n = A$, and for every i with $1 \leq i \leq n$, A_i is either $\emptyset$, or a one-element set, or $A_j \cup A_k$ for some $j, k < i$.

A recursive definition of a set like $\mathcal{F}$ is not usually this explicit. Probably the most common approach is to say something like our original statement 4; a less formal approach would be to say "Nothing else is in $\mathcal{F}$," and sometimes the final statement is skipped altogether.

EXAMPLE 2.19 An Induction Proof Involving a Language Defined Recursively

Suppose that the language L, a subset of $\{0, 1\}^*$, is defined recursively as follows.

1. $\Lambda \in L$.
2. For any $y \in L$, both $0y$ and $0y1$ are in L.
3. No string is in L unless it can be obtained from rules 1 and 2.

In order to determine what the strings in L are, we might try to use the definition to generate the first few elements. We know $\Lambda \in L$. From rule 2 it follows that 0 and 01 are both in L. Using rule 2 again, we obtain 00, 001, and 0011; one more application produces 000, 0001, 00011, and 000111. After studying these strings, we may be able to guess that the strings in L are all those of the form 0^i1^j, where $i \geq j \geq 0$. Let us prove that every string of this form is in L.

To simplify the notation, let $A = \{0^i 1^j \mid i \geq j \geq 0\}$. The statement $A \subseteq L$, as it stands, does not involve an integer. Just as we did in Example 2.8, however, we can introduce the length of a string as an integer on which to base an induction proof.

To prove: $A \subseteq L$; i.e., for every $n \geq 0$, every $x \in A$ satisfying $|x| = n$ is an element of L.

Basis step. We must show that every x in A with $|x| = 0$ is an element of L. This is true because if $|x| = 0$, then $x = \Lambda$, and statement 1 in the definition of L tells us that $\Lambda \in L$.

Induction hypothesis. $k \geq 0$, and every x in A with $|x| = k$ is an element of L.

Statement to show in induction step. Every x in A with $|x| = k + 1$ is an element of L.

Proof of induction step. Suppose $x \in A$, and $|x| = k + 1$. Then $x = 0^i 1^j$, where $i \geq j \geq 0$, and $i + j = k + 1$. We are trying to show that $x \in L$. According to the definition, the only ways this can happen are for x to be Λ, for x to be $0y$ for some $y \in L$, and for x to be $0y1$ for some $y \in L$. The first case is impossible, since $|x| = k + 1 > 0$. In either of the other cases, we can obtain the conclusion we want *if* we know that the string y is in L. Presumably we will show this by using the induction hypothesis. However, at this point we have a slight problem. If $x = 0y1$, then the string y to which we want to apply the induction hypothesis is not of length k, but of length $k - 1$. Fortunately, the solution to the problem is easy: Use the strong induction principle instead, which allows us to use the stronger induction hypothesis. The statement to be shown in the induction step remains the same.

Induction hypothesis (revised). $k \geq 0$, and every x in A with $|x| \leq k$ is an element of L.

Proof of induction step (corrected version). Suppose $x \in A$ and $|x| = k + 1$. Then $x = 0^i 1^j$, where $i \geq j \geq 0$ and $i + j = k + 1$. We consider two cases. If $i > j$, then we can write $x = 0y$ for some y that still has at least as many 0's as 1's (i.e., for some $y \in A$). In this case, since $|y| = k$, it follows from the induction hypothesis that $y \in L$; therefore, since $x = 0y$, it follows from the first part of statement 2 in the definition of L that x is also an element of L. In the second case, when $i = j$, we know that there must be at least one 0 and one 1 in the string. Therefore, $x = 0y1$ for some y. Furthermore, $y \in A$, because $y = 0^{i-1}1^{j-1}$ and $i = j$. Since $|y| \leq k$, it follows from the induction hypothesis that $y \in L$. Since $x = 0y1$, we can use the second part of statement 2 in the definition of L to conclude that $x \in L$.

The two sets L and A are actually equal. We have now proved half of this statement. The other half, the statement $L \subseteq A$, can also be proved by induction on the length of the string (see Exercise 2.41). In the next section, however, we consider another approach to an induction proof, which is more naturally related to the recursive definition of L and is probably easier.

2.5 | STRUCTURAL INDUCTION

We have already noticed the very close correspondence between recursive definitions of functions on the set $\mathcal{N}$ and proofs by mathematical induction of properties of those functions. The correspondence is exact, in the sense that when we want to

prove something about $f(k + 1)$ in the induction step of a proof, we can consult the definition of $f(k + 1)$, the recursive part of the definition.

When we began to formulate recursive definitions of sets, there was usually no integer involved explicitly (see Examples 2.15–2.19), and it may have appeared that any correspondence between the recursive definition and induction proofs involving elements of the set would be less direct (even though there is a general similarity between the recursive definition and the induction principle). In this section, we consider the correspondence more carefully. We can start by identifying an integer that arises naturally from the recursive definition, which can be introduced for the purpose of an induction proof. However, when we look more closely at the proof, we will be able to dispense with the integer altogether, and what remains will be an induction proof based on the structure of the definition itself.

The example we use to illustrate this principle is essentially a continuation of Example 2.19.

Continuation of Example 2.19 — EXAMPLE 2.20

We have the language L, defined recursively as follows:

1. $\Lambda \in L$.
2. For every $x \in L$, both $0x$ and $0x1$ are in L.

The third statement in the definition of L says that every element of L is obtained by starting with Λ and applying statement 2 a finite number of times (zero or more).

We also have the language $A = \{0^i 1^j \mid i \geq j \geq 0\}$. In Example 2.19, we proved that $A \subseteq L$, using mathematical induction on the length of the string. Now we wish to prove the opposite inclusion $L \subseteq A$. As we have already pointed out, there is no reason why using the length of the string would not work in an induction proof in this direction too—a string in L has a length, just as every other string does. However, for each element x of L, there is another integer that is associated with x not just because x is a string, but specifically because x is an element of L. This is the number of times rule 2 is used in order to obtain x from the definition. We construct an induction proof, based on this number, that $L \subseteq A$.

To prove: For every $n \geq 0$, every $x \in L$ obtained by n applications of rule 2 is an element of L.

Basis step. We must show that if $x \in L$ and x is obtained without using rule 2 at all, then $x \in A$. The only possibility for x in this case is Λ, and $\Lambda \in A$ because $\Lambda = 0^0 1^0$.

Induction hypothesis. $k \geq 0$, and every string in L that can be obtained by k applications of rule 2 is an element of A.

Statement to show in induction step. Any string in L that can be obtained by $k + 1$ applications of rule 2 is in A.

Proof of induction step. Let x be an element of L that is obtained by $k + 1$ applications of rule 2. This means that either $x = 0y$ or $x = 0y1$, where in either case y is a string in L that can be obtained by using the rule k times. By the induction hypothesis, $y \in A$, so that $y = 0^i 1^j$, with $i \geq j \geq 0$. Therefore, either $x = 0^{i+1}1^j$ or $x = 0^{i+1}1^{j+1}$, and in either case $x \in A$.

As simple as this proof is, we can make it even simpler. There is a sense in which the integers k and $k+1$ are extraneous. In the induction step, we wish to show that the *new* element x of L, the one obtained by applying rule 2 of the definition to some other element y of L, is an element of A. It is true that y is obtained by applying the rule k times, and therefore x is obtained by applying the rule $k+1$ times. The only fact we need in the induction step, however, is that $y \in A$ and for *any* element y of L that is in A, both $0y$ and $0y1$ are also elements of A. Once we verify this, k and $k+1$ are needed only to make the proof fit the framework of a standard induction proof.

Why not just leave them out? We still have a basis step, except that instead of thinking of Λ as the string obtained from zero applications of rule 2, we just think of it as the string in rule 1, the basis part of the definition of L. In the recursive part of the definition, we apply one of two possible operations to an element y of L. What we will need to know about the string y in the induction step is that it is in A, and therefore it is appropriate to designate this as our induction hypothesis. The induction step is simply to show that for this string y, $0y$ and $0y1$ (the two strings obtained from y by applying the operations in the definition) are both in L.

We can call this version of mathematical induction *structural induction*. Although there is an underlying integer involved, just as in an ordinary induction proof, it is usually unnecessary to mention it explicitly. Instead, the steps of the proof follow the structure of the recursive definition directly. Below is our modified proof for this example.

To prove: $L \subseteq A$.

Basis step. We must show that $\Lambda \in A$. This is true, because $\Lambda = 0^0 1^0$.

Induction hypothesis. The string $y \in L$ is an element of A.

Statement to show in induction step. Both $0y$ and $0y1$ are elements of A.

Proof of induction step. Since $y \in A$, $y = 0^i 1^j$, with $i \geq j \geq 0$. Therefore, $0y = 0^{i+1}1^j$, and $0y1 = 0^{i+1}1^{j+1}$, and both strings are in A.

In the induction step, instead of talking about an arbitrary string x obtainable by $k+1$ applications of rule 2, the proof is more explicit in anticipating *how* it is obtained: It will be either $0y$ or $0y1$, where in either case y is a string obtainable by k applications of rule 2. In the original proof, this property of y is needed in order to be able to apply the induction hypothesis to y; in the structural induction proof, we anticipate the property of y that follows from the induction hypothesis, and simply take the induction hypothesis to be that y *does* satisfy this property.

Although we will not formulate an official Principle of Structural Induction, the preceding example and the ones to follow should make it clear how to use this technique. If we have a recursive definition of a set L, structural induction can be used to show that every element of L has some property. In the previous example, the "property" is that of belonging to the set A, and any property we are interested in can be expressed this way if we wish (we can always replace the phrase "has the property" by the phrase "belongs to the set of objects having the property").

Another Property of Fully Parenthesized Algebraic Expressions — EXAMPLE 2.21

Let us return to the language AE defined in Example 2.17. AE, a subset of $\Sigma^* = \{i, (,), +, -\}^*$, is defined as follows:

1. $i \in AE$.
2. For any x and y in AE, $(x + y)$ and $(x - y)$ are in AE.
3. No other strings are in AE.

This time we try to show that every element of the set has the property of not containing the substring)(. As in the previous example, we could set up an induction proof based on the number of times rule 2 is used in obtaining a string from the definition. Notice that in this approach we would want the strong principle of induction: If z is obtained by a total of $k+1$ applications of rule 2, then either $z = (x + y)$ or $z = (x - y)$, where in either case both x and y are obtained by k or *fewer* (not exactly k) applications of rule 2. However, in a proof by structural induction, since the integer k is not present explicitly, these details are unnecessary. What we really need to know about x and y is that they both have the desired property, and the statement that they do is the appropriate induction hypothesis.

To prove: No string in AE contains the substring)(.

Basis step. The string i does not contain the substring)(. (This is obvious.)

Induction hypothesis. x and y are strings that do not contain the substring)(.

Statement to show in induction step. Neither $(x + y)$ nor $(x - y)$ contains the substring)(.

Proof of induction step. In both the expressions $(x + y)$ and $(x - y)$, the symbol preceding x is not), the symbol following x is not (, the symbol preceding y is not), and the symbol following y is not (. Therefore, the only way)(could appear would be for it to occur in x or y separately.

Note that for the sake of simplicity, we made the induction hypothesis weaker than we really needed to (that is, we proved slightly more than was necessary). In order to use structural induction, we must show that if x and y are any strings in AE not containing)(, then neither $(x + y)$ nor $(x - y)$ contains)(. In our induction step, we showed this not only for x and y in AE, but for *any* x and y. This simplification is often, though not always, possible (see Exercise 2.69).

The Language of Strings with More *a*'s than *b*'s — EXAMPLE 2.22

Suppose that the language $L \subseteq \{a, b\}^*$ is defined as follows:

1. $a \in L$.
2. For any $x \in L$, $ax \in L$.
3. For any x and y in L, all the strings bxy, xby, and xyb are in L.
4. No other strings are in L.

Let us prove that every element of L has more a's than b's. Again we may use the structural induction principle, and just as in the previous example, we can simplify the induction step by

proving something even stronger than we need. We will show that

1. a has more a's than b's.
2. For any x having more a's than b's, ax also does.
3. For any x and y having more a's than b's, each of the strings bxy, xby, and xyb also does.

If we were using ordinary induction on the number of applications of steps 2 or 3 in the recursive definition of L, an appropriate induction hypothesis would be that any element of L obtainable by k or fewer applications has more a's than b's. Since we can anticipate that we will use this hypothesis either on a single string x to which we will apply step 2 or on two strings x and y to which we will apply step 3, we formulate our induction hypothesis to take care of either case.

To prove: Every element of L has more a's than b's.

Basis step. The string a has more a's than b's. (This is obvious.)

Induction hypothesis. x and y are strings containing more a's than b's.

Statement to show in induction step. Each of the strings ax, bxy, xby, and xyb has more a's than b's.

Proof of induction step. The string ax clearly has more a's than b's, since x does. Since both x and y have more a's than b's, xy has at least two more a's than b's, and therefore any string formed by inserting one more b still has at least one more a than b's.

In Exercise 2.64 you are asked to prove the converse, that every string in $\{a, b\}^*$ having more a's than b's is in the language L.

EXAMPLE 2.23 The Transitive Closure of a Relation

For any relation R on a set S, the *transitive closure* of R (see Exercise 1.63) is the relation R^t on S defined as follows:

1. $R \subseteq R^t$.
2. For any $x, y, z \in S$, if $(x, y) \in R^t$ and $(y, z) \in R^t$, then $(x, z) \in R^t$.
3. No other pairs are in R^t.

(It makes sense to summarize statements 1 to 3 by saying that R^t is the *smallest transitive relation containing* R.) Let us show that if R_1 and R_2 are relations on S with $R_1 \subseteq R_2$, then $R_1^t \subseteq R_2^t$.

Structural induction is appropriate here since the statement we want to show says that every pair in R_1^t satisfies the property of membership in R_2^t.

The basis step is to show that every element of R_1 is an element of R_2^t. This is true because $R_1 \subseteq R_2 \subseteq R_2^t$; the first inclusion is our assumption, and the second is just statement 1 in the definition of R_2^t. The induction hypothesis is that (x, y) and (y, z) are elements of R_1 that are in R_2^t, and in the induction step we must show that $(x, z) \in R_2^t$. Once again the argument is simplified slightly by proving more than we need to. For *any* pairs (x, y) and (y, z) in R_2^t, whether they are in R_1 or not, $(x, z) \in R_2^t$, because this is exactly what statement 2 in the definition of R_2^t says.

A recursive definition of a set can also provide a useful way of defining a function on the set. The function definition can follow the structure of the definition, in much the same way that a proof using structural induction does. This idea is illustrated by the next example.

EXAMPLE 2.24 Recursive Definitions of the Length and Reverse Functions

In Example 2.15 we saw a recursive definition of the set Σ^*, for an alphabet Σ:

1. $\Lambda \in \Sigma^*$.
2. For every $x \in \Sigma^*$, and every $a \in \Sigma$, $xa \in \Sigma^*$.
3. No other elements are in Σ^*.

Two useful functions on Σ^* are the length function, for which we have already given a nonrecursive definition, and the reverse function *rev*, which assigns to each string the string obtained by reversing the order of the symbols. Let us give a recursive definition of each of these. The length function can be defined as follows:

1. $|\Lambda| = 0$.
2. For any $x \in \Sigma^*$ and any $a \in \Sigma$, $|xa| = |x| + 1$.

For the reverse function we will often use the notation x^r to stand for $rev(x)$, the reverse of x. Here is one way of defining the function recursively.

1. $\Lambda^r = \Lambda$.
2. For any $x \in \Sigma^*$ and any $a \in \Sigma$, $(xa)^r = ax^r$.

To convince ourselves that these are both valid definitions of functions on Σ^*, we could construct a proof using structural induction to show that every element of Σ^* satisfies the property of being assigned a unique value by the function.

Let us now show, using structural induction, a useful property of the length function: For every x and y in Σ^*,

$$|xy| = |x| + |y|$$

(The structural induction, like the recursive definition of the length function itself, follows the recursive definition of Σ^* above.) The formula seems obvious, of course, from the way we normally think about the length function. The point is that we do not have to depend on our intuitive understanding of length; the recursive definition is a practical one to use in discussing properties of the function. For the proof we choose y as the string on which to base the induction (see Exercise 2.34). That is, we interpret the statement as a statement about y—namely, that for every x, $|xy| = |x| + |y|$. The basis step of the structural induction is to show that this statement is true when $y = \Lambda$. It is, because for every x,

$$|x\Lambda| = |x| = |x| + 0 = |x| + |\Lambda|$$

The induction hypothesis is that y is a string for which the statement holds, and in the induction step we consider ya, for an arbitrary $a \in \Sigma$. We want to show that for any x, $|x(ya)| = |x| + |ya|$:

$$\begin{aligned} |x(ya)| &= |(xy)a| \quad \text{(because concatenation is associative)} \\ &= |xy| + 1 \quad \text{(by the recursive definition of the length function)} \\ &= (|x| + |y|) + 1 \quad \text{(by the induction hypothesis)} \\ &= |x| + (|y| + 1) \quad \text{(because addition is associative)} \\ &= |x| + |ya| \quad \text{(by the definition of the length function)} \end{aligned}$$

In this case, structural induction is not much different from ordinary induction on the length of y, but a little simpler. The proof involves lengths of strings because we are proving a statement about lengths of strings; at least we did not have to introduce them gratuitously in order to provide a framework for an induction proof.

Finally, it is worth pointing out that the idea of structural induction is general enough to include the ordinary induction principle as a special case. When we prove that a statement $P(n)$ is true for every $n \geq n_0$, we are showing that every element of the set $S = \{n \mid n \geq n_0\}$ satisfies the property P. The set S can be defined recursively as follows:

1. $n_0 \in S$.
2. For every $n \in S$, $n + 1 \in S$.
3. No integer is in S unless it can be obtained from rules 1 and 2.

If we compare the induction step in an ordinary induction proof to that in a proof by structural induction, we see that they are the same. In the first case, the induction hypothesis is the statement that $P(k)$ is true for some $k \geq n_0$, and the induction step is to show that $P(k + 1)$ is true. In the second case, we assume that some element n of S satisfies the property P, and show that the element obtained from n by rule 2 (i.e., $n + 1$) also satisfies P.

EXERCISES

2.1. Prove that the statements $(p \vee q) \rightarrow r$ and $(p \rightarrow r) \vee (q \rightarrow r)$ are logically equivalent. (See Example 2.6.)

2.2. For each of Examples 2.5 and 2.6, how would you classify the given proof: constructive, nonconstructive, or something in-between? Why?

2.3. Prove that if a and b are even integers, then ab is even.

2.4. Prove that for any positive integers i, j, and n, if $i * j = n$, then either $i \geq \sqrt{n}$ or $j \geq \sqrt{n}$. (See Example 2.2. The statement in that example may be more obviously useful since it tells you that for an integer $n > 1$, if none of the integers j in the range $2 \leq j \leq \sqrt{n}$ is a divisor of n, then n is prime.)

2.5. In the induction step in Example 2.8, starting with $x = 0y1$, we used $y1$ as the string of length k to which we applied the induction hypothesis. Redo the induction step, this time using $0y$ instead.

2.6. Prove the statement in Example 2.8 by using mathematical induction on the number of 0's in the string, rather than on the length of the string.

2.7. In Example 2.12, in order to show that every nonempty subset of $\mathcal{N}$ has a smallest element, we chose $P(n)$ to be the statement: Every subset of $\mathcal{N}$ containing n has a smallest element. Consider this alternative choice for $P(n)$: Every subset of $\mathcal{N}$ containing at least n elements has a smallest element. (We would want to try to prove that this statement $P(n)$ is true for every $n \geq 1$.) Why would this *not* be an appropriate choice?

In all the remaining exercises in this chapter, with the exception of 46 through 48, "prove" means "prove, using an appropriate version of mathematical induction."

2.8. Prove that for every $n \geq 0$,

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

2.9. Suppose that $a_0, a_1, \ldots,$ is a sequence of real numbers. Prove that for any $n \geq 1$,

$$\sum_{i=1}^{n} (a_i - a_{i-1}) = a_n - a_0$$

2.10. Prove that for every $n \geq 1$,

$$7 + 13 + 19 + \cdots + (6n + 1) = n(3n + 4)$$

2.11. Prove that for every $n \geq 0$,

$$\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$$

2.12. For natural numbers n and i satisfying $0 \leq i \leq n$, let $C(n, i)$ denote the number $n!/(i!(n - i)!)$.

a. Show that if $0 < i < n$, then $C(n, i) = C(n - 1, i - 1) + C(n - 1, i)$. (You don't need mathematical induction for this.)

b. Prove that for any $n \geq 0$,

$$\sum_{i=0}^{n} C(n, i) = 2^n$$

2.13. Suppose r is a real number other than 1. Prove that for any $n \geq 0$,

$$\sum_{i=1}^{n} r^i = \frac{1 - r^{n+1}}{1 - r}$$

2.14. Prove that for any $n \geq 0$,

$$1 + \sum_{i=1}^{n} i * i! = (n+1)!$$

2.15. Prove that for any $n \geq 4$, $n! > 2^n$.

2.16. Prove that if $a_0, a_1, \ldots$ is a sequence of real numbers so that $a_n \leq a_{n+1}$ for every $n \geq 0$, then for every $m, n \geq 0$, if $m \leq n$, then $a_m \leq a_n$.

2.17. Suppose x is any real number greater than -1. Prove that for any $n \geq 0$, $(1+x)^n \geq 1 + nx$. (Be sure you say in your proof exactly how you use the assumption that $x > -1$.)

2.18. A fact about infinite series is that the series $\sum_{i=1}^{\infty} 1/i$ diverges (i.e., is infinite). Prove the following statement, which implies the result: For every $n \geq 1$, there is an integer $k_n \geq 1$ so that $\sum_{i=1}^{k_n} 1/i > n$. (Hint: the sum $1/(k+1) + 1/(k+2) + \cdots + 1/(2k)$ is at least how big?)

2.19. Prove that for every $n \geq 1$,

$$\sum_{i=1}^{n} i * 2^i = (n-1) * 2^{n+1} + 2$$

2.20. Prove that for every $n \geq 2$,

$$1 + \sum_{i=2}^{n} \frac{1}{\sqrt{i}} > \sqrt{n}$$

2.21. Prove that for every $n \geq 0$, n is either even or odd, but not both. (By definition, an integer n is even if there is an integer i so that $n = 2 * i$, and n is odd if there is an integer i so that $n = 2 * i + 1$.)

2.22. Prove that for any language $L \subseteq \{0,1\}^*$, if $L^2 \subseteq L$, then $L^+ \subseteq L$.

2.23. Suppose that Σ is an alphabet, and that $f : \Sigma^* \to \Sigma^*$ has the property that $f(a) = a$ for every $a \in \Sigma$ and $f(xy) = f(x)f(y)$ for every $x, y \in \Sigma^*$. Prove that for every $x \in \Sigma^*$, $f(x) = x$.

2.24. Prove that for every $n \geq 0$, $n(n^2+5)$ is divisible by 6.

2.25. Suppose a and b are integers with $0 \leq a < b$. Prove that for every $n \geq 1$, $b^n - a^n$ is divisible by $b - a$.

2.26. Prove that every positive integer is the product of a power of 2 and an odd integer.

2.27. Suppose that $A_1, A_2, \ldots$ are sets. Prove that for every $n \geq 1$,

$$\left(\bigcap_{i=1}^{n} A_i\right)' = \bigcup_{i=1}^{n} A_i'$$

2.28. Prove that for every $n \geq 1$, the number of subsets of $\{1, 2, \ldots, n\}$ is 2^n.

2.29. Prove that for every $n \geq 1$ and every $m \geq 1$, the number of functions from $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, m\}$ is m^n.

2.30. In calculus, a basic formula involving derivatives is the product formula, which says that if f and g are functions that have derivatives, $\frac{d}{dx}(f * g) = f * \frac{dg}{dx} + \frac{df}{dx} * g$. Using this formula and the fact that $\frac{dx}{dx} = 1$, prove that for any $n \geq 1$, $\frac{d}{dx}(x^n) = nx^{n-1}$.

2.31. The numbers a_n, for $n \geq 0$, are defined recursively as follows:

$$a_0 = -2; \quad a_1 = -2; \quad \text{for } n \geq 2,\ a_n = 5a_{n-1} - 6a_{n-2}$$

Prove that for every $n \geq 0$, $a_n = 2 * 3^n - 4 * 2^n$.

2.32. The Fibonacci function was defined in Example 2.13 using the definition

$$f_0 = 0; \quad f_1 = 1; \quad \text{for } n \geq 2,\ f_n = f_{n-1} + f_{n-2}$$

a. Suppose C is a positive real number satisfying $C > 8/13$. Prove that for every $n \geq 0$, $f_n < C(13/8)^n$.

b. Prove that for every $n \geq 0$, $\sum_{i=0}^{n} f_i^2 = f_n f_{n+1}$.

c. Prove that for every $n \geq 0$, $\sum_{i=0}^{n} f_i = f_{n+2} - 1$.

2.33. Suppose we define a real-valued function f on the natural numbers as follows:

$$f(0) = 0; \quad \text{for } n > 0,\ f(n) = \sqrt{1 + f(n-1)}$$

a. Prove that for every $n \geq 0$, $f(n) < 2$.

b. Prove that f is an increasing function; in other words, for every $n \geq 0$, $f(n+1) > f(n)$.

2.34. In Example 2.24, a proof was given using structural induction based on the string y that for any strings x and y, $|xy| = |x| + |y|$. Can you prove the result using structural induction based on x? Why or why not?

2.35. Prove that for any string x, $|x^r| = |x|$.

2.36. Prove that if x is any string in AE (see Example 2.17), then any prefix of x contains at least as many left parentheses as right.

2.37. Suppose we modify the definition of AE to remove the restriction that the expressions be fully parenthesized. We call the new language GAE. One way to define GAE is as follows.

(i) $i \in GAE$.

(ii) For any x and y in GAE, both the strings $x + y$ and $x - y$ are in GAE.

(iii) For any $x \in GAE$, the string (x) is in GAE.

(iv) No other strings are in GAE.

a. Prove that every string in AE is in GAE.

b. Prove that every prefix of every string in GAE has at least as many left parentheses as right.

c. Suppose that we define an integer-valued function N on the language GAE as follows. $N(i) = 0$; for any x and y in GAE, N assigns to both the strings $x + y$ and $x - y$ the larger of the two numbers $N(x)$ and $N(y)$; for any $x \in GAE$, N assigns the value $N(x) + 1$ to the string (x).

Describe in words (nonrecursively) what the value $N(x)$ means for a string $x \in GAE$.

2.38. Consider the following modified version of the strong induction principle, in which the basis step seems to have been eliminated.

To prove that the statement $P(n)$ is true for every $n \geq n_0$, it is sufficient to show that for any $k \geq n_0$, if $P(n)$ is true for every n satisfying $n_0 \leq n < k$, then $P(k)$ is true.

Assuming that the strong principle of mathematical induction is correct, prove that this modified version is correct. (Note that the basis step has not really been eliminated, only disguised.)

2.39. In each part below, a recursive definition is given of a subset of $\{a, b\}^*$. Give a simple nonrecursive definition in each case. Assume that each definition includes an implicit last statement: "Nothing is in L unless it can be obtained by the previous statements."

a. $a \in L$; for any $x \in L$, xa and xb are in L.
b. $a \in L$; for any $x \in L$, bx and xb are in L.
c. $a \in L$; for any $x \in L$, ax and xb are in L.
d. $a \in L$; for any $x \in L$, xb, xa, and bx are in L.
e. $a \in L$; for any $x \in L$, xb, ax, and bx are in L.
f. $a \in L$; for any $x \in L$, xb and xba are in L.

2.40. Give recursive definitions of each of the following sets.

a. The set $\mathcal{N}$ of all natural numbers.
b. The set S of all integers (positive and negative) divisible by 7.
c. The set T of positive integers divisible by 2 or 7.
d. The set U of all strings in $\{0, 1\}^*$ containing the substring 00.
e. The set V of all strings of the form $0^i 1^j$, where $j \leq i \leq 2j$.
f. The set W of all strings of the form $0^i 1^j$, where $i \geq 2j$.

2.41. Let L and A be the languages defined in Example 2.19. Prove that $L \subseteq A$ by using induction on the length of the string.

2.42. Below are recursive definitions of languages L_1 and L_2, both subsets of $\{a, b\}^*$. Prove that each is precisely the language L of all strings not containing the substring aa.

a. $\Lambda \in L_1$; $a \in L_1$;
For any $x \in L_1$, xb and xba are in L_1;
Nothing else is in L_1.

b. $\Lambda \in L_2$; $a \in L_2$;
For any $x \in L_2$, bx and abx are in L_2;
Nothing else is in L_2.

2.43. For each $n \geq 0$, we define the strings a_n and b_n in $\{0, 1\}^*$ as follows:

$$a_0 = 0; \quad b_0 = 1; \quad \text{for } n > 0,\ a_n = a_{n-1}b_{n-1};\ b_n = b_{n-1}a_{n-1}$$

Prove that for every $n \geq 0$, the following statements are true.

a. The strings a_n and b_n are of the same length.
b. The strings a_n and b_n differ in every position.
c. The strings a_{2n} and b_{2n} are palindromes.
d. The string a_n contains neither the substring 000 nor the substring 111.

2.44. The "pigeonhole principle" says that if $n + 1$ objects are distributed among n pigeonholes, there must be at least one pigeonhole that ends up with more than one object. A more formal version of the statement is that if $f : A \to B$ and the sets A and B have $n + 1$ and n elements, respectively, then f cannot be one-to-one. Prove the second version of the statement.

2.45. The following argument cannot be correct, because the conclusion is false. Say exactly which statement in the argument is the first incorrect one, and why it is incorrect.)

To prove: all circles have the same diameter. More precisely, for any $n \geq 1$, if S is any set of n circles, then all the elements of S have the same diameter. The basis step is to show that all the circles in a set of 1 circle have the same diameter, and this is obvious. The induction hypothesis is that $k \geq 1$ and for any set S of k circles, all elements of S have the same diameter. We wish to show that for this k, and any set S of $k+1$ circles, all the circles in S have the same diameter. Let $S = \{C_1, C_2, \ldots, C_{k+1}\}$. Consider the subsets $T = \{C_1, C_2, \ldots, C_k\}$ and $R = \{C_1, C_2, \ldots, C_{k-1}, C_{k+1}\}$. T is simply S with the circle C_{k+1} deleted, and R is S with the element C_{k-1} deleted. Since both T and R are sets of k circles, all the circles in T have the same diameter, and all the circles in R have the same diameter; these statements follow from the induction hypothesis. Now observe that C_{k-1} is an element of both T and R. If d is the diameter of this circle, then any circle in T has diameter d, and so does any circle in R. Therefore, all the circles in S have this same diameter.

MORE CHALLENGING PROBLEMS

2.46. Prove that if p and q are distinct primes, and the integer n is divisible by both p and q, then n is divisible by pq. You may use the following generally accepted fact from mathematics: If p is prime and m and n are positive integers so that mn is divisible by p, then either m is divisible by p or n is divisible by p.

2.47. Prove that if p and q are distinct primes, then pq is the smallest integer that is divisible by both p and q.

2.48. Prove that if n is any positive integer that is not a perfect square, then $\sqrt{n}$ is not rational. (See Example 2.3. You may need the generally accepted fact mentioned in Exercise 2.46.)

2.49. Prove that for every $n \geq 1$,

$$\sum_{i=1}^{n} \sqrt{i} > 2n\sqrt{n}/3$$

2.50. Prove that every integer greater than 17 is a nonnegative integer combination of 4 and 7. In other words, for every $n > 17$, there exist integers i_n and j_n, both ≥ 0, so that $n = i_n * 4 + j_n * 7$.

2.51. Prove that for every $n \geq 1$, the number of subsets of $\{1, 2, \ldots, n\}$ having an even number of elements is 2^{n-1}.

2.52. Prove that the ordinary principle of mathematical induction implies the strong principle of mathematical induction. In other words, show that if $P(n)$ is a statement involving n that we wish to establish for every $n \geq N$, and

1. The ordinary principle of mathematical induction is true;
2. $P(N)$ is true;
3. For every $k \geq N$, $(P(N) \wedge P(N+1) \wedge \ldots P(k)) \Rightarrow P(k+1)$

then $P(n)$ is true for every $n \geq N$.

2.53. Suppose that f is the Fibonacci function (see Example 2.13).

a. Prove that for every $n \geq 0$, $f_n = c(a^n - b^n)$, where

$$c = \frac{1}{\sqrt{5}}, \quad a = \frac{1+\sqrt{5}}{2}, \quad \text{and } b = \frac{1-\sqrt{5}}{2}$$

b. Prove that for every $n \geq 0$, $f_{n+1}^2 = f_n f_{n+2} + (-1)^n$.

2.54. Prove that every positive integer can be expressed uniquely as the sum of distinct powers of 2. (Another way to say this is that every positive integer has a unique binary representation. Note that there are really two things to show: first, that every positive integer can be expressed as the sum of distinct powers of 2; and second, that for every positive integer n, there cannot be two different sets of powers of 2, both of which sum to n.)

2.55. Prove that for any $n \geq 1$, and any sequence $a_1, a_2, \ldots, a_n$ of positive real numbers, and any sequence $b_1, b_2, \ldots, b_n$ that is a permutation (rearrangement) of the a_i's,

$$\frac{a_1}{b_1} + \frac{a_2}{b_2} + \cdots + \frac{a_n}{b_n} \geq n$$

and the two expressions are equal if and only if $b_i = a_i$ for every i.

2.56. Consider the following loop, written in pseudocode:

```
while B do
  S;
```

The meaning of this is what you would expect. Test B; if it is true, execute S; test B again; if it is still true, execute S again; test B again; and so forth. In other words, continue executing S as long as the condition B remains true. A condition P is called an *invariant* of the loop if whenever P and B are both true and S is executed once, P is still true.

a. Prove that if P is an invariant of the loop, and P is true before the first iteration of the loop (i.e., when B is tested the first time), then *if* the loop eventually terminates (i.e., after some number of iterations, B is false), P is still true.

b. Suppose x and y are integer variables, and initially $x \geq 0$ and $y > 0$. Consider the following program fragment:

```
q = 0;
r = x;
while r >= y do
  q = q + 1;
  r = r - y;
```

(The loop condition B is $r \geq y$, and the loop body S is the pair of assignment statements.) By considering the condition $(r \geq 0) \wedge (x = q * y + r)$, prove that when this loop terminates, the values of q and r will be the integer quotient and remainder, respectively, when x is divided by y; in other words, $x = q * y + r$ and $0 \leq r < y$.

2.57. Suppose f is a function defined on the set of positive integers and satisfying these two conditions:

(i) $f(1) = 1$

(ii) for $n \geq 1$, $f(2n) = f(2n+1) = 2f(n)$

Prove that for every positive integer n, $f(n)$ is the largest power of 2 less than or equal to n.

2.58. The total time $T(n)$ required to execute a particular recursive sorting algorithm on an array of n elements is one second if $n = 1$, and otherwise no more than $Cn + 2T(n/2)$ for some constant C independent of n. Prove that if n is any power of 2, say 2^k, then

$$T(n) \leq n * (Ck + 1) = n(C \log_2 n + 1)$$

2.59. The function *rev*: $\Sigma^* \to \Sigma^*$ was defined recursively in Example 2.24. Using the recursive definition, prove the following facts about *rev*. (Recall that $rev(x)$ is also written x^r.)

a. For any strings x and y in Σ^*, $(xy)^r = y^r x^r$.

b. For any string $x \in \Sigma^*$, $(x^r)^r = x$.

c. For any string $x \in \Sigma^*$ and any $n \geq 0$, $(x^n)^r = (x^r)^n$.

2.60. On the one hand, we have a recursive definition of the set *pal*, given in Example 2.16. On the other hand, we have a recursive definition of x^r, the reverse of the string x, in Example 2.24. Using these definitions, prove that $pal = \{x \in \Sigma^* \mid x^r = x\}$.

2.61. Prove that the language L defined by the recursive definition below is the set of all elements of $\{a, b\}^*$ not containing the substring aab.

$\Lambda \in L$;

For every $x \in L$, xa, bx, and abx are in L;

Nothing else is in L.

2.62. In each part below, a recursive definition is given of a subset of $\{a, b\}^*$. Give a simple nonrecursive definition in each case. Assume that each definition includes an implicit last statement: "Nothing is in L unless it can be obtained by the previous statements."

a. Λ, a, and aa are in L; for any $x \in L$, xb, xba, and $xbaa$ are in L.

b. $\Lambda \in L$; for any $x \in L$, ax, xb, and xba are in L.

2.63. Suppose $L \subseteq \{0, 1\}^*$. Two languages L_1 and L_2 are defined recursively below. (These two definitions were both given in Example 2.15 as possible definitions of L^*.)

Definition of L_1:

1. $\Lambda \in L_1$.

2. For any $x \in L_1$ and any $y \in L$, $xy \in L_1$.

3. No string is in L_1 unless it can be obtained by using rules 1 and 2.

Definition of L_2:

1. $\Lambda \in L_2$.

2. For any $x \in L$, $x \in L_2$.

3. For any two elements x and y of L_2, $xy \in L_2$.

4. No string is in L_2 unless it can be obtained by using rules 1, 2, and 3.

a. Prove that $L_1 \subseteq L_2$.

b. Prove that for any two strings x and y in L_1, $xy \in L_1$.

c. Prove that $L_2 \subseteq L_1$.

2.64. Let L be the language defined in Example 2.22. Prove that L contains every element of $\{a, b\}^*$ having more a's than b's.

2.65. $\Lambda \in L$; for every x and y in L, $axby$ and $bxay$ are both in L; nothing else is in L. Prove that L is precisely the set of strings in $\{a, b\}^*$ with equal numbers of a's and b's.

2.66. Suppose S and T are both finite sets of strings, $\Lambda \notin T$, and we have a function $e : S \to T$. The function e can be thought of as an *encoding* of the elements of S, using the elements of T as code words. In this situation we can then encode S^* by letting the code string for $x_1x_2 \ldots x_n$ be $e^*(x_1x_2 \ldots x_n) = e(x_1)e(x_2) \ldots e(x_n)$. The encoding e has the *prefix property* if there do not exist elements x_1 and x_2 of S so that $x_1 \neq x_2$ and $e(x_1)$ is a prefix of $e(x_2)$. (If e has the prefix property, then in particular e is one-to-one.) Prove that if e has the prefix property, then every element of T^* has at most one decoding—that is, the function e^* is one-to-one.

2.67. (Adapted from the book by Paulos [1998]). A certain remote village contains a large number of husband-wife couples. Exactly n of the husbands are unfaithful to their wives. Each wife is immediately aware of any *other* husband's infidelity and knows that the same is true of the other wives; however, she has no way of knowing whether her own husband has been unfaithful. (No wife ever informs on any other woman's husband.) The village also has a very strict code of morality; each wife follows the code rigidly and knows that the other wives do also. If any wife determines conclusively that her husband has been unfaithful, she must kill him on the same day she finds this out. At midnight each night, if anyone in the village has been killed that day, a public announcement is made, so that everyone then knows. Finally, all the wives in the village are expert reasoners, and each wife is aware that all the other wives are expert reasoners.

One day, a guru, whose pronouncements all the wives trust absolutely, visits the village, convenes a meeting of all the wives, and announces to them that there is at least one unfaithful husband in the village. What happens as a result? Prove your answer. (Hint: if $n = 1$, the wife of the unfaithful husband already knows that no other husband is unfaithful. She concludes that her husband is unfaithful and kills him that day.)

2.68. Suppose $P(m, n)$ is a statement involving two natural numbers m and n, and suppose we can show these two statements: i) $P(0, 0)$ is true; ii) For any natural numbers i and j, if $P(i, j)$ is true, then $P(i, j + 1)$ and $P(i + 1, j)$ are true. Does it follow that $P(m, n)$ is true for all natural numbers m and n? Give reasons for your answer.

2.69. Suppose S is a set of integers defined as follows: $0 \in S$; for every $x \in S$, $x + 5 \in S$; no other elements are in S. In order to show using structural induction that every element of S satisfies some property P, it is enough to show that 0 satisfies P, and if $x \in S$ and x satisfies P, then $x + 5$ also satisfies P. Give an example of a property P for which this is true but for which it is not true that $x + 5$ satisfies P for every integer x satisfying P. In other words, give an example in which the structural induction proof cannot be simplified as we did in Examples 2.21 and 2.23.

2.70. Suppose that U is a *finite* set that is closed under some binary operation $\circ$, and I is a subset of U. Suppose also that S is defined as follows:

1. Every element of I is an element of S.
2. For every x and y in S, $x \circ y \in S$.
3. Nothing else is in S.

Describe an algorithm that will determine, for some arbitrary element of U, whether or not it is in S. (In particular, for each element of U, no matter whether the answer is yes or no for that element, the algorithm must produce the answer after a finite number of steps.)