chapter 4

Equivalence, Order, and Inductive Proof

Good order is the foundation of all things.
—Edmund Burke (1729–1797)

Classifying things and ordering things are activities in which we all engage from time to time. Whenever we classify or order a set of things, we usually compare them in some way. That's how binary relations enter the picture.

In this chapter we'll discuss some special properties of binary relations that are useful for solving comparison problems. We'll introduce techniques to construct binary relations with the properties that we need. We'll discuss the idea of equivalence by considering properties of the equality relation. We'll also study the properties of binary relations that characterize our intuitive ideas about ordering. We'll also see that ordering is the fundamental ingredient needed to discuss inductive proof techniques.

chapter guide

4.1 (Properties of Binary Relations) introduces some of the desired properties of binary relations and shows how to construct new relations by composition and closure. We'll see how the results apply to solving path problems in graphs.

4.2 (Equivalence Relations) concentrates on the idea of equivalence. We'll see that equivalence is closely related to partitioning of sets. We'll show how to generate equivalence relations, we'll solve a typical equivalence problem, and we'll see an application for finding a spanning tree for a graph.

4.3 (Order Relations) introduces the idea of order. We'll discuss partial orders and how to sort them. We'll introduce well-founded orders and show some techniques for constructing them. Ordinal numbers are also introduced.

4.4 (Inductive Proof) introduces inductive proof techniques. We'll discuss the technique of mathematical induction for proving statements indexed by the

natural numbers. Then we'll extend the discussion to inductive proof techniques for any well-founded set.

4.1 Properties of Binary Relations

Recall that the statement "R is a binary relation on the set A" means that R relates certain pairs of elements of A. Thus R can be represented as a set of ordered pairs (x, y), where $x, y \in A$. In other words, R is a subset of the Cartesian product $A \times A$. When $(x, y) \in R$, we also write $x \; R \; y$.

Binary relations that satisfy certain special properties can be very useful in solving computational problems. So let's discuss these properties.

Three Special Properties

For a binary relation R on a set A, we have the following definitions.

a. R is *reflexive* if $x \; R \; x$ for all $x \in A$.

b. R is *symmetric* if $x \; R \; y$ implies $y \; R \; x$ for all $x, y \in A$.

c. R is *transitive* if $x \; R \; y$ and $y \; R \; z$ implies $x \; R \; z$ for all $x, y, z \in A$.

Since a binary relation can be represented by a directed graph, we can describe the three properties in terms of edges: R is reflexive if there is an edge from x to x for each $x \in A$; R is symmetric if for each edge from x to y, there is also an edge from y to x. R is transitive if whenever there are edges from x to y and from y to z, there must also be an edge from x to z.

There are two more properties that we will also find to be useful.

Two More Properties

For a binary relation R on a set A, we have the following definitions.

a. R is *irreflexive* if $(x, x) \notin R$ for all $x \in A$.

b. R is *antisymmetric* if $x \; R \; y$ and $y \; R \; x$ implies $x = y$ for all $x, y \in A$.

From a graphical point of view we can say that R is irreflexive if there are no loop edges from x to x for all $x \in A$; and R is antisymmetric if whenever there is an edge from x to y with $x \neq y$, then there is no edge from y to x.

Many well-known relations satisfy one or more of the properties that we've been discussing. So we better look at a few examples.

example 4.1 Five Binary Relations

Here are some sample binary relations with the properties that they satisfy.

a. The equality relation on any set is reflexive, symmetric, transitive, and antisymmetric.

b. The $<$ relation on real numbers is transitive, irreflexive, and antisymmetric.

c. The $\leq$ relation on real numbers is reflexive, transitive, and antisymmetric.

d. The "is parent of" relation is irreflexive and antisymmetric.

e. The "has the same birthday as" relation is reflexive, symmetric, and transitive.

end example

4.1.1 Composition of Relations

Relations can often be defined in terms of other relations. For example, we can describe the "is grandparent of" relation in terms of the "is parent of" relation by saying that "a is grandparent of c" if and only if there is some b such that "a is parent of b" and "b is parent of c". This example demonstrates the fundamental idea of composing binary relations.

Definition of Composition

If R and S are binary relations, then the *composition* of R and S, which we denote by $R \circ S$, is the following relation:

$$R \circ S = \{(a, c) \mid (a, b) \in R \text{ and } (b, c) \in S \text{ for some element } b\}.$$

From a directed graph point of view, if we find an edge from a to b in the graph of R and we find an edge from b to c in the graph of S, then we must have an edge from a to c in the graph of $R \circ S$.

example 4.2 Grandparents

To construct the "isGrandparentOf" relation we can compose "isParentOf" with itself.

$$\text{isGrandparentOf} = \text{isParentOf} \circ \text{isParentOf}.$$

Similarly, we can construct the "isGreatGrandparentOf" relation by the following composition:

$$\text{isGreatGrandparentOf} = \text{isGrandparentOf} \circ \text{isParentOf}.$$

end example

example 4.3 Numeric Relations

Suppose we consider the relations "less," "greater," "equal," and "notEqual" over the set $\mathbb{R}$ of real numbers. We want to compose some of these relations to see what we get. For example, let's verify the following equality.

$$\text{greater} \circ \text{less} = \mathbb{R} \times \mathbb{R}.$$

For any pair (x, y), the definition of composition says that x (greater $\circ$ less) y if and only if there is some number z such that x greater z and z less y. We can write this statement more concisely as follows:

x $(> \circ <)$ y iff there is some number z such that $x > z$ and $z < y$.

We know that for any two real numbers x and y there is always another number z that is less than both. So the composition must be the whole universe $\mathbb{R} \times \mathbb{R}$. Many combinations are possible. For example, it's easy to verify the following two equalities:

$$\begin{aligned} \text{equal} \circ \text{notEqual} &= \text{notEqual}, \\ \text{notEqual} \circ \text{notEqual} &= \mathbb{R} \times \mathbb{R}. \end{aligned}$$

end example

Other Combining Methods

Since relations are just sets (of ordered pairs), they can also be combined by the usual set operations of union, intersection, difference, and complement.

example 4.4 Combining Relations

The following samples show how we can combine some familiar numeric relations. Check out each one with a few example pairs of numbers.

$$\begin{aligned} \text{equal} \cap \text{less} &= \varnothing, \\ \text{equal} \cap \text{lessOrEqual} &= \text{equal}, \\ (\text{lessOrEqual})' &= \text{greater}, \\ \text{greaterOrEqual} - \text{equal} &= \text{greater}, \\ \text{equal} \cup \text{greater} &= \text{greaterOrEqual}, \\ \text{less} \cup \text{greater} &= \text{notEqual}. \end{aligned}$$

end example

Let's list some fundamental properties of combining relations. We'll leave the proofs of these properties as exercises.

Properties of Combining Relations (4.1)

a. $R \circ (S \circ T) = (R \circ S) \circ T$. (associativity)

b. $R \circ (S \cup T) = R \circ S \cup R \circ T$.

c. $R \circ (S \cap T) \subseteq R \circ S \cap R \circ T$.

Notice that Part (c) is stated as a set containment rather than an equality. For example, let R, S, and T be the following relations:

$$R = \{(a, b), (a, c)\}, S = \{(b, b)\}, T = \{(b, c), (c, b)\}.$$

Then $S \cap T = \varnothing$, $R \circ S = \{(a, b)\}$, and $R \circ T = \{(a, c), (a, b)\}$. Therefore

$$R \circ (S \cap T) = \varnothing \text{ and } R \circ S \cap R \circ T = \{(a, b)\}.$$

So (4.1c) isn't always an equality. But there are cases in which equality holds. For example, if $R = \varnothing$ or if $S \subseteq T$, then (4.1c) is an equality.

Representations

If R is a binary relation on A, then we'll denote the composition of R with itself n times by writing

$$R^n.$$

For example, if we compose isParentOf with itself, we get some familiar names as follows:

$$\text{isParentOf}^2 = \text{isGrandparentOf},$$
$$\text{isParentOf}^3 = \text{isGreatGrandparentOf}.$$

We mentioned in Chapter 1 that binary relations can be thought of as digraphs and, conversely, that digraphs can be thought of as binary relations. In other words, we can think of (x, y) as an edge from x to y in a digraph and as a member of a binary relation. So we can talk about the digraph of a binary relation.

An important and useful representation of R^n is as the digraph consisting of all edges (x, y) such that there is a path of length n from x to y. For example, if $(x, y) \in R^2$, then $(x, z), (z, y) \in R$ for some element z. This says that there is a path of length 2 from x to y in the digraph of R.

Example 4.5 Compositions

Let $R = \{(a, b), (b, c), (c, d)\}$. The digraphs shown in Figure 4.1 are the digraphs for the three relations R, R^2, and R^3.

end example

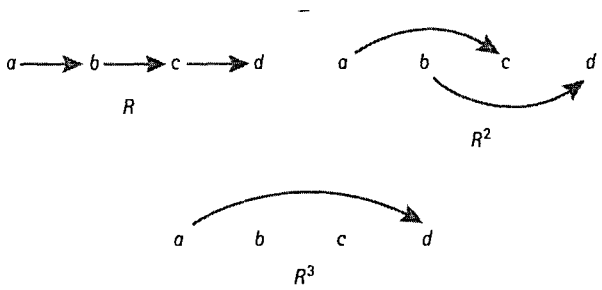


Figure 4.1 Composing a relation.

Let's give a more precise definition of R^n using induction. (Notice the interesting choice for R^0.)

$$R^0 = \{(a, a) \mid a \in A\} \quad \text{(basic equality)}$$
$$R^{n+1} = R^n \circ R.$$

We defined R^0 as the basic equality relation because we want to infer the equality $R^1 = R$ from the definition. To see this, observe the following evaluation of R^1:

$$R^1 = R^{0+1} = R^0 \circ R = \{(a, a) \mid a \in A\} \circ R = R.$$

We also could have defined $R^{n+1} = R \circ R^n$ instead of $R^{n+1} = R^n \circ R$ because composition of binary operations is associative by (4.1a).

Let's note a few other interesting relationships between R and R^n.

Inheritance Properties (4.2)

a. If R is reflexive, then R^n is reflexive.

b. If R is symmetric, then R^n is symmetric.

c. If R is transitive, then R^n is transitive.

On the other hand, if R is irreflexive, then it may not be the case that R^n is irreflexive. Similarly, if R is antisymmetric, it may not be the case that R^n is antisymmetric. We'll examine these statements in the exercises.

example 4.6 Integer Relations

Let $R = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid x + y \text{ is odd}\}$. We'll calculate R^2 and R^3. To calculate R^2, we'll examine an arbitrary element $(x, y) \in R^2$. This means there is an element z such that $(x, z) \in R$ and $(z, y) \in R$. So $x + z$ is odd and $z + y$ is odd. We know that a sum is odd if and only if one number is even and the other number is odd. If x is even, then since $x + z$ is odd, it follows that z is

odd. So, since $z + y$ is odd, it follows that y is even. Similarly, if x is odd, the same kind of reasoning shows that y is odd. So we have

$$R^2 = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid x \text{ and } y \text{ are both even or both odd}\}.$$

To calculate R^3, we'll examine an arbitrary element $(x, y) \in R^3$. This means there is an element z such that $(x, z) \in R$ and $(z, y) \in R^2$. In other words, $x + z$ is odd and z and y are both even or both odd. If x is even, then since $x + z$ is odd, it follows that z is odd. So y must be odd. Similarly, if x is odd, the same kind of reasoning shows that y is even. So if $(x, y) \in R^3$, then one of x and y is even and the other is odd. In other words, $x + y$ is odd. Therefore

$$R^3 = R.$$

We don't have to go to higher powers now because, for example,

$$R^4 = R^3 \circ R = R \circ R = R^2.$$

end example

4.1.2 Closures

We've seen how to construct a new binary relation by composing two existing binary relations. Now we'll see how to construct a new binary relation by adding pairs of elements to an existing binary relation so that the new relation satisfies some particular property. For example, from the "isParentOf" relation for a family, we could construct the "isAncestorOf" relation for the family by adding the isGrandParentOf pairs, the isGreatGrandParentOf pairs, and so on. But for some properties this process cannot always be done. For example, if $R = \{(a, b), (b, a)\}$, then R is symmetric, so it cannot be a subset of any antisymmetric relation. To discuss this further we need to introduce the idea of closure.

Definition of Closure

If R is a binary relation and p is a property that can be satisfied by adding pairs of elements to R, then the *p closure* of R is the smallest binary relation containing R that has property p.

Three properties of interest for which closures always exist are reflexive, symmetric, and transitive. To introduce each of the three closures we'll use the following relation on the set $A = \{a, b, c\}$:

$$R = \{(a, a), (a, b), (b, a), (b, c)\}.$$

Notice that R is not reflexive, not symmetric, and not transitive. So the closures of R that we construct will all contain R as a proper subset.

Reflexive Closure

If R is a binary relation on A, then the *reflexive closure* of R, which we'll denote by $r(R)$, can be constructed by including all pairs (x, x) that are not already in R. Recall that the relation $\{(x, x) \mid x \in A\}$ is called the equality relation on A and it is also denoted by R^0. So we can say that

$$r(R) = R \cup R^0.$$

In our example, the pairs (b, b) and (c, c) are missing from R. So $r(R)$ is R together with these two pairs.

$$r(R) = \{(a, a), (a, b), (b, a), (b, c), (b, b), (c, c)\}.$$

Symmetric Closure

If R is a binary relation, then the *symmetric closure* of R, which we'll denote by $s(R)$, must include all pairs (x, y) for which $(y, x) \in R$. The set $\{(x, y) \mid (y, x) \in R\}$ is called the *converse* of R, which we'll denote by R^c. So we can say that

$$s(R) = R \cup R^c.$$

Notice that R is symmetric if and only if $R = R^c$.

In our example, the only problem is with the pair $(b, c) \in R$. Once we include the pair (c, b) we'll have $s(R)$.

$$s(R) = \{(a, a), (a, b), (b, a), (b, c), (c, b)\}.$$

Transitive Closure

If R is a binary relation, then the *transitive closure* of R is denoted by $t(R)$. We'll use our example to show how to construct $t(R)$. Notice that R contains the pairs (a, b) and (b, c), but (a, c) is not in R. Similarly, R contains the pairs (b, a) and (a, b), but (b, b) is not in R. So $t(R)$ must contain the pairs (a, c) and (b, b). Is there some relation that we can union with R that will add the two needed pairs? The answer is yes, it's R^2. Notice that

$$R^2 = \{(a, a), (a, b), (b, a), (b, b), (a, c)\}.$$

It contains the two missing pairs along with three other pairs that are already in R. Thus we have

$$t(R) = R \cup R^2 = \{(a, a), (a, b), (b, a), (b, c), (a, c), (b, b)\}.$$

To get some further insight into constructing the transitive closure, we need to look at another example. Let $A = \{a, b, c, d\}$, and let R be the following relation.

$$R = \{(a, b), (b, c), (c, d)\}.$$

To compute $t(R)$, we need to add the three pairs (a, c), (b, d), and (a, d). In this case, $R^2 = \{(a, c), (b, d)\}$. So the union of R with R^2 is missing (a, d).

Can we find another relation to union with R and R^2 that will add this missing pair? Notice that $R^3 = \{(a, d)\}$. So for this example, $t(R)$ is the union

$$\begin{aligned} t(R) &= R \cup R^2 \cup R^3 \\ &= \{(a,b), (b,c), (c,d), (a,c), (b,d), (a,d)\}. \end{aligned}$$

As the examples show, $t(R)$ is a bit more difficult to construct than the other two closures.

Constructing the Three Closures

The three closures can be calculated by using composition and union. Here are the construction techniques.

Constructing Closures (4.3)

If R is a binary relation over a set A, then:

a. $r(R) = R \cup R^0$ (R^0 is the equality relation.)

b. $s(R) = R \cup R^c$ (R^c is the converse relation.)

c. $t(R) = R \cup R^2 \cup R^3 \cup \cdots$.

d. If A is finite with n elements, then $t(R) = R \cup R^2 \cup \cdots \cup R^n$.

Let's discuss (4.3d). If $(x, y) \in R^m$, then there is a path of length m from x to y in the digraph representation of R. If $m > n$, then some element of A occurs at least twice on the path from x to y. So there is a shorter path from x to y. Thus $(x, y) \in R^k$ for some $k < m$. This argument can be repeated, if necessary, until we find that $(x, y) \in R^k$ for some $k \leq n$. So nothing new gets added to $t(R)$ by adding powers of R that are higher than n.

Sometimes we don't have to compute all the powers of R. For example, let $A = \{a, b, c, d, e\}$ and $R = \{(a, b), (b, c), (b, d), (d, e)\}$. The digraphs of R and $t(R)$ are drawn in Figure 4.2. Convince yourself that $t(R) = R \cup R^2 \cup R^3$. In other words, the relations R^4 and R^5 don't add anything new. In fact, you should verify that $R^4 = R^5 = \varnothing$.

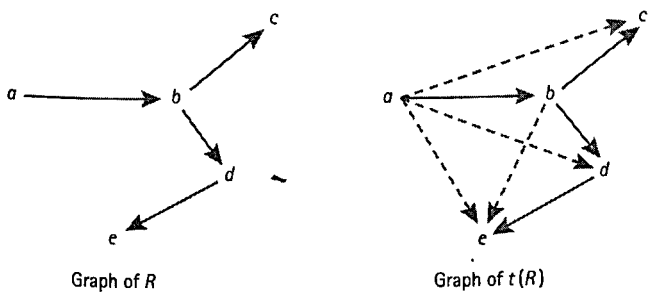


Figure 4.2 R and its transitive closure.

example 4.7 A Big Transitive Closure

Let $A = \{a, b, c\}$ and $R = \{(a, b), (b, c), (c, a)\}$. Then we have

$$R^2 = \{(a, c), (c, b), (b, a)\} \text{ and } R^3 = \{(a, a), (b, b), (c, c)\}.$$

So the transitive closure of R is the union

$$t(R) = R \cup R^2 \cup R^3 = A \times A.$$

end example

example 4.8 A Small Transitive Closure

Let $A = \{a, b, c\}$ and $R = \{(a, b), (b, c), (c, b)\}$. Then we have

$$R^2 = \{(a, c), (b, b), (c, c)\} \text{ and } R^3 = \{(a, b), (b, c), (c, b)\} = R.$$

So the transitive closure of R is the union of the sets, which gives

$$t(R) = \{(a, b), (b, c), (c, b), (a, c), (b, b), (c, c)\}.$$

end example

example 4.9 Generating Less-Than

Suppose $R = \{(x, x + 1) \mid x \in \mathbb{N}\}$. Then $R^2 = \{(x, x + 2) \mid x \in \mathbb{N}\}$. In general, for any natural number $k > 0$ we have

$$R^k = \{(x, x + k) \mid x \in \mathbb{N}\}.$$

Since $t(R)$ is the union of all these sets, it follows that $t(R)$ is the familiar "less" relation over $\mathbb{N}$. Just notice that if $x < y$, then $y = x + k$ for some k, so the pair (x, y) is in R^k.

end example

example 4.10 Closures of Numeric Relations

We'll list some closures for the numeric relations "less" and "notEqual" over the set $\mathbb{N}$ of natural numbers.

$$\begin{aligned} r\,(\text{less}) &= \text{lessOrEqual}, \\ s\,(\text{less}) &= \text{notEqual}, \\ t\,(\text{less}) &= \text{less}, \\ r\,(\text{notEqual}) &= \mathbb{N} \times \mathbb{N}, \\ s\,(\text{notEqual}) &= \text{notEqual}, \\ t\,(\text{notEqual}) &= \mathbb{N} \times \mathbb{N}. \end{aligned}$$

end example

Properties of Closures

Some properties are retained by closures. For example, we have the following results, which we'll leave as exercises:

Inheritance Properties (4.4)

a. If R is reflexive, then $s(R)$ and $t(R)$ are reflexive.

b. If R is symmetric, then $r(R)$ and $t(R)$ are symmetric.

c. If R is transitive, then $r(R)$ is transitive.

Notice that (4.4c) doesn't include the statement "$s(R)$ is transitive" in its conclusion. To see why, we can let $R = \{(a, b), (b, c), (a, c)\}$. It follows that R is transitive. But $s(R)$ is not transitive because, for example, we have (a, b), $(b, a) \in s(R)$ and $(a, a) \notin s(R)$.

Sometimes, it's possible to take two closures of a relation and not worry about the order. Other times, we have to worry. For example, we might be interested in the double closure $r(s(R))$, which we'll denote by $rs(R)$. Do we get the same relation if we interchange r and s and compute $sr(R)$? The inheritance properties (4.4) should help us see that the answer is yes. Here are the facts:

Double Closure Properties (4.5)

a. $rt(R) = tr(R)$.

b. $rs(R) = sr(R)$.

c. $st(R) \subseteq ts(R)$.

Notice that (4.5c) is not an equality. To see why, let $A = \{a, b, c\}$, and consider the relation $R = \{(a, b), (b, c)\}$. Then $st(R)$ and $ts(R)$ are

$$st(R) = \{(a,b), (b,a), (b,c), (c,b), (a,c), (c,a)\}.$$
$$ts(R) = A \times A.$$

Therefore, $st(R)$ is a proper subset of $ts(R)$. Of course, there are also situations in which $st(R) = ts(R)$. For example, if $R = \{(a,a), (b,b), (c,c), (a,b), (b,c)\}$, then you can verify that $st(R) = ts(R)$.

Before we finish this discussion of closures, we should remark that the symbols R^+ and R^* are often used to denote the closures $t(R)$ and $rt(R)$.

4.1.3 Path Problems

Suppose we need to write a program that inputs two points in a city and outputs a bus route between the two points. A solution to the problem depends on the definition of "point." For example, if a point is any street intersection, then the solution may be harder than in the case in which a point is a bus stop.

This problem is an instance of a path problem. Let's consider some typical path problems in terms of a digraph.

Some Path Problems (4.6)

Given a digraph and two of its vertices i and j.

a. Find out whether there is a path from i to j. For example, find out whether there is a bus route from i to j.

b. Find a path from i to j. For example, find a bus route from i to j.

c. Find a path from i to j with the minimum number of edges. For example, find a bus route from i to j with the minimum number of stops.

d. Find a shortest path from i to j, where each edge has a nonnegative weight. For example, find the shortest bus route from i to j, where shortest might refer to distance or time.

e. Find the length of a shortest path from i to j. For example, find the number of stops (or the time or miles) on the shortest bus route from i to j.

Each problem listed in (4.6) can be phrased as a question and the same question is often asked over and over again (e.g., different people asking about the same bus route). So it makes sense to get the answers in advance if possible. We'll see how to solve each of the problems in (4.6).

Adjacency Matrix

A useful way to represent a binary relation R over a finite set A (equivalently, a digraph with vertices A and edges R) is as a special kind of matrix called an *adjacency matrix* (or incidence matrix). For ease of notation we'll assume that $A = \{1,..., n\}$ for some n. The adjacency matrix for R is an n by n matrix M with entries defined as follows:

$$M_{ij} = \text{if } (i, j) \in R \text{ then } 1 \text{ else } 0.$$

example 4.11 An Adjacency Matrix

Consider the relation $R = \{(1, 2), (2, 3), (3, 4), (4, 3)\}$ over $A = \{1, 2, 3, 4\}$. We can represent R as a directed graph or as an adjacency matrix M. Figure 4.3 shows the two representations.

end example

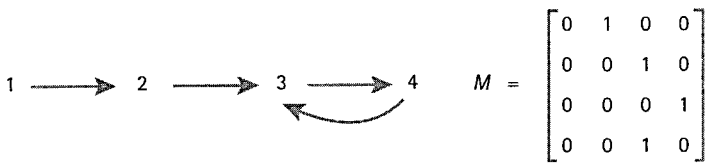


Figure 4.3 Directed graph and adjacency matrix.

If we look at the digraph in Figure 4.3, it's easy to see that R is neither reflexive, symmetric, nor transitive. We can see from the matrix M in Figure 4.3 that R is not reflexive because there is at least one zero on the main diagonal formed by the elements M_{ii}. Similarly, R is not symmetric because a reflection on the main diagonal is not the same as the original matrix. In other words, there are indices i and j such that $M_{ij} \neq M_{ji}$. R is not transitive, but there isn't any visual pattern in M that corresponds to transitivity.

It's an easy task to construct the adjacency matrix for $r(R)$: Just place 1's on the main diagonal of the adjacency matrix. It's also an easy task to construct the adjacency matrix for $s(R)$. We'll leave this one as an exercise.

Warshall's Algorithm for Transitive Closure

Let's look at an interesting algorithm to construct the adjacency matrix for $t(R)$. The idea, of course, is to repeat the following process until no new edges can be added to the adjacency matrix: If (i, k) and (k, j) are edges, then construct a new edge (i, j). The following algorithm to accomplish this feat with three **for**-loops is due to Warshall [1962].

Warshall's Algorithm for Transitive Closure (4.7)

Let M be the adjacency matrix for a relation R over $\{1,..., n\}$. The algorithm replaces M with the adjacency matrix for $t(R)$.

```
for k := 1 to n do
    for i := 1 to n do
        for j := 1 to n do
            if (M_ik = M_kj = 1) then M_ij := 1
od od od
```

example **4.12 Applying Warshall's Algorithm**

We'll apply Warshall's algorithm to find the transitive closure of the relation R given in Example 4.11. So the input to the algorithm will be the adjacency matrix M for R shown in Figure 4.3. The four matrices in Figure 4.4 show how

$$M \rightarrow \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \xrightarrow{k=1} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \xrightarrow{k=2} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \xrightarrow{k=3} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \xrightarrow{k=4}$$

Figure 4.4 Matrix transformations via Warshall's algorithm.

Warshall's algorithm transforms M into the adjacency matrix for $t(R)$. Each matrix represents the value of M for the given value of k after the inner i and j loops have executed. To get some insight into how Warshall's algorithm works, draw the four digraphs for the adjacency matrices in Figure 4.4.

end example

Now we have an easy way find out whether there is a path from i to j in a digraph. Let R be the set of edges in the digraph. First we represent R as an adjacency matrix. Then we apply Warshall's algorithm to construct the adjacency matrix for $t(R)$. Now we can check to see whether there is a path from i to j in the original digraph by checking M_{ij} in the adjacency matrix M for $t(R)$. So we have all the solutions to problem (4.6a).

Floyd's Algorithm for Length of Shortest Path

Let's look at problem (4.6e). Can we compute the length of a shortest path in a weighted digraph? Sure. Let R denote the set of edges in the digraph. We'll represent the digraph as a *weighted adjacency matrix* M as follows: First of all, we set $M_{ii} = 0$ for $1 \leq i \leq n$ because we're not interested in the shortest path from i to itself. Next, for each edge $(i, j) \in R$ with $i \neq j$, we set M_{ij} to be the nonnegative weight for that edge. Lastly, if $(i, j) \notin R$ with $i \neq j$, then we set $M_{ij} = \infty$, where ∞ represents some number that is larger than the sum of all the weights on all the edges of the digraph.

example **4.13 A Weighted Adjacency Matrix**

The diagram in Figure 4.5 represents the weighted adjacency matrix M for a weighted digraph over the vertex set $\{1, 2, 3, 4, 5, 6\}$.

end example

Now we can present an algorithm to compute the shortest distances between vertices in a weighted digraph. The algorithm, due to Floyd [1962], modifies

	1	2	3	4	5	6
1	0	10	10	∞	20	10
2	∞	0	∞	30	∞	∞
3	∞	∞	0	30	∞	∞
4	∞	∞	∞	0	∞	∞
5	∞	∞	∞	40	0	∞
6	∞	∞	∞	∞	5	0

Figure 4.5 Sample weighted adjacency matrix.

the weighted adjacency matrix M so that M_{ij} is the shortest distance between distinct vertices i and j. For example, if there are two paths from i to j, then the entry M_{ij} denotes the smaller of the two path weights. So again, transitive closure comes into play. Here's the algorithm.

Floyd's Algorithm for Shortest Distances (4.8)

Let M be the weighted adjacency matrix for a weighted digraph over the set $\{1,..., n\}$. The algorithm replaces M with a weighted adjacency matrix that represents the shortest distances between distinct vertices.

```
for k := 1 to n do
    for i := 1 to n do
        for j := 1 to n do
            M_ij := min{M_ij, M_ik + M_kj}
od od od
```

example **4.14 Applying Floyd's Algorithm**

We'll apply Floyd's algorithm to the weighted adjacency matrix in Figure 4.5. The result is given in Figure 4.6. The entries M_{ij} that are not zero and not ∞ represent the minimum distances (weights) required to travel from i to j in the original digraph.

end example

Let's summarize our results so far. Algorithm (4.8) creates a matrix M that allows us to easily answer two questions: Is there a path from i to j for distinct vertices i and j? Yes, if $M_{ij} \neq \infty$. What is the distance of a shortest path from i to j? It's M_{ij} if $M_{ij} \neq \infty$.

	1	2	3	4	5	6
1	0	10	10	40	15	10
2	∞	0	∞	30	∞	∞
3	∞	∞	0	30	∞	∞
4	∞	∞	∞	0	∞	∞
5	∞	∞	∞	40	0	∞
6	∞	∞	∞	45	5	0

Figure 4.6 The result of Floyd's algorithm.

Floyd's Algorithm for Finding the Shortest Path

Now let's try to find a shortest path. We can make a slight modification to (4.8) to compute a "path" matrix P, which will hold the key to finding a shortest path. We'll initialize P to be all zeros. The algorithm will modify P so that $P_{ij} = 0$ means that the shortest path from i to j is the edge from i to j and $P_{ij} = k$ means that a shortest path from i to j goes through k. The modified algorithm, which computes M and P, is stated as follows:

Shortest Distances and Shortest Paths Algorithm **(4.9)**

Let M be the weighted adjacency matrix for a weighted digraph over the set $\{1,..., n\}$. Let P be the n by n matrix of zeros. The algorithm replaces M by a matrix of shortest distances and it replaces P by a path matrix.

```
for k := 1 to n do
    for i := 1 to n do
        for j := 1 to n do
            if M_ik + M_kj < M_ij then
                M_ij := M_ik + M_kj ;
                P_ij := k
od od od fi
```

example 4.15 The Path Matrix

We'll apply (4.9) to the weighted adjacency matrix in Figure 4.5. The algorithm produces the matrix M in Figure 4.6, and it produces the path matrix P given in Figure 4.7.

For example, the shortest path between 1 and 4 passes through 2 because $P_{14} = 2$. Since $P_{12} = 0$ and $P_{24} = 0$, the shortest path between 1 and 4 consists of the sequence 1, 2, 4. Similarly, the shortest path between 1 and 5 is the sequence 1, 6, 5, and the shortest path between 6 and 4 is the sequence 6, 5, 4. So once

	1	2	3	4	5	6
1	0	0	0	2	6	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	5	0	0

Figure 4.7 A path matrix.

we have matrix P from (4.9), it's an easy matter to compute a shortest path between two points. We'll leave this as an exercise.

end example

Let's make a few observations about Example 4.15. We should note that there is another shortest path from 1 to 4, namely, 1, 3, 4. The algorithm picked 2 as the intermediate point of the shortest path because the outer index k increments from 1 to n. When the computation got to $k = 3$, the value M_{14} had already been set to the minimal value, and P_{24} had been set to 2. So the condition of the if-then statement was false, and no changes were made. Therefore, P_{ij} gets the value of k closest to 1 whenever there are two or more values of k that give the same value to the expression $M_{ik} + M_{kj}$, and that value is less than M_{ij}.

Application Notes

Before we finish with this topic, let's make a couple of comments. If we have a digraph that is not weighted, then we can still find shortest distances and shortest paths with (4.8) and (4.9). Just let each edge have weight 1. Then the matrix M produced by either (4.8) or (4.9) will give us the length of a shortest path, and the matrix P produced by (4.9) will allow us to find a path of shortest length.

If we have a weighted graph that is not directed, then we can still use (4.8) and (4.9) to find shortest distances and shortest paths. Just modify the weighted adjacency matrix M as follows: For each edge between i and j having weight d, set $M_{ij} = M_{ji} = d$.

Exercises

Properties

1. Write down all of the properties that each of the following binary relations satisfies from among the five properties reflexive, symmetric, transitive, irreflexive, and antisymmetric.

a. The similarity relation on the set of triangles.
b. The congruence relation on the set of triangles.
c. The relation on people that relates people with the same parents.
d. The subset relation on sets.
e. The if and only if relation on the set of statements that may be true or false.
f. The relation on people that relates people with bachelor's degrees in computer science.
g. The "is brother of" relation on the set of people.
h. The "has a common national language with" relation on countries.
i. The "speaks the primary language of" relation on the set of people.
j. The "is father of" relation on the set of people.

2. Write down all of the properties that each of the following relations satisfies from among the properties reflexive, symmetric, transitive, irreflexive, and antisymmetric.

a. $R = \{(a, b) \mid a^2 + b^2 = 1\}$ over the real numbers.
b. $R = \{(a, b) \mid a^2 = b^2\}$ over the real numbers.
c. $R = \{(x, y) \mid x \bmod y = 0 \text{ and } x, y \in \{1, 2, 3, 4\}\}$.
d. $R = \{(x, y) \mid x \text{ divides } y\}$ over the positive integers.
e. $R = \{(x, y) \mid \gcd(x, y) = 1\}$ over the positive integers.

3. Explain why each of the following relations has the properties listed.

a. The empty relation $\varnothing$ over any set is irreflexive, symmetric, antisymmetric, and transitive.
b. For any set A, the universal relation $A \times A$ is reflexive, symmetric, and transitive. If $|A| = 1$, then $A \times A$ is also antisymmetric.

4. For each of the following conditions, find the smallest relation over the set $A = \{a, b, c\}$ that satisfies the stated properties.

a. Reflexive but not symmetric and not transitive.
b. Symmetric but not reflexive and not transitive.
c. Transitive but not reflexive and not symmetric.
d. Reflexive and symmetric but not transitive.
e. Reflexive and transitive but not symmetric.
f. Symmetric and transitive but not reflexive.
g. Reflexive, symmetric, and transitive.

Composition

5. Write down suitable names for each of the following compositions.

a. isChildOf $\circ$ isChildOf.

b. isSisterOf ∘ isParentOf.

c. isSonOf ∘ isSiblingOf.

d. isChildOf ∘ isSiblingOf ∘ isParentOf.

6. Suppose we define $x\ R\ y$ to mean "x is the father of y and y has a brother." Write R as the composition of two well-known relations.

7. For each of the following properties, find a binary relation R such that R has the property but R^2 does not.

a. Irreflexive.

b. Antisymmetric.

8. Given the relation "less" over the natural numbers $\mathbb{N}$, describe each of the following compositions as a set of the form $\{(x, y) \mid \text{property}\}$.

a. less ∘ less.

b. less ∘ less ∘ less.

9. Given the three relations "less," "greater," and "notEqual" over the natural numbers $\mathbb{N}$, find each of the following compositions.

a. less ∘ greater.

b. greater ∘ less.

c. notEqual ∘ less.

d. greater ∘ notEqual.

10. Let $R = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid x + y \text{ is even}\}$. Find R^2.

Closure

11. Describe the reflexive closure of the empty relation $\varnothing$ over a set A.

12. Find the symmetric closure of each of the following relations over the set $\{a, b, c\}$.

a. $\varnothing$.

b. $\{(a, b), (b, a)\}$.

c. $\{(a, b), (b, c)\}$.

d. $\{(a, a), (a, b), (c, b), (c, a)\}$.

13. Find the transitive closure of each of the following relations over the set $\{a, b, c, d\}$.

a. $\varnothing$.

b. $\{(a, b), (a, c), (b, c)\}$.

c. $\{(a, b), (b, a)\}$.

d. $\{(a, b), (b, c), (c, d), (d, a)\}$.

14. Let $R = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid x + y \text{ is odd}\}$. Use the results of Example 4.6 to calculate $t(R)$.

15. Find an appropriate name for the transitive closure of each of the following relations.
    a. isParentOf.
    b. isChildOf.
    c. $\{(x + 1, x) \mid x \in \mathbb{N}\}$.

Path Problems

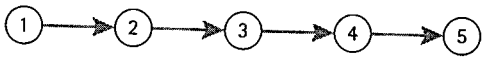
16. Suppose G is the following weighted digraph, where the triple (i, j, d) represents edge (i, j) with distance d:

$$\{(1, 2, 20), (1, 4, 5), (2, 3, 10), (3, 4, 10), (4, 3, 5), (4, 2, 10)\}.$$

    a. Draw the weighted adjacency matrix for G.
    b. Use (4.9) to compute the two matrices representing the shortest distances and the shortest paths in G.

17. Write an algorithm to compute the shortest path between two points of a weighted digraph from the matrix P produced by (4.9).

18. How many distinct path matrices can describe the shortest paths in the following graph, where it is assumed that all edges have weight = 1?



19. Write algorithms to perform each of the following actions for a binary relation R represented as an adjacency matrix.
    a. Check R for reflexivity.
    b. Check R for symmetry.
    c. Check R for transitivity.
    d. Compute $r(R)$.
    e. Compute $s(R)$.

Proofs and Challenges

20. For each of the following properties, show that if R has the property, then so does R^2.
    a. Reflexive.
    b. Symmetric.
    c. Transitive.

21. For the "less" relation over $\mathbb{N}$, show that $st(\text{less}) \neq ts(\text{less})$.

22. Prove each of the following statements about binary relations.

a. $R \circ (S \circ T) = (R \circ S) \circ T$. (associativity)

b. $R \circ (S \cup T) = R \circ S \cup R \circ T$.

c. $R \circ (S \cap T) \subseteq R \circ S \cap R \circ T$.

23. Let A be a set, R be any binary relation on A, and E be the equality relation on A. Show that $E \circ R = R \circ E = R$.

24. Prove each of the following statements about a binary relation R over a set A.

a. If R is reflexive, then $s(R)$ and $t(R)$ are reflexive.

b. If R is symmetric, then $r(R)$ and $t(R)$ are symmetric.

c. If R is transitive, then $r(R)$ is transitive.

25. Prove each of the following statements about a binary relation R over a set A.

a. $rt(R) = tr(R)$.

b. $rs(R) = sr(R)$.

c. $st(R) \subseteq ts(R)$.

26. A binary relation R over a set A is *asymmetric* if ($x\,R\,y$ and $y\,R\,x$) is false for all $x, y \in A$. Prove the following statements.

a. If R is asymmetric, then R is irreflexive.

b. If R is asymmetric, then R is antisymmetric.

c. R is asymmetric if and only if R is irreflexive and antisymmetric.

4.2 Equivalence Relations

The word "equivalent" is used in many ways. For example, we've all seen statements like "Two triangles are equivalent if their corresponding angles are equal." We want to find some general properties that describe the idea of "equivalence."

The Equality Problem

We'll start by discussing the idea of "equality" because, to most people, "equal" things are examples of "equivalent" things, whatever meaning is attached to the word "equivalent." Let's consider the following problem.

The Equality Problem

Write a computer program to check whether two objects are equal.

What is equality? Does it depend on the elements of the set? Why is equality important? What are some properties of equality? We all have an intuitive notion

of what equality is because we use it all the time. Equality is important in computer science because programs use equality tests on data. If a programming language doesn't provide an equality test for certain data, then the programmer may need to implement such a test.

The simplest equality on a set A is basic equality: $\{(x, x) \mid x \in A\}$. But most of the time we use the word "equality" in a much broader context. For example, suppose A is the set of arithmetic expressions made from natural numbers and the symbol $+$. Thus A contains expressions like $3 + 7$, 8, and $9 + 3 + 78$. Most of us already have a pretty good idea of what equality means for these expressions. For example, we probably agree that $3 + 2$ and $2 + 1 + 2$ are equal. In other words, two expressions (*syntactic* objects) are equal if they have the same value (meaning or *semantics*), which is obtained by evaluating all $+$ operations.

Are there some fundamental properties that hold for any definition of equality on a set A? Certainly we want to have $x = x$ for each element x in A (the basic equality on A). Also, whenever $x = y$, it ought to follow that $y = x$. Lastly, if $x = y$ and $y = z$, then $x = z$ should hold. Of course, these are the three properties reflexive, symmetric, and transitive.

Most equalities are more than just basic equality. That is, they equate different syntactic objects that have the same meaning. In these cases the symmetric and transitive properties are needed to convey our intuitive notion of equality. For example, the following statements are true if we let "=" mean "has the same value as":

If $2 + 3 = 1 + 4$, then $1 + 4 = 2 + 3$.

If $2 + 5 = 1 + 6$ and $1 + 6 = 3 + 4$, then $2 + 5 = 3 + 4$.

4.2.1 Definition and Examples

Now we're ready to define equivalence. Any binary relation that is reflexive, symmetric, and transitive is called an *equivalence* relation. Sometimes people refer to an equivalence relation as an RST relation in order to remember the three properties.

Equivalence relations are all around us. Of course, the basic equality relation on any set is an equivalence relation. Similarly, the notion of equivalent triangles is an equivalence relation.

For another example, suppose we relate two books in the Library of Congress if their call numbers start with the same letter. (This is an instance in which it seems to be official policy to have a number start with a letter.) This relation is clearly an equivalence relation. Each book is related to itself (reflexive). If book A and book B have call numbers that begin with the same letter, then so do books B and A (symmetric). If books A and B have call numbers beginning with the same letter and books B and C have call numbers beginning with the same letter, then so do books A and C (transitive).

example 4.16 Sample Equivalence Relations

Here are a few more samples of equivalence relations, where the symbol $\sim$ denotes each relation.

a. For the set of integers, let $x \sim y$ mean $x + y$ is even.

b. For the set of nonzero rational numbers, let $x \sim y$ mean $xy > 0$.

c. For the set of rational numbers, let $x \sim y$ mean $x - y$ is an integer.

d. For the set of triangles, let $x \sim y$ mean x and y are similar.

e. For the set of integers, let $x \sim y$ mean $x \bmod 4 = y \bmod 4$.

f. For the set of binary trees, let $x \sim y$ mean x and y have the same depth.

g. For the set of binary trees, let $x \sim y$ mean x and y have the same number of nodes.

h. For the set of real numbers, let $x \sim y$ mean $x^2 = y^2$.

i. For the set of people, let $x \sim y$ mean x and y have the same mother.

j. For the set of TV programs, let $x \sim y$ mean x and y start at the same time and day.

end example

Intersections and Equivalence Relations

We can always verify that a binary relation is an equivalence relation by checking that the relation is reflexive, symmetric, and transitive. But in some cases we can determine equivalence by other means. For example, we have the following intersection result, which we'll leave as an exercise.

Intersection Property of Equivalence (4.10)

If E and F are equivalence relations on the set A, then $E \cap F$ is an equivalence relation on A.

The practical use of (4.10) comes about when we notice that a relation $\sim$ on a set A is defined in the following form, where E and F are relations on A.

$$x \sim y \text{ iff } x \, E \, y \text{ and } x \, F \, y.$$

This is just another way of saying that $x \sim y$ iff $(x, y) \in E \cap F$. So if we can show that E and F are equivalence relations, then (4.10) tells us that $\sim$ is an equivalence relation.

example **4.17 Equivalent Binary Trees**

Suppose we define the relation $\sim$ on the set of binary trees by

$x \sim y$ iff x and y have the same depth and the same number of nodes.

From Example 4.16 we know that "has the same depth as" and "has the same number of nodes as" are both equivalence relations. Therefore, $\sim$ is an equivalence relation.

end example

Equivalence Relations from Functions (Kernel Relations)

A very powerful technique for obtaining equivalence relations comes from the fact that any function defines a natural equivalence relation on its domain by relating elements that map to the same value. In other words, for any function $f : A \to B$, we obtain an equivalence relation $\sim$ on A by

$$x \sim y \quad \text{iff} \quad f(x) = f(y).$$

It's easy to see that $\sim$ is an equivalence relation. The reflexive property follows because $f(x) = f(x)$ for all $x \in A$. The symmetric property follows because $f(x) = f(y)$ implies $f(y) = f(x)$. The transitive property follows because $f(x) = f(y)$ and $f(y) = f(z)$ implies $f(x) = f(z)$.

An equivalence relation defined in this way is called the *kernel relation* for f. Let's state the result for reference.

Kernel Relations **(4.11)**

If f is a function with domain A, then the relation $\sim$ defined by

$$x \sim y \text{ iff } f(x) = f(y)$$

is an equivalence relation on A, and it is called the *kernel relation* of f.

For example, notice that the relation given in Part (e) of Example 4.16 is the kernel relation for the function $f(x) = x \bmod 4$. Thus Part (e) of Example 4.16 is an equivalence relation by (4.11). Several other parts of Example 4.16 are also kernel relations. The nice thing about kernel relations is that they are always equivalence relations. So there is nothing to check. For example, we can use (4.11) to generalize Part (e) of Example 4.16 to the following important result.

Mod Function Equivalence **(4.12)**

If S is any set of integers and n is a positive integer, then the relation $\sim$ defined by

$$x \sim y \text{ iff } x \bmod n = y \bmod n$$

is an equivalence relation over S.

In many cases it's possible to show that a relation is an equivalence relation by rewriting its definition so that it is the kernel relation of some function.

example 4.18 A Numeric Equivalence Relation

Suppose we're given the relation $\sim$ defined on integers by

$$x \sim y \text{ if and only if } x - y \text{ is an even integer.}$$

We'll show that $\sim$ is an equivalence relation by writing it as the kernel relation of a function. Notice that $x - y$ is even if and only if x and y are both even or both odd. We can test whether an integer x is even or odd by checking whether $x \bmod 2 = 0$ or 1. So we can write our original definition of $\sim$ in terms of the mod function:

$$\begin{aligned} x \sim y \quad &\text{iff} \quad x - y \text{ is an even integer} \\ &\text{iff} \quad x \text{ and } y \text{ are both even or both odd} \\ &\text{iff} \quad x \bmod 2 = y \bmod 2. \end{aligned}$$

We can now conclude that $\sim$ is an equivalence relation because it's the kernel relation of the function f defined by $f(x) = x \bmod 2$.

end example

The Equivalence Problem

We can generalize the equality problem to the following more realistic problem of equivalence.

> **The Equivalence Problem**
> Write a computer program to check whether two objects are equivalent.

example 4.19 Binary Trees with the Same Structure

Suppose we need two binary trees to be equivalent whenever they have the same structure regardless of the values of the nodes. For binary trees S and T, let equiv(S, T) be true if S and T are equivalent and false otherwise. Here is a program to compute equiv.

$$\begin{aligned} \text{equiv}\,(S,T) = \ &\text{if } S = \langle\,\rangle \text{ and } T = \langle\,\rangle \text{ then True} \\ &\text{else if } S = \langle\,\rangle \text{ or } T = \langle\,\rangle \text{ then False} \\ &\text{else equiv}\,(\text{left}\,(S)\,, \text{left}\,(T)) \text{ and equiv}\,(\text{right}\,(S)\,, \text{right}\,(T))\,. \end{aligned}$$

end example

4.2.2 Equivalence Classes

The nice thing about an equivalence relation over a set is that it defines a natural way to group elements of the set into disjoint subsets. These subsets are called equivalence classes, and here's the definition.

Equivalence Class

Let R be an equivalence relation on a set S. If $a \in S$, then the *equivalence class* of a, denoted by $[a]$, is the subset of S consisting of all elements that are equivalent to a. In other words, we have

$$[a] = \{x \in S \mid x \; R \; a\}.$$

For example, we always have $a \in [a]$ because of the property $a \; R \; a$.

example 4.20 Equivalent Strings

Consider the relation $\sim$ defined on strings over the alphabet $\{a, b\}$ by

$$x \sim y \text{ iff } x \text{ and } y \text{ have the same length.}$$

Notice that $\sim$ is an equivalence relation because it is the kernel relation of the length function. Some sample equivalences are $abb \sim bab$ and $ba \sim aa$. Let's look at a few equivalence classes.

$$\begin{aligned}
[\Lambda] &= \{\Lambda\}, \\
[a] &= \{a, b\}, \\
[ab] &= \{ab, aa, ba, bb\}, \\
[aaa] &= \{aaa, aab, aba, baa, abb, bab, bba, bbb\}.
\end{aligned}$$

Notice that any member of an equivalence class can define the class. For example, we have

$$\begin{aligned}
[a] &= [b] = \{a, b\}, \\
[ab] &= [aa] = [ba] = [bb] = \{ab, aa, ba, bb\}.
\end{aligned}$$

end example

Equivalence classes enjoy a very nice property, namely that any two such classes are either equal or disjoint. Here is the result in more formal terms.

Property of Equivalences (4.13)

Let S be a set with an equivalence relation R. If $a, b \in S$, then either $[a] = [b]$ or $[a] \cap [b] = \varnothing$.

Proof: It suffices to show that $[a] \cap [b] \neq \varnothing$ implies $[a] = [b]$. If $[a] \cap [b] \neq \varnothing$, then there is a common element $c \in [a] \cap [b]$. It follows that cRa and cRb. From the symmetric and transitive properties of R, we conclude that aRb. To show that $[a] = [b]$, we'll show that $[a] \subseteq [b]$ and $[b] \subseteq [a]$. Let $x \in [a]$. Then xRa. Since aRb, the transitive property tells us that xRb, which implies that $x \in [b]$. Therefore, $[a] \subseteq [b]$. In an entirely similar manner we obtain $[b] \subseteq [a]$. Therefore, we have the desired result $[a] = [b]$. QED.

4.2.3 Partitions

By a *partition* of a set we mean a collection of nonempty subsets that are disjoint from each other and whose union is the whole set. For example, the set $S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ can be partitioned in many ways, one of which consists of the following three subsets of S:

$$\{0, 1, 4, 9\}, \{2, 5, 8\}, \{3, 6, 7\}.$$

Notice that, if we wanted to, we could define an equivalence relation on S by saying that $x \sim y$ iff x and y are in the same set of the partition. In other words, we would have

$$\begin{aligned} [0] &= \{0, 1, 4, 9\}, \\ [2] &= \{2, 5, 8\}, \\ [3] &= \{3, 6, 7\}. \end{aligned}$$

We can do this for any partition of any set.

But something more interesting happens when we start with an equivalence relation on S. For example, let $\sim$ be the following relation on S:

$$x \sim y \text{ iff } x \bmod 4 = y \bmod 4.$$

This relation is an equivalence relation because it is the kernel relation of the function $f(x) = x \bmod 4$. Now let's look at some of the equivalence classes.

$$\begin{aligned} [0] &= \{0, 4, 8\}. \\ [1] &= \{1, 5, 9\}. \\ [2] &= \{2, 6\}. \\ [3] &= \{3, 7\}. \end{aligned}$$

Notice that these equivalence classes form a partition of S. This is no fluke. It always happens for any equivalence relation on any set S. To see this, notice that if $s \in S$, then $s \in [s]$, which says that S is the union of the equivalence classes. We also know from (4.13) that distinct equivalence classes are disjoint.

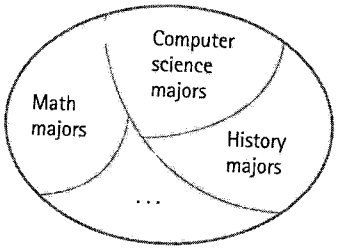


Figure 4.8 A partition of students.

Therefore, the set of equivalence classes forms a partition of S. Here's a summary of our discussion.

Equivalence Relations and Partitions **(4.14)**

If R is an equivalence relation on the set S, then the equivalence classes form a partition of S. Conversely, if P is a partition of a set S, then there is an equivalence relation on S whose equivalence classes are sets of P.

For example, let S denote the set of all students at some university, and let M be the relation on S that relates two students if they have the same major. (Assume here that every student has exactly one major.) It's easy to see that M is an equivalence relation on S and each equivalence class is the set of all the students majoring in the same subject. For example, one equivalence class is the set of computer science majors. The partition of S is pictured by the Venn diagram in Figure 4.8.

example **4.21 Partitioning a Set of Strings**

The relation from Example 4.20 is defined on the set $S = \{a, b\}^*$ of all strings over the alphabet $\{a, b\}$ by

$$x \sim y \text{ iff } x \text{ and } y \text{ have the same length.}$$

For each natural number n, the equivalence class $[a^n]$ contains all strings over $\{a, b\}$ that have length n. So we can write

$$S = \{a, b\}^* = [\Lambda\,] \cup [a] \cup [aa] \cup \cdots \cup [a^n] \cup \cdots.$$

end example

example 4.22 A Partition of the Natural Numbers

Let $\sim$ be the relation on the natural numbers defined by

$$x \sim y \text{ iff } \lfloor x/10 \rfloor = \lfloor y/10 \rfloor.$$

This is an equivalence relation because it is the kernel relation of the function $f(x) = \lfloor x/10 \rfloor$. After checking a few values we see that each equivalence class is a decade of numbers. For example,

$$[0] = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\},$$
$$[10] = \{10, 11, 12, 13, 14, 15, 16, 17, 18, 19\},$$

and in general, for any natural number n,

$$[10n] = \{10n, 10n + 1, \ldots, 10n + 9].$$

So we have $\mathbb{N} = [0] \cup [10] \cup \cdots \cup [10n] \cup \cdots$.

end example

example 4.23 Partitioning with Mod 5

Let R be the equivalence relation on the integers $\mathbb{Z}$ defined by

$$a \; R \; b \text{ iff } a \bmod 5 = b \bmod 5.$$

After some checking we see that the partition of $\mathbb{Z}$ consists of the following five equivalence classes

$$[0] = \{\ldots -10, -5, 0, 5, 10, \ldots\},$$
$$[1] = \{\ldots -9, -4, 1, 6, 11, \ldots\},$$
$$[2] = \{\ldots -8, -3, 2, 7, 12, \ldots\},$$
$$[3] = \{\ldots -7, -2, 3, 8, 13, \ldots\},$$
$$[4] = \{\ldots -6, -1, 4, 9, 14, \ldots\}.$$

Remember, it doesn't matter which element of a class is used to represent it. For example, $[0] = [5] = [-15]$. It is clear that the five classes are disjoint from each other and that $\mathbb{Z}$ is the union of the five classes.

end example

example 4.24 Software Testing

If the input data set for a program is infinite, then the program can't be tested on every input. However, every program has a finite number of instructions. So we should be able to find a finite data set to cause all instructions of the program

to be executed. For example, suppose p is the following program, where x is an integer and q, r, and s represent other parts of the program:

$p(x)$: **if** $x > 0$ **then** $q(x)$
else if x is even **then** $r(x)$
else $s(x)$
fi
fi

The condition "$x > 0$" causes a natural partition of the integers into the positives and the nonpositives. The condition "x is even" causes a natural partition of the nonpositives into the even nonpositives and the odd nonpositives. So we have a partition of the integers into the following three subsets:

$$\{1, 2, 3, \ldots\}, \{0, -2, -4, \ldots\}, \{-1, -3, -5, \ldots\}.$$

Now we can test the instructions in q, r, and s by picking three numbers, one from each set of the partition. For example, $p(1)$, $p(0)$, and $p(-1)$ will do the job. Of course, further partitioning may be necessary if q, r, or s contains further conditional statements. The equivalence relation induced by the partition relates two integers x and y if and only if $p(x)$ and $p(y)$ execute the same set of instructions.

end example

Refinement of a Partition

Suppose that P and Q are two partitions of a set S. If each set of P is a subset of a set in Q, then P is a *refinement* of Q. We also say P is *finer* than Q or Q is *coarser* than P. The finest of all partitions on S is the collection of singleton sets. The coarsest of all partitions of S is $\{S\}$.

For example, here is a listing of four partitions of $\{a, b, c, d\}$ that are successive refinements from the coarsest to finest:

$\{\{a, b, c, d\}\}$ (coarsest)
$\{\{a, b\}, \{c, d\}\}$
$\{\{a, b\}, \{c\}, \{d\}\}$
$\{\{a\}, \{b\}, \{c\}, \{d\}\}$ (finest).

example **4.25 Partitioning with Mod 2**

Let R be the relation over $\mathbb{N}$ defined by

$$a \; R \; b \text{ iff } a \bmod 2 = b \bmod 2.$$

Then R is an equivalence relation because it is the kernel relation of the function f defined by $f(x) = x \bmod 2$. The corresponding partition of $\mathbb{N}$ consists of the two subsets

$$[0] = \{0, 2, 4, 6, \dots\},$$
$$[1] = \{1, 3, 5, 7, \dots\}.$$

Can we find a refinement of this partition? Sure. Let T be defined by

$$a \; T \; b \text{ iff } a \bmod 4 = b \bmod 4.$$

T induces the following partition of $\mathbb{N}$ that is a refinement of the partition induced by R because we get the following four equivalence classes:

$$[0] = \{0, 4, 8, 12, \dots\},$$
$$[1] = \{1, 5, 9, 13, \dots\},$$
$$[2] = \{2, 6, 10, 14, \dots\},$$
$$[3] = \{3, 7, 11, 15, \dots\}.$$

This partition is indeed a refinement of the preceding partition. Can we find a refinement of this partition? Yes, because we can continue the process forever. Just let k be a power of 2 and define T_k by

$$a \; T_k \; b \text{ iff } a \bmod k = b \bmod k.$$

So the partition for each T_{2k} is a refinement of the partition for T_k.

end example

An Intersection Property

We noted in (4.10) that the intersection of equivalence relations over a set A is also an equivalence relation over A. It also turns out that the equivalence classes for the intersection are intersections of equivalence classes for the given relations. Here is the statement and we'll leave the proof as an exercise.

Intersection Property of Equivalence (4.15)

Let E and F be equivalence relations on a set A. Then the equivalence classes for the relation $E \cap F$ are of the form $[x] = [x]_E \cap [x]_F$, where $[x]_E$ and $[x]_F$ denote the equivalence classes of x for E and F, respectively.

example **4.26 Intersecting Equivalence Relations**

Let $\sim$ be the relation on the natural numbers defined by

$$x \sim y \quad \text{iff} \quad \lfloor x/10 \rfloor = \lfloor y/10 \rfloor \text{ and } x + y \text{ is even.}$$

Notice that $\sim$ is the intersection of two relations E and F, where $x\ E\ y$ means $\lfloor x/10 \rfloor = \lfloor y/10 \rfloor$ and $x\ F\ y$ means $x + y$ is even. We can observe that $x + y$ is even if and only if $x \bmod 2 = y \bmod 2$. So both E and F are kernel relations of functions and thus are equivalence relations. Therefore, $\sim$ is an equivalence relation by (4.10). We computed the equivalence classes for E and F in Examples 4.22 and 4.25. The equivalence classes for E are of the following form for each natural number n.

$$[10n] = \{10n,\ 10n + 1,\ \ldots,\ 10n + 9\}.$$

The equivalence classes for F are

$$[0] = \{0, 2, 4, 6, \ldots\},$$
$$[1] = \{1, 3, 5, 7, \ldots\}.$$

By (4.15) the equivalence classes for $\sim$ have the following form for each n:

$$[10n] \cap [0] = \{10n, 10n + 2, 10n + 4, 10n + 6, 10n + 8\},$$
$$[10n] \cap [1] = \{10n + 1, 10n + 3, 10n + 5, 10n + 7, 10n + 9\}.$$

end example

example **4.27 Solving the Equality Problem**

If we want to define an equality relation on a set S of objects that do not have any established meaning, then we can use the basic equality relation $\{(x, x) \mid x \in S\}$. On the other hand, suppose a meaning has been assigned to each element of S. We can represent the meaning by a mapping m from S to a set of values V. In other words, we have a function $m : S \to V$. It's natural to define two elements of S to be equal if they have the same meaning. That is, we define $x = y$ if and only if $m(x) = m(y)$. This equality relation is just the kernel relation of m.

For example, let S denote the set of arithmetic expressions made from nonempty unary strings and the symbol $+$. For example, some typical expressions in S are 1, 11, 111, 1+1, 11+111+1. Now let's assign a meaning to each expression in S. Let $m(1^n) = n$ for each positive natural number n. If $e + e'$ is an expression of S, we define $m(e + e') = m(e) + m(e')$. We'll assume that $+$ is applied left to right. For example, the value of the expression $1 + 11 + 11$ can be calculated as follows:

$$\begin{aligned} m(1 + 111 + 11) &= m((1 + 111) + 11) \\ &= m(1 + 111) + m(11) \\ &= m(1) + m(111) + 2 \\ &= 1 + 3 + 2 \\ &= 6. \end{aligned}$$

If we define two expressions of S to be equal when they have the same meaning, then the desired equality relation on S is the kernel relation of m. So the partition of S induced by the kernel relation of m consists of the sets of expressions with equal values. For example, the equivalence class [1111] contains the eight expressions

1+1+1+1, 1+1+11, 1+11+1, 11+1+1, 11+11, 1+111, 111+1, 1111.

end example

4.2.4 Generating Equivalence Relations

Any binary relation can be considered as the *generator* of an equivalence relation obtained by adding just enough pairs to make the result reflexive, symmetric, and transitive. In other words, we can take the reflexive, symmetric, and transitive closures of the binary relation.

Does the order that we take closures make a difference? For example, what about $str(R)$? An example will suffice to show that $str(R)$ need not be an equivalence relation. Let $A = \{a, b, c\}$ and $R = \{(a, b), (a, c), (b, b)\}$. Then

$$str(R) = \{(a, a), (b, b), (c, c), (a, b), (b, a), (a, c), (c, a)\}.$$

This relation is reflexive and symmetric, but it's not transitive. On the other hand, we have $tsr(R) = A \times A$, which is an equivalence relation. As the next result shows, $tsr(R)$ is always an equivalence relation.

The Smallest Equivalence Relation (4.16)

If R is a binary relation on A, then $tsr(R)$ is the smallest equivalence relation that contains R.

Proof: The inheritance properties of (4.4) tell us that $tsr(R)$ is an equivalence relation. To see that it's the smallest equivalence relation containing R, we'll let T be an arbitrary equivalence relation containing R. Since $R \subseteq T$ and T is reflexive, it follows that $r(R) \subseteq T$. Since $r(R) \subseteq T$ and T is symmetric, it follows that $sr(R) \subseteq T$. Since $sr(R) \subseteq T$ and T is transitive, it follows that $tsr(R) \subseteq T$. So $tsr(R)$ is contained in every equivalence relation that contains R. Thus it's the smallest equivalence relation containing R. QED.

example **4.28 Family Trees**

Let R be the "is parent of" relation for a set of people. In other words, $(x, y) \in R$ iff x is a parent of y. Suppose we want to answer questions like the following:

Is x a descendant of y?
Is x an ancestor of y?

Are x and y related in some way?
What is the relationship between x and y?

Each of these questions can be answered from the given information by finding whether an appropriate path exists between x and y. But if we construct $t(R)$, then things get better because $(x, y) \in t(R)$ iff x is an ancestor of y. So we can find out whether x is an ancestor of y or x is a descendant of y by looking to see whether $(x, y) \in t(R)$ or $(y, x) \in t(R)$.

If we want to know whether x and y are related in some way, then we would have to look for paths in $t(R)$ taking each of x and y to a common ancestor. But if we construct $ts(R)$, then things get better because $(x, y) \in ts(R)$ iff x and y have a common ancestor. So we can find out whether x and y are related in some way by looking to see whether $(x, y) \in ts(R)$.

If x and y are related, then we might want to know the relationship. This question is asking for paths from x and y to a common ancestor, which can be done by searching $t(R)$ for the common ancestor and keeping track of each person along the way.

Notice also that the set of people can be partitioned into family trees by the equivalence relation $tsr(R)$. So the simple "is parent of" relation is the generator of an equivalence relation that constructs family trees.

end example

An Equivalence Problem

Suppose we have an equivalence relation over a set S that is generated by a given set of pairs. For example, the equivalence relation might be the family relationship "is related to" and the generators might be a set of parent-child pairs.

Can we represent the generators in such a way that we can find out whether two arbitrary elements of S are equivalent? If two elements are equivalent, can we find a sequence of generators to confirm the fact? The answer to both questions is yes. We'll present a solution due to Galler and Fischer [1964], which uses a special kind of tree structure to represent the equivalence classes.

The idea is to use the generating pairs to build the partition of S induced by the equivalence relation. For example, let $S = \{1, 2, \ldots, 10\}$, let $\sim$ denote the equivalence relation on S, and let the generators be the following pairs:

$$1 \sim 8,\ 4 \sim 5,\ 9 \sim 2,\ 4 \sim 10,\ 3 \sim 7,\ 6 \sim 3,\ 4 \sim 9.$$

To have something concrete in mind, let the numbers $1, 2, \ldots, 10$ be people, let $\sim$ be "is related to," and let the generators be "parent $\sim$ child" pairs.

The construction process starts by building the following ten singleton equivalence classes to represent the partition of S caused by the reflexive property $x \sim x$.

$$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}.$$

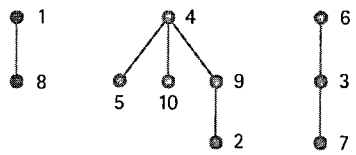


Figure 4.9 Equivalence classes as trees.

i	1	2	3	4	5	6	7	8	9	10
$p[i]$	0	9	6	0	4	0	3	1	4	4

Figure 4.10 Equivalence classes as an array.

Now we process the generators, one at a time. The generator $1 \sim 8$ is processed by forming the union of the equivalence classes that contain 1 and 8. In other words, the partition becomes

$$\{1, 8\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{9\}, \{10\}.$$

Continuing in this manner to process the other generators, we eventually obtain the partition of S consisting of the following three equivalence classes.

$$\{1, 8\}, \{2, 4, 5, 9, 10\}, \{3, 6, 7\}.$$

Representing Equivalence Classes

To answer questions about an equivalence relation, we need to consider its representation. We can represent each equivalence class in the partition as a tree, where the generator $a \sim b$ will be processed by creating the branch "a is the parent of b." For our example, if we process the generators in the order in which they are written, then we obtain the three trees in Figure 4.9.

A simple way to represent these trees is with a 10-tuple (a 1-dimensional array of size 10) named p, where $p[i]$ denotes the parent of i. We'll let $p[i] = 0$ mean that i is a root. Figure 4.10 shows the three equivalence classes represented by p.

Now it's easy to answer the question "Is $a \sim b$?" Just find the roots of the trees to which a and b belong. If the roots are the same, the answer is yes. If the answer is yes, then there is another question, "Can you find a sequence of equivalences to show that $a \sim b$?" One way to do this is to locate one of the numbers, say b, and rearrange the tree to which b belongs so that b becomes the root. This can be done easily by reversing the links from b to the root. Once we have b at the root, it's an easy matter to read off the equivalences from a to b. We'll leave it as an exercise to construct an algorithm to do the reversing.

For example, if we ask whether $5 \sim 2$, we find that 5 and 2 belong to the same tree. So the answer is yes. To find a set of equivalences to prove that $5 \sim 2$,

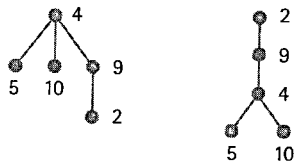


Figure 4.11 Proof that $5 \sim 2$.

we can, for example, reverse the links from 2 to the root of the tree. The before and after pictures are given in Figure 4.11.

Now it's an easy computation to traverse the tree from 5 to the root 2 and read off the equivalences $5 \sim 4$, $4 \sim 9$, and $9 \sim 2$.

Kruskal's Algorithm for Minimal Spanning Trees

In Chapter 1 we discussed Prim's algorithm to find a minimal spanning tree for a connected weighted undirected graph. Let's look an another such algorithm, due to Kruskal [1956], which uses equivalence classes.

The algorithm constructs a minimal spanning tree as follows: Starting with an empty tree, an edge $\{a, b\}$ of smallest weight is chosen from the graph. If there is no path in the tree from a to b, then the edge $\{a, b\}$ is added to the tree. This process is repeated with the remaining edges of the graph until the tree contains all vertices of the graph.

At any point in the algorithm, the edges in the spanning tree define an equivalence relation on the set of vertices of the graph. Two vertices a and b are equivalent iff there is a path between a and b in the tree. Whenever an edge $\{a, b\}$ is added to the spanning tree, the equivalence relation is modified by creating the equivalence class $[a] \cup [b]$. The algorithm ends when there is exactly one equivalence class consisting of all the vertices of the graph. Here are the steps of the algorithm.

Kruskal's Algorithm

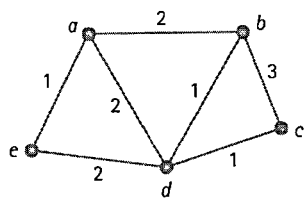
1. Sort the edges of the graph by weight, and let L be the sorted list.
2. Let T be the minimal spanning tree and initialize $T := \varnothing$.
3. For each vertex v of the graph, create the equivalence class $[v] = \{v\}$.
4. **while** there are 2 or more equivalence classes **do**
   Let $\{a, b\}$ be the edge at the head of L;
   $L :=$ tail(L);
   if $[a] \neq [b]$ **then**
   $T := T \cup \{\{a, b\}\}$;
   Replace the equivalence classes $[a]$ and $[b]$ by $[a] \cup [b]$
   fi

   od

To implement the algorithm, we must find a representation for the equivalence classes. For example, we might use a parent array like the one we've been discussing.

example 4.29 Minimal Spanning Trees

We'll use Kruskal's algorithm to construct a minimal spanning tree for the following weighted graph:



To see how the algorithm works, we'll do a trace of each step. We'll assume that the edges have been sorted by weight in the following order:

$$\{a, e\}, \{b, d\}, \{c, d\}, \{a, b\}, \{a, d\}, \{e, d\}, \{b, c\}.$$

The following table shows the value of the spanning tree T and the equivalence classes at each step, starting with the initialization values.

Spanning Tree T	*Equivalence Classes*
$\{\}$	$\{a\}, \{b\}, \{c\}, \{d\}, \{e\}$
$\{\{a, e\}\}$	$\{a, e\}, \{b\}, \{c\}, \{d\}$
$\{\{a, e\}, \{b, d\}\}$	$\{a, e\}, \{b, d\}, \{c\}$
$\{\{a, e\}, \{b, d\}, \{c, d\}\}$	$\{a, e\}, \{b, c, d\}$
$\{\{a, e\}, \{b, d\}, \{c, d\}, \{a, b\}\}$	$\{a, b, c, d, e\}$

The algorithm stops because there is only one equivalence class. So T is a spanning tree for the graph.

end example

Exercises

Properties

1. Verify that each of the following relations is an equivalence relation.
   a. $x \sim y$ iff x and y are points in a plane equidistant from a fixed point.
   b. $s \sim t$ iff s and t are strings with the same occurrences of each letter.

c. $x \sim y$ iff $x + y$ is even, over the set of natural numbers.
d. $x \sim y$ iff $x - y$ is an integer, over the set of rational numbers.
e. $x \sim y$ iff $xy > 0$, over the set of nonzero rational numbers.

2. Each of the following relations is not an equivalence relation. In each case, find the properties that are not satisfied.

a. $a\ R\ b$ iff $a + b$ is odd, over the set of integers.
b. $a\ R\ b$ iff a/b is an integer, over the set of nonzero rational numbers.
c. $a\ R\ b$ iff $|a - b| \leq 5$, over the set of natural numbers.
d. $a\ R\ b$ iff either $a \bmod 4 = b \bmod 4$ *or* $a \bmod 6 = b \bmod 6$, over $\mathbb{N}$.
e. $a\ R\ b$ iff $x < a/10 < x + 1$ and $x \leq b/10 < x + 1$ for some integer x.

Equivalence Classes

3. For each of the following functions f with domain $\mathbb{N}$, describe the equivalence classes of the kernel relation of f.

a. $f(x) = 7$.
b. $f(x) = x$.
c. $f(x) = \text{floor}(x/2)$.
d. $f(x) = \text{floor}(x/3)$.
e. $f(x) = \text{floor}(x/4)$.
f. $f(x) = \text{floor}(x/k)$ for a fixed positive integer k.
g. $f(x) =$ if $0 \leq x \leq 10$ then 10 else $x - 1$.

4. For each of the following functions f, describe the equivalence classes of the kernel relation of f that partition the domain of f.

a. $f : \mathbb{Z} \to \mathbb{N}$ is defined by $f(x) = |x|$.
b. $f : \mathbb{R} \to \mathbb{Z}$ is defined by $f(x) = \text{floor}(x)$.

5. Describe the equivalence classes for each of the following relations on $\mathbb{N}$.

a. $x \sim y$ iff $x \bmod 2 = y \bmod 2$ *and* $x \bmod 3 = y \bmod 3$.
b. $x \sim y$ iff $x \bmod 2 = y \bmod 2$ *and* $x \bmod 4 = y \bmod 4$.
c. $x \sim y$ iff $x \bmod 4 = y \bmod 4$ *and* $x \bmod 6 = y \bmod 6$.

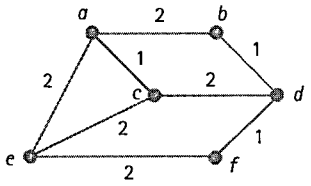
6. Given the following set of words.

$$\{rot,\ tot,\ root,\ toot,\ roto,\ toto,\ too,\ to,\ otto\}.$$

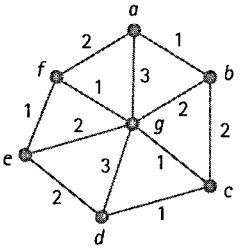
a. Let f be the function that maps a word to its set of letters. For the kernel relation of f, describe the equivalence classes.
b. Let f be the function that maps a word to its bag of letters. For the kernel relation of f, describe the equivalence classes.

Spanning Trees

7. Use Kruskal's algorithm to find a minimal spanning tree for each of the following weighted graphs.



a.



b.

Proofs and Challenges

8. Let R be a relation on a set S such that R is symmetric and transitive and for each $x \in S$ there is an element $y \in S$ such that $x \, R \, y$. Prove that R is an equivalence relation (i.e., prove that R is reflexive).

9. Let E and F be equivalence relations on the set A, Show that $E \cap F$ is an equivalence relation on A.

10. Let E and F be equivalence relations on a set A and for each $x \in A$ let $[x]_E$ and $[x]_F$ denote the equivalence classes of x for E and F, respectively. Show that the equivalence classes for the relation $E \cap F$ are of the form $[x] = [x]_E \cap [x]_F$ for all $x \in A$.

11. Which relations among the following list are equal to $tsr(R)$, the smallest equivalence relation generated by R?

$$trs(R),\ str(R),\ srt(R),\ rst(R),\ rts(R).$$

12. In the equivalence problem we represented equivalence classes as a set of trees, where the nodes of the trees are the numbers $1, 2, \ldots, n$. Suppose the trees are represented by an array $p[1], \ldots, p[n]$, where $p[i]$ is the parent of i. Suppose also that $p[i] = 0$ when i is a root. Write a procedure that takes a node i and rearranges the tree that i belongs to so that i is the root, by reversing the links from the root to i.

13. (*Factoring a Function*). An interesting consequence of equivalence relations and partitions is that any function f can be factored into a composition of two functions, one an injection and one a surjection. For a function $f : A \to B$, let P be the partition of A by the kernel relation of f. Then define the function $s : A \to P$ by $s(a) = [a]$ and define $i : P \to B$ by $i([a]) = f(a)$. Prove that s is a surjection, i is an injection, and $f = i \circ s$.

4.3 Order Relations

Each day we see the idea of "order" used in many different ways. For example, we might encounter the expression $1 < 2$. We might notice that someone is older than someone else. We might be interested in the third component of the tuple (x, d, c, m). We might try to follow a recipe. Or we might see that the word "aardvark" resides at a certain place in the dictionary. The concept of order occurs in many different forms, but they all have the common idea of some object preceding another object.

Two Essential Properties of Order

Let's try to formally describe the concept of order. To have an ordering, we need a set of elements together with a binary relation having certain properties. What are these properties?

Well, our intuition tells us that if a, b, and c are objects that are ordered so that a precedes b and b precedes c, then we certainly want a to precede c. In other words, an ordering should be transitive. For example, if a, b, and c are natural numbers and $a < b$ and $b < c$, then we have $a < c$.

Our intuition also tells us that we don't want distinct objects preceding each other. In other words, if a and b are distinct objects and a precedes b, then b can't precede a. In still other words, if a precedes b and b precedes a then we better have $a = b$. For example, if a, b, and c are natural numbers and $a \leq b$ and $b \leq a$, we certainly want $a = b$. In other words, an ordering should be antisymmetric.

For example, over the natural numbers we recognize that the relation $<$ is an ordering and we notice that it is transitive and antisymmetric. Similarly, the relation $\leq$ is an ordering and we notice that it is transitive and antisymmetric. So the two essential properties of any kind of order are antisymmetric and transitive.

Let's look at how different orderings can occur in trying to perform the tasks of a recipe.

example 4.30 A Pancake Recipe

Suppose we have the following recipe for making pancakes.

1. Mix the dry ingredients (flour, sugar, baking powder) in a bowl.
2. Mix the wet ingredients (milk, eggs) in a bowl.
3. Mix the wet and dry ingredients together.
4. Oil the pan. (It's an old pan.)
5. Heat the pan.
6. Make a test pancake and throw it away.
7. Make pancakes.

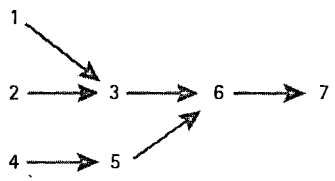


Figure 4.12 A pancake recipe.

Steps 1 through 7 indicate an ordering for the steps of the recipe. But the steps could also be done in some other order. To help us discover some other orders, let's define a relation R on the seven steps of the pancake recipe as follows:

$i\ R\ j$ means that step i must be done before step j.

Notice that R is antisymmetric and transitive. We can picture R as the digraph (without the transitive arrows) in Figure 4.12.

The graph helps us pick out different orders for the steps of the recipe. For example, the following ordering of steps will produce pancakes just as well.

4, 5, 2, 1, 3, 6, 7.

So there are several ways to perform the recipe. For example, three people could work in parallel doing tasks 1, 2, and 4 at the same time.

end example

This example demonstrates that different orderings for time-oriented tasks are possible whenever some tasks can be done at different times without changing the outcome. The orderings can be discovered by modeling the tasks by a binary relation R defined by

$i\ R\ j$ means that step i must be done before step j.

Notice that R is irreflexive because time-oriented tasks can't be done before themselves. If there are at least two tasks that are not related by R, as in Example 4.30, then there will be at least two different orderings of the tasks.

4.3.1 Partial Orders

Now let's get down to business and discuss the basic ideas and techniques of ordering. The two essential properties of order suffice to define the notion of partial order.

Definition of a Partial Order

A binary relation is called a *partial order* if it is antisymmetric, transitive, and either reflexive or irreflexive.

We should note that this definition is a bit more general than most definitions found in the literature. Some definitions require the reflexive property, while others require the irreflexive property. The reasons for requiring one of these properties are mostly historical and neither property is required to describe the idea of ordering. So if a partial order is reflexive and we wish to emphasize it, we'll call it a *reflexive partial order*. For example, $\leq$ is a reflexive partial order on the integers. If a partial order is irreflexive and we wish to emphasize it, we'll call it an *irreflexive partial order*. For example, $<$ is an irreflexive partial order on the integers.

Definition of a Partially Ordered Set

The set over which a partial order is defined is called a *partially ordered set*—or *poset* for short. If we want to emphasize the fact that R is the partial order that makes S a poset, we'll write $\langle S, R\rangle$ and call it a poset.

For example, in our pancake example we defined a partial order R on the set of recipe steps $\{1, 2, 3, 4, 5, 6, 7\}$. So we can say that $\langle\{1, 2, 3, 4, 5, 6, 7\}, R\rangle$ is a poset. There are many more examples of partial orders. For example, $\langle \mathbb{N}, < \rangle$ and $\langle \mathbb{N}, \leq \rangle$ are posets because the relations $<$ and $\leq$ are both antisymmetric and transitive.

Partial and Total

The word "partial" is used in the definition because we include the possibility that some elements may not be related to each other, as in the pancake recipe example. For another example, consider the subset relation on power$(\{a, b, c\})$. Certainly the subset relation is antisymmetric and transitive. So we can say that $\langle$power$(\{a, b, c\}), \subseteq \rangle$ is a poset. Notice that there are some subsets that are not related. For example, $\{a, b\}$ and $\{a, c\}$ are not related by the relation $\subseteq$.

Suppose R is a binary relation on a set S and $x, y \in S$. We say that x and y are *comparable* if either $x\ R\ y$ or $y\ R\ x$. In other words, elements that are related are comparable. If every pair of distinct elements in a partial order are comparable, then the order is called a *total* order (also called a *linear* order). If R is a total order on the set S, then we also say that S is a *totally ordered set* or a *linearly ordered set*. For example, the natural numbers are totally ordered by both "less" and "lessOrEqual." In other words, $\langle \mathbb{N}, < \rangle$ and $\langle \mathbb{N}, \leq \rangle$ are totally ordered sets.

example 4.31 The Divides Relation

Let's look at some interesting posets that can be defined by the divides relation, $|$. First we'll consider the set $\mathbb{N}$. If $a|b$ and $b|c$, then $a|c$. Thus $|$ is transitive. Also, if $a|b$ and $b|a$, then it must be the case that $a = b$. So $|$ is antisymmetric. Therefore,

$\langle \mathbb{N}, | \rangle$ is a poset.

But $\langle \mathbb{N}, | \rangle$ is not totally ordered because, for example, 2 and 3 are not comparable. To obtain a total order, we need to consider subsets of $\mathbb{N}$. For example, it's easy to see that for any m and n, either $2^m | 2^n$ or $2^n | 2^m$. Therefore,

$$\langle \{2^n \mid n \in \mathbb{N}\}, | \rangle \text{ is a totally ordered set.}$$

Let's consider some finite subsets of $\mathbb{N}$. For example, it's easy to see that

$$\langle \{1, 3, 9, 45\}, | \rangle \text{ is a totally ordered set.}$$

It's also easy to see that

$$\langle \{1, 2, 3, 4\}, | \rangle \text{ is a poset that is not totally ordered}$$

because 3 can't be compared to either 2 or 4.

end example

Notation for Partial Orders

When talking about partial orders, we'll often use the symbols

$$\prec \text{ and } \preceq$$

to stand for an irreflexive partial order and a reflexive partial order, respectively. We can read $a \prec b$ as "a is less than b," and we can read $a \preceq b$ as "a is less than or equal to b." The two symbols can be defined in terms of each other. For example, if $\langle A, \prec \rangle$ is a poset, then we can define the relation $\preceq$ in terms of $\prec$ by writing

$$\preceq = \prec \cup \{(x, x) \mid x \in A\}.$$

In other words, $\preceq$ is the reflexive closure of $\prec$. So $x \preceq y$ always means $x \prec y$ or $x = y$. Similarly, if $\langle B, \preceq \rangle$ is a poset, then we can define the relation $\prec$ in terms of $\preceq$ by writing

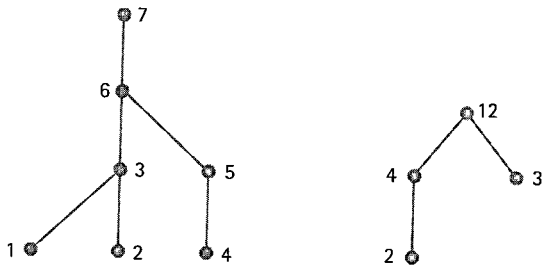
$$\prec = \preceq - \{(x, x) \mid x \in B\}.$$

Therefore, $x \prec y$ always means $x \preceq y$ and $x \neq y$. We also write the expression $y \succ x$ to mean the same thing as $x \prec y$.

Chains

A set of elements in a poset is called a *chain* if all the elements are comparable—linked—to each other. For example, any totally ordered set is itself a chain. A sequence of elements $x_1, x_2, x_3, \ldots$ in a poset is said to be a *descending chain* if $x_i \succ x_{i+1}$ for each $i \geq 1$. We can write the descending chain in the following familiar form:

$$x_1 \succ x_2 \succ x_3 \succ \cdots.$$



Pancake recipe (Example 4.30) $\langle\{2, 3, 4, 12\}, |\rangle$

Figure 4.13 Two poset diagrams.

For example, $4 > 2 > 0 > -2 > -4 > -6 > \ldots$ is a descending chain in $\langle \mathbb{Z}, < \rangle$. For another example, $\{a, b, c\} \supset \{a, b\} \supset \{a\} \supset \varnothing$ is a finite descending chain in $\langle \text{power}(\{a, b, c\}), \subseteq \rangle$. We can define an *ascending chain* of elements in a similar way. For example, $1 \mid 2 \mid 4 \mid \ldots \mid 2^n \mid \ldots$ is an ascending chain in the poset $\langle \mathbb{N}, | \rangle$.

Predecessors and Successors

If $x \prec y$, then we say that x is a *predecessor* of y, or y is a *successor* of x. Suppose that $x \prec y$ and there are no elements between x and y. In other words, suppose we have the following situation:

$$\{z \in A \mid x \prec z \prec y\} = \varnothing.$$

When this is the case, we say that x is an *immediate predecessor* of y, or y is an *immediate successor* of x. In a finite poset an element with a successor has an immediate successor. Some infinite posets also have this property. For example, every natural number x has an immediate successor $x + 1$ with respect to the "less" relation. But no rational number has an immediate successor with respect to the "less" relation.

Poset Diagrams

A poset can be represented by a special graph called a *poset diagram* or a *Hasse diagram*—after the mathematician Helmut Hasse (1898–1979). Whenever $x \prec y$ and x is an immediate predecessor of y, then place an edge (x, y) in the poset diagram with x at a lower level than y. A poset diagram can often help us observe certain properties of a poset. For example, the two poset diagrams in Figure 4.13 represent the pancake recipe poset from Example 4.30 and the poset $\langle\{2, 3, 4, 12\}, |\rangle$.

The three poset diagrams shown in Figure 4.14 are for the natural numbers and the integers with their usual orderings and for $\text{power}(\{a, b\})$ with the subset relation.

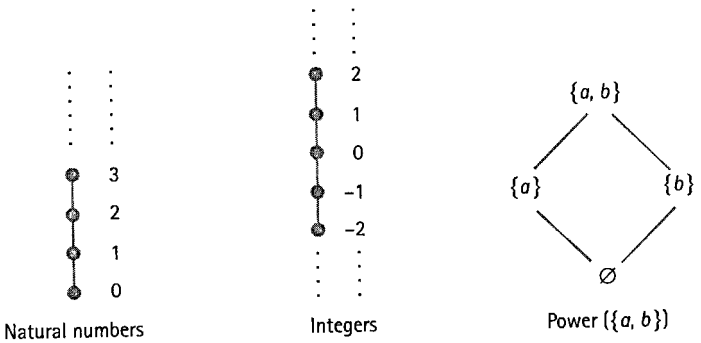


Figure 4.14 Three poset diagrams.

Maxima, Minima, and Bounds

When we have a partially ordered set, it's natural to use words like "minimal," "least," "maximal," and "greatest." Let's give these words some formal definitions.

Suppose that S is any nonempty subset of a poset P. An element $x \in S$ is called a *minimal element* of S if x has no predecessors in S. An element $x \in S$ is called the *least element* of S if $x \preceq y$ for all $y \in S$. For example, let's consider the poset $\langle \mathbb{N}, | \rangle$.

The subset $\{2, 4, 5, 10\}$ has two minimal elements, 2 and 5.
The subset $\{2, 4, 12\}$ has least element 2.
The set $\mathbb{N}$ has least element 1 because $1|x$ for all $x \in \mathbb{N}$.

For another example, let's consider the poset $\langle \text{power}(\{a, b, c\}), \subseteq \rangle$. The subset $\{\{a, b\}, \{a\}, \{b\}\}$ has two minimal elements, $\{a\}$ and $\{b\}$. The power set itself has least element $\varnothing$.

In a similar way we can define *maximal elements* and the *greatest element* of a subset of a poset. For example, let's consider the poset $\langle \mathbb{N}, | \rangle$.

The subset $\{2, 4, 5, 10\}$ has two maximal elements, 4 and 10.
The subset $\{2, 4, 12\}$ has greatest element 12.
The set $\mathbb{N}$ itself has greatest element 0 because $x|0$ for all $x \in \mathbb{N}$.

For another example, let's consider the poset $\langle \text{power}(\{a, b, c\}), \subseteq \rangle$. The subset $\{\varnothing, \{a\}, \{b\}\}$ has two maximal elements, $\{a\}$ and $\{b\}$. The power set itself has greatest element $\{a, b, c\}$.

Some sets may not have any minimal elements, yet still be bounded below by some element. For example, the set of positive rational numbers has no least element yet is bounded below by the number 0. Let's introduce some standard terminology that can be used to discuss ideas like this.

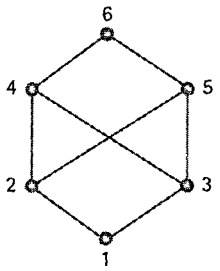


Figure 4.15 A poset diagram.

If S is a nonempty subset of a poset P, an element $x \in P$ is called a *lower bound* of S if $x \preceq y$ for all $y \in S$. An element $x \in P$ is called the *greatest lower bound* (or *glb*) of S if x is a lower bound and $z \preceq x$ for all lower bounds z of S. The expression $\text{glb}(S)$ denotes the greatest lower bound of S, if it exists. For example, if we let $\mathbb{Q}^+$ denote the set of positive rational numbers, then over the poset $\langle \mathbb{Q}, \leq \rangle$ we have $\text{glb}(\mathbb{Q}^+) = 0$.

In a similar way we define upper bounds for a subset S of the poset P. An element $x \in P$ is called an *upper bound* of S if $y \preceq x$ for all $y \in S$. An element $x \in P$ is called the *least upper bound* (or *lub*) of S if x is an upper bound and $x \preceq z$ for all upper bounds z of S. The expression $\text{lub}(S)$ denotes the least upper bound of S, if it exists. For example, $\text{lub}(\mathbb{Q}^+)$ does not exist in $\langle \mathbb{Q}, \leq \rangle$.

For another example, in the poset $\langle \mathbb{N}, \leq \rangle$, every finite subset has a glb—the least element—and a lub—the greatest element. Every infinite subset has a glb but no upper bound.

Can subsets have upper bounds without having a least upper bound? Sure. Here's an example.

example 4.32 Upper Bounds

Suppose the set $\{1, 2, 3, 4, 5, 6\}$ represents six time-oriented tasks. You can think of the numbers as chapters in a book, as processes to be executed on a computer, or as the steps in a recipe for making ice cream. In any case, suppose the tasks are partially ordered according to the poset diagram in Figure 4.15.

The subset $\{2, 3\}$ is bounded above, but it has no least upper bound. Notice that 4, 5, and 6 are all upper bounds of $\{2, 3\}$, but none of them is a least upper bound.

end example

Lattices

A *lattice* is a poset with the property that every pair of elements has a glb and a lub. So the poset of Example 4.32 is not a lattice. For example, $\langle \mathbb{N}, \leq \rangle$ is a

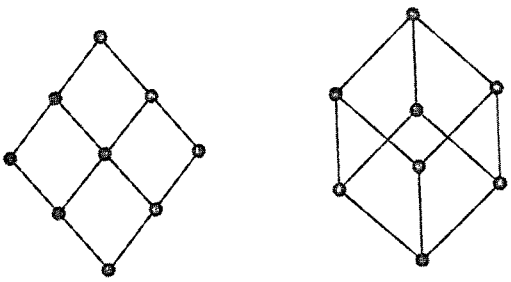


Figure 4.16 Two lattices.

lattice in which the glb of two elements is their minimum and the lub is their maximum. For another example, if A is any set, then $\langle \text{power}(A), \subseteq \rangle$ is a lattice, where $\text{glb}(X, Y) = X \cap Y$ and $\text{lub}(X, Y) = X \cup Y$. The word "lattice" is used because lattices that aren't totally ordered often have poset diagrams that look like "latticeworks" or "trellisworks."

For example, the two poset diagrams in Figure 4.16 represent lattices. These two poset diagrams can represent many different lattices. For example, the poset diagram on the left represents the lattice whose elements are the positive divisors of 36, ordered by the divides relation. In other words, it represents the lattice $\langle \{1, 2, 3, 4, 6, 9, 12, 18, 36\}, | \rangle$. See whether you can label the poset diagram with these numbers. The diagram on the right represents the lattice $\langle \text{power}(\{a, b, c\}), \subseteq \rangle$. It also represents the lattice whose elements are the positive divisors of 70, ordered by the divides relation. See whether you can label the poset diagram with both of these lattices. We'll give some more examples in the exercises.

4.3.2 Topological Sorting

A typical computing task is to sort a list of elements taken from a totally ordered set. Here's the problem statement.

The Sorting Problem
Find an algorithm to sort a list of elements from a totally ordered set.

For example, suppose we're given the list $\langle x_1, x_2, \ldots, x_n \rangle$, where the elements of the list are related by a total order relation R. We might sort the list by a program "sort," which we could call as follows:

$$\text{sort}(R, \langle x_1, x_2, \ldots, x_n \rangle).$$

For example, we should be able to obtain the following results with sort:

$$\text{sort}\,(<, \langle 8, 3, 10, 5 \rangle) = \langle 3, 5, 8, 10 \rangle\,,$$
$$\text{sort}\,(>, \langle 8, 3, 10, 5 \rangle) = \langle 10, 8, 5, 3 \rangle\,.$$

Programming languages normally come equipped with several totally ordered sets. If a total order R is not part of the language, then R must be implemented as a relational test, which can then be called into action whenever a comparison is required in the sorting algorithm.

Topological Sorting

Can a partially ordered set be sorted? The answer is yes if we broaden our idea of what sorting means. Here's the problem statement.

The Topological Sorting Problem

Find an algorithm to sort a list of elements from a partially ordered set.

How can we "sort" a list when some elements may not be comparable? Well, we try to find a listing that maintains the partial ordering, as in the pancake recipe from Example 4.30. Given a partial order R on a set, a list of elements from the set is *topologically sorted* if whenever two elements in the list satisfy $a \; R \; b$, then a is to the left of b in the list.

The ordering of a set of tasks is a topological sorting problem. For example, the list $\langle 4, 5, 2, 1, 3, 6, 7 \rangle$ is a topological sort of the steps in the pancake recipe from Example 4.30. Another example of a topological sort is the ordering of the chapters in a textbook in which the partial order is defined to be the dependence of one chapter upon another. In other words, we hope that we don't have to read some chapter further on in the book to understand what we're reading now.

Is there a technique to do topological sorting? Yes. Suppose R is a partial order on a finite set A. For each element $y \in A$, let $P(y)$ be the number of immediate predecessors of y, and let $S(y)$ be the set of immediate successors of y. Let *Sources* be the set of sources—minimal elements—in A. Therefore, y is a source if and only if $P(y) = 0$. A topological sort algorithm goes something like the following:

Topological Sort Algorithm **(4.17)**

While the set of sources is not empty, do the following steps:

1. Output a source y.
2. For all z in $S(y)$, decrement $P(z)$; if $P(z) = 0$, then add z to *Sources*.

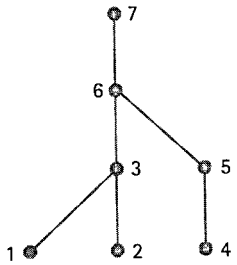


Figure 4.17 Poset of a pancake recipe.

A more detailed description of the algorithm can be given as follows:

Detailed Topological Sort

while *Sources* $\neq \varnothing$ **do**
  Pick a source y from *Sources;*
  Output y;
  for each z in $S(y)$ **do**
    $P(z) := P(z) - 1$;
    if $P(z) = 0$ **then** *Sources* $:=$ *Sources* $\cup \{z\}$
  od;
  Sources $:=$ *Sources* $- \{y\}$;
od

Let's do an example that includes some details on how the data for the algorithm might be represented.

example 4.33 A Topological Sort

We'll consider the steps of the pancake recipe from Example 4.30. Figure 4.17 shows the poset diagram for the steps of the recipe.

The initial set of sources is $\{1, 2, 4\}$. Letting P be an array of integers, we get the following initial table of predecessor counts:

i	1	2	3	4	5	6	7
$P(i)$	0	0	2	0	1	2	1

The following table is the initial table of successor sets S:

i	1	2	3	4	5	6	7
$S(i)$	$\{3\}$	$\{3\}$	$\{6\}$	$\{5\}$	$\{6\}$	$\{7\}$	$\varnothing$

You should trace the algorithm for these data representations.

end example

There is a very interesting and efficient implementation of algorithm (4.17) in Knuth [1968]. It involves the construction of a novel data structure to represent the set of sources, the sets $S(y)$ for each y, and the numbers $P(z)$ for each z.

4.3.3 Well-Founded Orders

Let's look at a special property of the natural numbers. Suppose we're given a descending chain of natural numbers that begins as follows:

$$29 > 27 > 25 > \cdots.$$

Can this descending chain continue forever? Of course not. We know that 0 is the least natural number, so the given chain must stop after only a finite number of terms. This is not an earthshaking discovery, but it is an example of the property of well-foundedness that we're about to discuss.

Definition of a Well-Founded Order

We're going to consider posets with the property that every descending chain of elements is finite. So we'll give these posets a name.

Well-Founded
A poset is said to be *well-founded* if every descending chain of elements is finite. In this case, the partial order is called a *well-founded order.*

For example, we've seen that $\mathbb{N}$ is a well-founded set with respect to the less relation $<$. In fact, any set of integers with a least element is well-founded by $<$. For example, the following three sets of integers are well-founded.

$$\{1, 2, 3, 4, \ldots\},\ \{m \mid m \geq -3\},\ \text{and}\ \{5, 9, 13, 17, \ldots\}.$$

For another example, any collection of finite sets is well-founded by $\subseteq$. This is easy to see because any descending chain must start with a finite set. If the set has n elements, it can start a descending chain of at most $n + 1$ subsets. For example, the following expression displays a longest descending chain starting with the set $\{a, b, c\}$.

$$\{a, b, c\} \supset \{b, c\} \supset \{c\} \supset \varnothing.$$

So the power set of a finite set is well-founded with respect to $\subseteq$.

But many posets are not well-founded. For example, the integers and the positive rationals are not well-founded with respect to the less relation because they have infinite descending chains as the following examples show.

$$2 > 0 > -2 > -4 > \cdots$$
$$\frac{1}{2} > \frac{1}{3} > \frac{1}{4} > \frac{1}{5} > \cdots.$$

The power set of an infinite set is not well-founded by $\subseteq$. For example, if we let $S_k = \mathbb{N} - \{0, 1, \ldots, k\}$, then we obtain the following infinite descending chain in power($\mathbb{N}$):

$$S_0 \supset S_1 \supset S_2 \supset \cdots \supset S_k \supset \cdots.$$

Are well-founded sets good for anything? The answer is yes. We'll see in the next section that they are basic tools for inductive proofs. So we should get familiar with them. We'll do this by looking at another property that well-founded sets possess.

The Minimal Element Property

Does every subset of $\mathbb{N}$ have a least element? A quick-witted person might say, "Yes," and then think a minute and say, "except that the empty set doesn't have any elements, so it can't have a least element." Suppose the question is modified to "Does every nonempty subset of $\mathbb{N}$ have a least element?" Then a bit of thought will convince most of us that the answer is yes.

We might reason as follows: Suppose S is some nonempty subset of $\mathbb{N}$ and x_1 is some element of S. If x_1 is the least element of S, then we are done. So assume that x_1 is not the least element of S. Then x_1 must have a predecessor x_2 in S—otherwise, x_1 would be the least element of S. If x_2 is the least element of S, then we are done. If x_2 is not the least element of S, then it has a predecessor x_3 in S, and so on. If we continue in this manner, we will obtain a descending chain of distinct elements in S:

$$x_1 > x_2 > x_3 > \cdots.$$

This looks familiar. We already know that this chain of natural numbers can't be infinite. So it stops at some value, which must be the least element of S. So every nonempty subset of the natural numbers has a least element.

This property is not true for all posets. For example, the set of integers has no least element. The open interval of real numbers (0, 1) has no least element. Also the power set of a finite set can have collections of subsets that have no least element.

Notice however that every collection of subsets of a finite set does contain a minimal element. For example, the collection $\{\{a\}, \{b\}, \{a, b\}\}$ has two minimal elements $\{a\}$ and $\{b\}$. Remember, the property that we are looking for must be

true for all well-founded sets. So the existence of least elements is out; it's too restrictive.

But what about the existence of minimal elements for nonempty subsets of a well-founded set? This property is true for the natural numbers. (Least elements are certainly minimal.) It's also true for power sets of finite sets. In fact, this property is true for all well-founded sets, and we can state the result as follows:

Descending Chains and Minimality **(4.18)**

If A is a well-founded set, then every nonempty subset of A has a minimal element. Conversely, if every nonempty subset of A has a minimal element, then A is well-founded.

It follows from (4.18) that the property of finite descending chains is equivalent to the property of nonempty subsets having minimal elements. In other words, if a poset has one of the properties, then it also has the other property. Thus it is also correct to define a well-founded set to be a poset with the property that every nonempty subset has a minimal element. We will call this latter property the *minimum condition* on a poset.[1]

Whenever a well-founded set is totally ordered, then each nonempty subset has a single minimal element, the least element. Such a set is called a *well-ordered set*. So a well-ordered set is a totally ordered set such that every nonempty subset has a least element. For example, $\mathbb{N}$ is well-ordered by the "less" relation. Let's examine a few more total orderings to see whether they are well-ordered.

Lexicographic Ordering of Tuples

The linear ordering $<$ on $\mathbb{N}$ can be used to create the *lexicographic* order on $\mathbb{N}^k$, which is defined as follows.

$$(x_1, \ldots, x_k) \prec (y_1, \ldots, y_k)$$

if and only if there is an index $j \geq 1$ such that $x_j < y_j$ and for each $i < j$, $x_i = y_i$. This ordering is a total ordering on $\mathbb{N}^k$. It's also a well-ordering.

For example, the lexicographic order on $\mathbb{N} \times \mathbb{N}$ has least element $(0, 0)$. Every nonempty subset of $\mathbb{N} \times \mathbb{N}$ has a least element, namely, the pair (x, y) with the smallest value of x, where y is the smallest value among second components of pairs with x as the first component. For example, $(0, 10)$ is the least element in the set $\{(0, 10), (0, 11), (1, 0)\}$. Notice that $(1, 0)$ has infinitely many predecessors of the form $(0, y)$, but $(1, 0)$ has no immediate predecessor.

[1] Other names for a well-founded set are *poset with minimum condition*, *poset with descending chain condition*, and *Artinian poset*, after Emil Artin, who studied algebraic structures with the descending chain condition. Some people use the term *Noetherian*, after Emmy Noether, who studied algebraic structures with the ascending chain condition.

Lexicographic Ordering of Strings

Another type of lexicographic ordering involves strings. To describe it we need to define the prefix of a string. If a string x can be written as $x = uv$ for some strings u and v, then u is called a *prefix* of x. If $v \neq \Lambda$, then u is a *proper prefix* of x. For example, the prefixes of the string aba over the alphabet $\{a, b\}$ are Λ, a, ab, and aba. The proper prefixes of aba are Λ, a, and ab.

Definition of Lexicographic Ordering

Let A be a finite alphabet with some agreed-upon linear ordering. Then the *lexicographic* ordering on A^* is defined as follows: $x \prec y$ iff either x is a proper prefix of y or x and y have a longest common proper prefix u such that $x = uv$, $y = uw$, and head(v) precedes head(w) in A.

The lexicographic ordering on A^* is often called the *dictionary ordering* because it corresponds to the ordering of words that occur in a dictionary. The definition tells us that if $x \neq y$, then either $x \prec y$ or $y \prec x$. So the lexicographic ordering on A^* is a total (i.e., linear) ordering. It also follows that every string x has an immediate successor xa, where a is the first letter of A.

If A has at least two elements, then the lexicographic ordering on A^* is *not* well-ordered. For example, let $A = \{a, b\}$ and suppose that a precedes b. Then the elements in the set $\{a^n b \mid n \in \mathbb{N}\}$ form an infinite descending chain:

$$b \succ ab \succ aab \succ aaab \succ \cdots \succ a^n b \succ \cdots .$$

Notice also that b has no immediate predecessor because if $x \prec b$, then we have $x \prec xa \prec b$.

Standard Ordering of Strings

Now let's look at an ordering that is well-ordered. The *standard ordering on strings* uses a combination of length and the lexicographic ordering.

Definition of Standard Ordering

Let A be a finite alphabet with some agreed-upon linear ordering. The standard ordering on A^* is defined as follows, where $\prec_L$ denotes the lexicographic ordering on A^*:

$x \prec y$ iff either length$(x) <$ length(y), or length$(x) =$ length(y) and $x \prec_L y$.

It's easy to see that $\prec$ is a total order and every string has an immediate successor and an immediate predecessor. The standard ordering on A^* is also well-ordered because each string has a finite number of predecessors. For example, let $A = \{a, b\}$ and suppose that a precedes b. Then the first few elements in the standard order of A^* are given as follows:

$$\Lambda, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \ldots .$$

Constructing Well-Founded Orderings

Collections of strings, lists, trees, graphs, or other structures that programs process can usually be made into well-founded sets by defining an appropriate order relation. For example, any finite set can be made into a well-founded set—actually a well-ordered set—by simply listing its elements in any order we wish, letting the leftmost element be the least element.

Let's look at some ways to build well-founded orderings for infinite sets. Suppose we want to define a well-founded order on some infinite set S. A simple and useful technique is to associate each element of S with some element in an existing well-founded set. For example, the natural numbers are well-founded by $<$. So we'll use them as a building block for well-founded constructions.

Constructing a Well-Founded Order **(4.19)**

Given any function $f : S \to \mathbb{N}$, there is a well-founded order $\prec$ defined on S in the following way, where $x, y \in S$:

$$x \prec y \quad \text{means} \quad f(x) < f(y).$$

Does the new relation $\prec$ make S into a well-founded set? Sure. Suppose we have a descending chain of elements in S as follows:

$$x_1 \succ x_2 \succ x_3 \succ \cdots.$$

The chain must stop because $x \succ y$ is defined to mean $f(x) > f(y)$, and we know that any descending chain of natural numbers must stop. Let's look at a few more examples.

example **4.34 Some Well-Founded Orderings**

a. Any set of lists, where $L \prec M$ means $\text{length}(L) < \text{length}(M)$.

b. Any set of strings, where $s \prec t$ means $\text{length}(s) < \text{length}(t)$.

c. Any set of trees, where $B \prec C$ means the number of nodes in B is less than the number of nodes in C.

d. Any set of trees, where $B \prec C$ means the number of leaves in B is less than the number of leaves in C.

e. Any set of nonempty trees, where $B \prec C$ means $\text{depth}(B) < \text{depth}(C)$.

f. Any set of people can be well-founded by the age at the last birthday of each person. What are the minimal elements?

g. The set $\{\ldots, -3, -2, -1\}$ of negative integers, where $x \prec y$ means $x > y$.

end example

As the examples show, we can use known properties of structures to find useful well-founded orderings for sets of structures. The next example constructs a finite, hence well-founded, lexicographic order.

example **4.35 A Finite Lexicographic Order**

Let $S = \{0, 1, 2, \ldots, m\}$. Then we can define a lexicographic ordering on the set S^k in a natural way. Since S is finite, it follows that the lexicographic ordering on S^k is well-founded. The least element is $(0, \ldots, 0)$, and the greatest element is $(m, \ldots, m)$. For example, if $k = 3$, then the immediate successor of any element can be defined as

$$\begin{aligned}\text{succ}\,((x, y, z)) = \text{if } z < m \text{ then } &(x, y, z + 1)\\ \text{else if } y < m \text{ then } &(x, y + 1, 0)\\ \text{else if } x < m \text{ then } &(x + 1, 0, 0)\\ \text{else error (no successor).}&\end{aligned}$$

end example

Inductively Defined Sets are Well-Founded

It's easy to make an inductively defined set W into a well-founded set if no two elements are defined in terms of each other. We'll give two methods. Both methods let the basis elements of W be the minimal elements of the well-founded order.

Method 1: (4.20)

Define a function $f : W \to \mathbb{N}$ as follows:

1. $f(c) = 0$ for all basis elements c of W.
2. If $x \in W$ and x is constructed from elements $y_1, y_2, \ldots, y_n$ in W, then define $f(x) = 1 + \max\{f(y_1), f(y_2), \ldots, f(y_n)\}$.

Let $x \prec y$ mean $f(x) < f(y)$.

Since 0 is the least element of $\mathbb{N}$ and $f(c) = 0$ for all basis elements c of W, it follows that the basis elements of W are minimal elements under the ordering defined by (4.20). For example, if c is a basis element of W and if $x \prec c$, then $f(x) < f(c) = 0$, which can't happen with natural numbers. Therefore c is a minimal element of W.

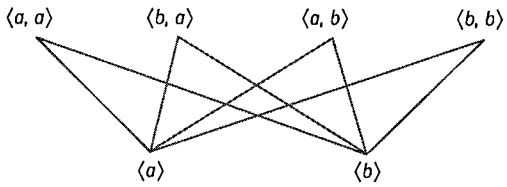


Figure 4.18 Part of a poset diagram.

example **4.36 A Well-Founded Ordering**

Let W be the set of all nonempty lists over $\{a, b\}$. First we'll give an inductive definition of W. The lists $\langle a \rangle$ and $\langle b \rangle$ are the basis elements of W. For the induction case, if $L \in W$, then the lists cons(a, L) and cons(b, L) are in W. Now we'll use (4.20) to make W into a well-founded set. The function f of (4.20) turns out to be $f(L) = \text{length}(L) - 1$. So for any lists L and M in W, we define $L \prec M$ to mean $f(L) < f(M)$, which means $\text{length}(L) - 1 < \text{length}(M) - 1$, which also means $\text{length}(L) < \text{length}(M)$. The diagram in Figure 4.18 shows the bottom two layers of a poset diagram for W with its two minimal lists $\langle a \rangle$ and $\langle b \rangle$.

end example

Method 1 can relate many elements. For example, to add one more level to the diagram in Figure 4.18, we have to include eight 3-element lists and draw 32 lines from the two element lists up to the three element lists. Sometimes it isn't necessary to have an ordering that relates so many elements. This brings us to the second method for defining a well-founded ordering on an inductively defined set W.

Method 2: (4.21)

The ordering $\prec$ is defined as follows:

1. Let the basis elements of W be minimal elements.
2. If $x \in W$ and x is constructed from elements $y_1, y_2, \ldots, y_n$ in W, then define $y_i \prec x$ for each $i = 1, \ldots, n$.

The actual ordering is the transitive closure of $\prec$.

The ordering of (4.21) is well-founded because any x can be constructed from basis elements with finitely many constructions. Therefore, there can be no

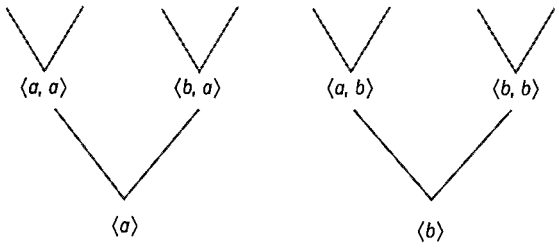


Figure 4.19 Part of a poset diagram.

infinite descending chain starting at x. With this ordering, there can be many pairs that are not related.

For example, we'll use the preceding example of nonempty lists over the set $\{a, b\}$. The picture in Figure 4.19 shows the bottom two levels of the poset diagram for the well-founded ordering constructed by (4.21).

Notice that each list has only two immediate successors. For example, the two successors of $\langle a\rangle$ are $\text{cons}(a, \langle a\rangle) = \langle a, a\rangle$ and $\text{cons}(b, \langle a\rangle) = \langle b, a\rangle$. The two successors of $\langle b, a\rangle$ are $\langle a, b, a\rangle$ and $\langle b, b, a\rangle$. This is much simpler than the ordering we got using (4.20).

Let's look at some examples of inductively defined sets that are well-founded sets by the method of (4.21).

example 4.37 Using One Part of a Product

We'll define the set $\mathbb{N} \times \mathbb{N}$ inductively by using the first copy of $\mathbb{N}$. For the basis case we put $(0, n) \in \mathbb{N} \times \mathbb{N}$ for all $n \in \mathbb{N}$. For the induction case, whenever the pair $(m, n) \in \mathbb{N} \times \mathbb{N}$, we put $(m+1, n) \in \mathbb{N} \times \mathbb{N}$. The relation on $\mathbb{N} \times \mathbb{N}$ induced by this inductive definition and (4.21) is not linearly ordered.

For example, $(0, 0)$ and $(0, 1)$ are not related because they are both basis elements. Notice that any pair (m, n) is the beginning of a descending chain containing at most $m + 1$ pairs. For example, the following chain is the longest descending chain that starts with $(3, 17)$.

$$(3, 17), (2, 17), (1, 17), (0, 17).$$

end example

example 4.38 Using Both Parts of a Product

Let's define the set $\mathbb{N} \times \mathbb{N}$ inductively by using both copies of $\mathbb{N}$. The single basis element is $(0, 0)$. For the induction case, if $(m, n) \in \mathbb{N} \times \mathbb{N}$, then put $(m + 1, n), (m, n + 1) \in \mathbb{N} \times \mathbb{N}$. Notice that each pair with both components nonzero is defined twice by this definition. The relation induced by this definition and (4.21) is nonlinear.

For example, the two pairs (2, 1) and (1, 2) are not related. Any pair (m, n) is the beginning of a descending chain of at most $m + n + 1$ pairs. For example, the following descending chain has maximum length among the descending chains that start at the pair (2, 3).

$$(2, 3), (2, 2), (1, 2), (1, 1), (0, 1), (0, 0).$$

Can you find a different chain of the same length starting at (2, 3)?

end example

4.3.4 Ordinal Numbers

We'll finish our discussion of order by introducing the *ordinal numbers.* These numbers are ordered, and they can be used to count things. An ordinal number may be represented as a set with certain properties. For example, any ordinal number x has an immediate successor defined by $\text{succ}(x) = x \cup \{x\}$. The expression $x + 1$ is also used to denote $\text{succ}(x)$. The natural numbers denote ordinal numbers when we define $0 = \varnothing$ and interpret $+$ as addition, in which case it's easy to see that

$$x + 1 = \{0, \ldots, x\}.$$

For example, $1 = \{0\}$, $2 = \{0, 1\}$, and $5 = \{0, 1, 2, 3, 4\}$. In this way, each natural number is an ordinal number, called a *finite ordinal.*

Now let's define some *infinite ordinals.* The first infinite ordinal is

$$\omega = \{0, 1, 2, \ldots\},$$

the set of natural numbers. The next infinite ordinal is

$$\omega + 1 = \text{succ}(\omega) = \omega \cup \{\omega\} = \{\omega, 0, 1, \ldots\}.$$

If α is an ordinal number, we'll write $\alpha + n$ in place of $\text{succ}^n(\alpha)$. So the first four infinite ordinals are ω, $\omega + 1$, $\omega + 2$, and $\omega + 3$. The infinite ordinals continue in this fashion. To get beyond this sequence of ordinals, we need to make a definition similar to the one for ω. The main idea is that any ordinal number is the union of all its predecessors. For example, we define $\omega 2 = \omega \cup \{\omega, \omega + 1, \ldots\}$. The ordinals continue with $\omega 2 + 1$, $\omega 2 + 2$, and so on. Of course, we can continue and define $\omega 3 = \omega 2 \cup \{\omega 2, \omega 2 + 1, \ldots\}$. After ω, $\omega 2$, $\omega 3, \ldots$ comes the ordinal ω^2. Then we get $\omega^2 + 1$, $\omega^2 + 2, \ldots$, and we eventually get $\omega^2 + \omega$. Of course, the process goes on forever.

We can order the ordinal numbers by defining $\alpha < \beta$ iff $\alpha \in \beta$. For example, we have $x < x + 1$ for any ordinal x because $x \in \text{succ}(x) = x + 1$. So we get the familiar ordering $0 < 1 < 2 < \ldots$ for the finite ordinals. For any finite ordinal n we have $n < \omega$ because $n \in \omega$. Similarly, we have $\omega < \omega + 1$, and for any finite ordinal n we have $\omega + n < \omega 2$. So it goes. There are also uncountable ordinals, the least of which is denoted by Ω. And the ordinals continue on after this too.

Although every ordinal number has an immediate successor, there are some ordinals that don't have any immediate predecessors. These ordinals are called *limit ordinals* because they are defined as "limits" or unions of all their predecessors. The limit ordinals that we've seen are

$$0, \omega, \omega 2, \omega 3, \ldots, \omega^2, \ldots, \Omega, \ldots.$$

An interesting fact about ordinal numbers states that for any set S there is a bijection between S and some ordinal number. For example, there is a bijection between the set $\{a, b, c\}$ and the ordinal number $3 = \{0, 1, 2\}$. For another example there are bijections between the set $\mathbb{N}$ of natural numbers and each of the ordinals $\omega, \omega + 1, \omega + 2, \ldots$. Some people define the cardinality of a set to be the least ordinal number that is bijective to the set. So we have $|\{a, b, c\}| = 3$ and $|\mathbb{N}| = \omega$.

More information about ordinal numbers—including ordinal arithmetic—can be found in the excellent book by Halmos [1960].

Exercises

Partial Orders

1. Sometimes our intuition about a symbol can be challenged. For example, suppose we define the relation $\prec$ on the integers by saying that $x \prec y$ means $|x| < |y|$. Assign the value True or False to each of the following statements.

   a. $-7 \prec 7$. b. $-7 \prec -6$. **c.** $6 \prec -7$. d. $-6 \prec 2$.

2. State whether each of the following relations is a partial order.

   a. isFatherOf. b. isAncestorOf. **c.** isOlderThan.
   d. isSisterOf. **e.** $\{(a, b), (a, a), (b, a)\}$. f. $\{(2, 1), (1, 3), (2, 3)\}$.

3. Draw a poset diagram for each of the following partially ordered relations.

   a. $\{(a, a), (a, b), (b, c), (a, c), (a, d)\}$.
   b. power($\{a, b, c\}$), with the subset relation.
   c. lists($\{a, b\}$), where $L \prec M$ if length(L) < length(M).
   d. The set of all binary trees over the set $\{a, b\}$ that contain either one or two nodes. Let $s \prec t$ mean that s is either the left or right subtree of t.

4. Suppose we wish to evaluate the following expression as a set of time-oriented tasks:

$$(f(x) + g(x))(f(x)g(x)).$$

We'll order the subexpressions by data dependency. In other words, an expression can't be evaluated until its data are available. So the subexpressions that occur in the evaluation process are

$$x,\ f(x),\ g(x),\ f(x) + g(x),\ f(x)g(x),\ \text{and}\ (f(x) + g(x))(f(x)g(x)).$$

Draw the poset diagram for the set of subexpressions. Is the poset a lattice?

5. For any positive integer n, let D_n be the set of positive divisors of n. The poset $\langle D_n, | \rangle$ is a lattice. Describe the glb and lub for any pair of elements.

Well-Founded Property

6. Why is it true that every partially ordered relation over a finite set is well-founded?

7. For each set S, show that the given partial order on S is well-founded.

a. Let S be a set of trees. Let $s \prec t$ mean that s has fewer nodes than t.

b. Let S be a set of trees. Let $s \prec t$ mean that s has fewer leaves than t.

c. Let S be a set of lists. Let $L \prec M$ mean that $\text{length}(L) < \text{length}(M)$.

8. Example 4.38 discussed a well-founded ordering for the set $\mathbb{N} \times \mathbb{N}$. Use this ordering to construct two distinct descending chains that start at the pair (4, 3), both of which have maximum length.

9. Suppose we define the relation $\prec$ on $\mathbb{N} \times \mathbb{N}$ as follows:

$$(a, b) \prec (c, d) \quad \text{if and only if} \quad \max\{a, b\} < \max\{c, d\}.$$

Is $\mathbb{N} \times \mathbb{N}$ well-founded with respect to $\prec$?

Topological Sorting

10. Trace the topological sort algorithm (4.17) for the pancake recipe in Example 4.30 by starting with the source 1. There are several possible answers because any source can be output by the algorithm.

11. Describe a way to perform a topological sort that uses an adjacency matrix to represent the partial order.

Proofs and Challenges

12. Show that the two properties irreflexive and transitive imply the antisymmetric property. So an irreflexive partial order can be defined by just the two properties irreflexive and transitive.

13. Prove the two statements of (4.18).

14. For a poset P, a function $f : P \to P$ is said to be *monotonic* if $x \preceq y$ implies $f(x) \preceq f(y)$ for all $x, y \in P$. For each poset and function definition, determine whether the function is monotonic.
    a. $\langle \mathbb{N}, < \rangle$, $f(x) = 2x + 3$.
    b. $\langle \mathbb{N}, < \rangle$, $f(x) = x^2$.
    c. $\langle \mathbb{Z}, < \rangle$, $f(x) = x^2$.
    d. $\langle \mathbb{N}, | \rangle$, $f(x) = 2x + 3$.
    e. $\langle \mathbb{N}, | \rangle$, $f(x) = x^2$.
    f. $\langle \mathbb{N}, | \rangle$, $f(x) = x \bmod 5$.
    g. $\langle \text{power}(A), \subseteq \rangle$ for some set A, $f(X) = A - X$.
    h. $\langle \text{power}(\mathbb{N}), \subseteq \rangle$, $f(X) = \{n \in \mathbb{N} \mid n \text{ divides } x \text{ for some } x \in X\}$.

4.4 Inductive Proof

When discussing properties of things we deal not only with numbers, but also with structures such as strings, lists, trees, graphs, programs, and more complicated structures constructed from them. Do the objects that we construct have the properties that we expect? Does a program halt when it's supposed to halt and give the proper answer?

To answer these questions, we must find ways to reason about the objects that we construct. This section concentrates on a powerful proof technique called inductive proof. We'll see that the technique springs from the idea of a well-founded set that we discussed in Section 4.3.

4.4.1 Proof by Mathematical Induction

Suppose we want to find the value of the sum $2 + 4 + \cdots + 2n$ for any natural number n. Consider the following two programs written by two different students to calculate this sum:

$$f(n) = \text{if } n = 0 \text{ then } 0 \text{ else } f(n-1) + 2n$$
$$g(n) = n(n+1).$$

Are these programs correct? That is, do they both compute the correct value of the sum $2 + 4 + \cdots + 2n$? We can test a few cases such as $n = 0$, $n = 1$, $n = 2$ until we feel confident that the programs are correct. Or maybe we just can't get any feeling of confidence in these programs. Is there a way to prove, once and for all, that these programs are correct for all natural numbers n? Let's look at the second program. If it's correct, then the following equation must be true for all natural numbers n:

$$2 + 4 + \cdots + 2n = n(n+1).$$

Certainly we don't have the time to check it for the infinity of natural numbers. Is there some other way to prove it? Happily, we will be able to prove the infinitely many cases in just two steps with a technique called *proof by induction,* which

we discuss next. If you don't want to see why it works, you can skip ahead to (4.23).

A Basis for Mathematical Induction

Interestingly, the technique that we present is based on the fact that any nonempty subset of the natural numbers has a least element. Recall that this is the same as saying that any descending chain of natural numbers is finite. In fact, this is just a statement that $\mathbb{N}$ is a well-founded set. In fact we can generalize a bit. Let m be an integer, and let W be the following set.

$$W = \{m, m + 1, m + 2, \dots\}.$$

Every nonempty subset of W has a least element. Let's see whether this property can help us find a tool to prove infinitely many things in just two steps. First, we state the following result, which forms a basis for the inductive proof technique.

A Basis for Mathematical Induction (4.22)

Let $m \in \mathbb{Z}$ and $W = \{m, m + 1, m + 2, \dots\}$. Let S be a subset of W such that the following two conditions hold.

1. $m \in S$.
2. Whenever $k \in S$, then $k + 1 \in S$.

Then $S = W$.

Proof: We'll prove $S = W$ by contradiction. Suppose $S \neq W$. Then $W - S$ has a least element x because every nonempty subset of W has a least element. The first condition of (4.22) tells us that $m \in S$. So it follows that $x > m$. Thus $x - 1 \geq m$, and it follows that $x - 1 \in S$. Thus we can apply the second condition to obtain $(x - 1) + 1 \in$ S. In other words, we are forced to conclude that $x \in S$. This is a contradiction, since we can't have both $x \in S$ and $x \in W - S$ at the same time. Therefore $S = W$. QED.

We should note that there is an alternative way to think about (4.22). First, notice that W is an inductively defined set. The basis case is $m \in W$. The inductive step states that whenever $k \in W$, then $k + 1 \in W$. Now we can appeal to the closure part of an inductive definition, which can be stated as follows: If S is a subset of W and S satisfies the basis and inductive steps for W, then $S = W$. From this point of view, (4.22) is just a restatement of the closure part of the inductive definition of W.

The Principle of Mathematical Induction

Let's put (4.22) into a practical form that can be used as a proof technique for proving that infinitely many cases of a statement are true. The technique is called the *principle of mathematical induction,* which we state as follows.

The Principle of Mathematical Induction (4.23)

Let $m \in \mathbb{Z}$. To prove that $P(n)$ is true for all integers $n \geq m$, perform the following two steps:

1. Prove that $P(m)$ is true.
2. Assume that $P(k)$ is true for an arbitrary $k \geq m$. Prove that $P(k+1)$ is true.

Proof: Let $W = \{n \mid n \geq m\}$, and let $S = \{n \mid n \geq m \text{ and } P(n) \text{ is true}\}$. Assume that we have performed the two steps of (4.23). Then S satisfies the hypothesis of (4.22). Therefore $S = W$. So $P(n)$ is true for all $n \geq m$. QED.

The principle of mathematical induction contains a technique to prove that infinitely many statements are true in just two steps. This proof technique is just what we need to prove our opening example about computing a sum of even natural numbers.

example 4.39 A Correct Formula

Let's prove that the following equation is true for all natural numbers $n \geq 1$:

$$2 + 4 + \cdots + 2n = n(n+1).$$

Proof: To see how to use (4.23), we can let $P(n)$ denote the above equation. Now we need to perform two steps. First, we have to show that $P(1)$ is true. Second, we have to assume that $P(k)$ is true and then prove that $P(k+1)$ is true. When $n = 1$, the equation becomes the true statement

$$2 = 1(1+1).$$

Therefore, $P(1)$ is true. Now assume that $P(k)$ is true. This means that we assume that the following equation is true:

$$2 + 4 + \cdots + 2k = k(k+1).$$

To prove that $P(k+1)$ is true, start on the left side of the equation for the expression $P(k+1)$:

$$\begin{aligned} 2+4+\cdots+2k+2(k+1) &= (2+4+\cdots+2k)+2(k+1) && \text{(associate)} \\ &= k(k+1)+2(k+1) && \text{(assumption)} \\ &= (k+1)(k+2) \\ &= (k+1)[(k+1)+1]. \end{aligned}$$

The last term is the right-hand side of $P(k+1)$. Thus $P(k+1)$ is true. So we have performed both steps of (4.23). Therefore, $P(n)$ is true for all natural numbers $n \geq 1$. QED.

end example

example **4.40 A Correct Recursively Defined Function**

We'll show that the following function computes $2+4+\cdots+2n$ for any natural number n:

$$f(n) = \text{if } n = 0 \text{ then } 0 \text{ else } f(n-1) + 2n.$$

Proof: For each $n \in \mathbb{N}$, let $P(n)$ be the statement "$f(n) = 2+4+\cdots+2n$." We want to show that $P(n)$ is true for all $n \in \mathbb{N}$. To start, notice that $f(0) = 0$. Thus $P(0)$ is true. Now assume that $P(k)$ is true for an arbitrary $k \geq 0$. To prove that $P(k+1)$ is true, we'll start on the left side of $P(k+1)$ to obtain

$$\begin{aligned} f(k+1) &= f(k+1-1)+2(k+1) && \text{(definition of } f\text{)} \\ &= f(k)+2(k+1) \\ &= (2+4+\cdots+2k)+2(k+1) && \text{(assumption)} \\ &= 2+4+\cdots+2(k+1). \end{aligned}$$

The last term is the right-hand side of $P(k+1)$. Therefore, $P(k+1)$ is true. So we have performed both steps of (4.23). It follows that $P(n)$ is true for all $n \in \mathbb{N}$. In other words, $f(n) = 2+4+\cdots+2n$ for all $n \in \mathbb{N}$. QED.

end example

A Classic Example: Arithmetic Progressions

When Gauss—mathematician Karl Friedrich Gauss (1777–1855)—was a 10-year-old boy, his schoolmaster, Buttner, gave the class an arithmetic progression of numbers to add up to keep them busy. We should recall that an *arithmetic progression* is a sequence of numbers where each number differs from its successor by the same constant. Gauss wrote down the answer just after Buttner finished

writing the problem. Although the formula was known to Buttner, no child of 10 had ever discovered it.

For example, suppose we want to add up the seven numbers in the following arithmetic progression:

$$3, 7, 11, 15, 19, 23, 27.$$

The trick is to notice that the sum of the first and last numbers, which is 30, is the same as the sum of the second and next to last numbers, and so on. In other words, if we list the numbers in reverse order under the original list, each column totals to 30.

3	7	11	15	19	23	27
27	23	19	15	11	7	3
30	30	30	30	30	30	30

If S is the sum of the progression, then $2S = 7(30) = 210$. So $S = 105$.

The Sum of an Arithmetic Progression

The example illustrates a use of the following formula for the sum of an arithmetic progression of n numbers $a_1, a_2, \ldots, a_n$.

Sum of an Arithmetic Progression (4.24)

$$a_1 + a_2 + \cdots + a_n = \frac{n(a_1 + a_n)}{2}.$$

Proof: We'll prove it by induction. Let $P(n)$ denote Equation (4.24). We'll show that $P(n)$ is true for all natural numbers $n \geq 1$. Starting with $P(1)$, we obtain

$$a_1 = \frac{(a_1 + a_1)}{2}.$$

Since this equation is true, $P(1)$ is true. Next we'll assume that $P(n)$ is true, as stated in (4.24), and try to prove the statement $P(n + 1)$, which is

$$a_1 + a_2 + \cdots + a_n + a_{n+1} = \frac{(n+1)(a_1 + a_{n+1})}{2}.$$

Since the progression $a_1, a_2, \ldots, a_{n+1}$ is arithmetic, there is a constant d such that it can be written in the following form, where $a = a_1$.

$$a, a + d, a + 2d, \ldots, a + nd.$$

In other words, $a_k = a + (k-1)d$ for $1 \leq k \leq n+1$. Starting with the left-hand side of the equation, we obtain

$$\begin{aligned} a_1 + a_2 + \cdots + a_n + a_{n+1} &= (a_1 + a_2 + \cdots + a_n) + a_{n+1} \\ &= \frac{n(a_1 + a_n)}{2} + a_{n+1} && \text{(induction)} \\ &= \frac{n(a + a + (n-1)d)}{2} + (a + nd) && \text{(write in terms of } d\text{)} \\ &= \frac{2na + n(n-1)d + 2a + 2nd}{2} \\ &= \frac{2a(n+1) + (n+1)nd}{2} \\ &= \frac{(n+1)(2a + nd)}{2} \\ &= \frac{(n+1)(a_1 + a_{n+1})}{2}. \end{aligned}$$

Therefore, $P(n+1)$ is true. So by (4.23), Equation (4.24) is correct for all arithmetic progressions of n numbers. QED.

We should observe that (4.24) can be used to calculate the sum of the arithmetic progression $2, 4, \ldots, 2n$ in Example 4.38. The best known arithmetic progression is $1, 2, \ldots, n$ and we can use (4.24) to calculate its sum.

A Well-Known Sum **(4.25)**

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

A Classic Example: Geometric Progressions

A *geometric progression* is a sequence of numbers where the ratio of each number and its successor is the same constant, called the common ratio. For example, the following sequence is a geometric progression:

$$3, 6, 12, 24, 48, 96.$$

The common ratio for this progression is 2 and we can write it in the following form.

$$3, 3 \cdot 2, 3 \cdot 2^2, 2 \cdot 2^3, 2 \cdot 2^4, 2 \cdot 2^5.$$

Any geometric progression beginning with the number a can be written in the following form, where r is the common ratio.

$$a, ar, ar^2, ar^3, \ldots, ar^n.$$

The common ratio can be negative, too. For example, the following progression is geometric with $a = 3$ and $r = -2$:

$$3, -6, 12, -24, 48, -96.$$

The Sum of a Geometric Progression

We're interested in a formula for the sum of a geometric progression a, ar, ar^2, $\ldots$, ar^n, where $r \neq 1$. In other words, we want a formula for the sum

$$a + ar + ar^2 + \cdots + ar^n.$$

We can find a formula for the sum by multiplying the given expression by the term $r - 1$ to obtain the equation

$$(r-1)(a + ar + ar^2 + \cdots + ar^n) = a(r^{n+1} - 1).$$

Now divide both sides by $r - 1$ to obtain the following formula for the sum of a geometric progression, where $r \neq 1$.

The Sum of a Geometric Progression (4.26)

$$a + ar + ar^2 + \cdots + ar^n = \frac{a(r^{n+1} - 1)}{r - 1}.$$

We'll give an induction proof of the formula.

Proof: If $n = 0$, then both sides are a. So assume that the formula is true for n, and prove that it is true for $n + 1$. Starting with the left-hand side, we have

$$\begin{aligned} a + ar + ar^2 + \cdots + ar^n + ar^{n+1} &= (a + ar + ar^2 + \cdots + ar^n) + ar^{n+1} \\ &= \frac{a(r^{n+1} - 1)}{r - 1} + ar^{n+1} \\ &= \frac{a(r^{n+1} - 1) + (r-1)ar^{n+1}}{r - 1} \\ &= \frac{a(r^{(n+1)+1} - 1)}{r - 1}. \end{aligned}$$

Thus, by (4.23), the formula is true for all natural numbers n. QED.

The most popular geometric progression starts with $a = 1$. In this case we obtain the following sum.

A Well-Known Sum (4.27)

$$1 + r + r^2 + \cdots + r^n = \frac{r^{n+1} - 1}{r - 1}.$$

Sometimes, (4.23) does not have enough horsepower to do the job. For example, we might need to assume more than is allowed by (4.23), or we might be dealing with structures that are not numbers, such as lists, strings, or binary trees, and there may be no easy way to apply (4.23). The solution to many of these problems is a stronger version of induction based on well-founded sets. That's next.

4.4.2 Proof by Well-Founded Induction

Let's extend the idea of inductive proof to well-founded sets. Recall that a well-founded set is a poset whose nonempty subsets have minimal elements or, equivalently, every descending chain of elements is finite. We'll start by noticing an easy extension of (4.22) to the case of well-founded sets. If you aren't interested in why the method works, you can skip ahead to (4.29).

The Basis of Well-Founded Induction **(4.28)**

Let W be a well-founded set, and let S be a subset of W satisfying the following two conditions.

1. S contains all the minimal elements of W.
2. Whenever an element $x \in W$ has the property that all its predecessors are elements of S, then $x \in S$.

Then $S = W$.

Proof: The proof is by contradiction. Suppose $S \neq W$. Then $W - S$ has a minimal element x. Since x is a minimal element of $W - S$, each predecessor of x cannot be in $W - S$. In other words, each predecessor of x must be in S. The second condition now forces us to conclude that $x \in S$. This is a contradiction, since we can't have both $x \in S$ and $x \in W - S$ at the same time. Therefore, $S = W$. QED.

You might notice that Condition 1 of (4.28) was not used in the proof. This is because it's a consequence of Condition 2 of (4.28). We'll leave this as an exercise (something about an element that doesn't have any predecessors). Condition 1 is stated explicitly because it helps to understand the ideas, and students are advised to begin an inductive proof by establishing it separately.

The Technique of Well-Founded Induction

Let's find a more practical form of (4.28) that gives us a technique for proving a collection of statements of the form $P(x)$ for each x in a well-founded set W. The technique is called *well-founded induction.*

Well-Founded Induction (4.29)

Let $P(x)$ be a statement for each x in the well-founded set W. To prove $P(x)$ is true for all $x \in W$, perform the following two steps:

1. Prove that $P(m)$ is true for all minimal elements $m \in W$.
2. Let x be an arbitrary element of W, and assume that $P(y)$ is true for all elements y that are predecessors of x. Prove that $P(x)$ is true.

Proof: Let $S = \{x \mid x \in W \text{ and } P(x) \text{ is true}\}$. Assume that we have performed the two steps of (4.29). Then S satisfies the hypothesis of (4.28). Therefore $S = W$. In other words, $P(x)$ is true for all $x \in W$. QED.

Second Principle of Mathematical Induction

Now we can state a corollary of (4.29), which lets us make a bigger assumption than we were allowed in (4.23). The principle is sometimes called "course-of-values induction."

Second Principle of Mathematical Induction (4.30)

Let $m \in \mathbb{Z}$. To prove that $P(n)$ is true for all integers $n \geq m$, perform the following two steps:

1. Prove that $P(m)$ is true.
2. Assume that n is an arbitrary integer $n > m$, and assume that $P(k)$ is true for all k in the interval $m \leq k < n$. Prove that $P(n)$ is true.

Proof: Let $W = \{n \mid n \geq m\}$. Notice that W is a well-founded set (actually well-ordered) whose least element is m. Let $S = \{n \mid n \in W \text{ and } P(n) \text{ is true}\}$. Assume that Steps 1 and 2 have been performed. Then $m \in S$, and if $n > m$ and all predecessors of n are in S, then $n \in S$. Therefore, $S = W$, by (4.29). QED.

example 4.41 Products of Primes

We'll prove the following well-known result about prime numbers.

Every natural number $n \geq 2$ is prime or a product of prime numbers.

Proof: For $n \geq 2$, let $P(n)$ be the statement "n is prime or a product of prime numbers." We need to show that $P(n)$ is true for all $n \geq 2$. Since 2 is prime, it follows that $P(2)$ is true. So Step 1 of (4.30) is finished. For Step 2 we'll assume that $n > 2$ and $P(k)$ is true for $2 \leq k < n$. With this assumption we must show that $P(n)$ is true. If n is prime, then $P(n)$ is true. So assume that n is not prime.

Then $n = xy$, where $2 \leq x < n$ and $2 \leq y < n$. By our assumption, $P(x)$ and $P(y)$ are both true, which means that x and y are products of primes. Therefore, n is a product of primes. So $P(n)$ is true. Now (4.30) implies that $P(n)$ is true for all $n \geq 2$. QED.

Notice that we can't use (4.23) for the proof because its induction assumption is the single statement that $P(n-1)$ is true. We need the stronger assumption that $P(k)$ is true for $2 \leq k < n$ to allow us to say that $P(x)$ and $P(y)$ are true.

end example

Things You Must Do

Let's pause and make a few comments about inductive proof. Remember, when you are going to prove something with an inductive proof technique, there are always two distinct steps to be performed. First prove the basis case, showing that the statement is true for each minimal element. Now comes the second step. The most important part about this step is making an assumption. Let's write it down for emphasis.

You are required to make an assumption in the inductive step of a proof.

Some people find it hard to make assumptions. But inductive proof techniques require it. So if you find yourself wondering about what to do in an inductive proof, here are two questions to ask yourself: "Have I made an induction assumption?" If the answer is yes, ask the question, "Have I used the induction assumption in my proof?" Let's write it down for emphasis:

In the inductive step, MAKE AN ASSUMPTION and then USE IT.

Look at the previous examples and find the places where the basis case was proved, where the assumption was made, and where the assumption was used. Do the same thing as you read through the remaining examples.

4.4.3 A Variety of Examples

Now let's do some examples that do not involve numbers. Thus we'll be using well-founded induction (4.29). We should note that some people refer to well-founded induction as "structural induction" because well-founded sets can contain structures other than numbers, such as lists, strings, binary trees, and Cartesian products of sets. Whatever it's called, let's see how to use it.

example 4.42 Correctness of MakeSet

The following function is supposed to take any list K as input and return the list obtained by removing all repeated occurrences of elements from K:

$$\begin{aligned} \text{makeSet}(\langle\,\rangle) &= \langle\,\rangle\,, \\ \text{makeSet}(a :: L) &= \text{if isMember}(a, L) \text{ then makeSet}(L) \\ &\qquad \text{else } a :: \text{makeSet}(L)\,. \end{aligned}$$

We'll assume that isMember correctly checks whether an element is a member of a list. Let $P(K)$ be the statement "makeSet(K) is a list obtained from K by removing its repeated elements." We'll prove that $P(K)$ is true for any list K.

Proof: We'll define a well-founded ordering on lists by letting $K \prec M$ mean length(K) < length(M). So the basis element is $\langle\ \rangle$. The definition of makeSet tells us that makeSet($\langle\ \rangle$) = $\langle\ \rangle$. Thus $P(\langle\ \rangle)$ is true. Next, we'll let K be an arbitrary nonempty list and assume that $P(L)$ is true for all lists $L \prec K$. In other words, we're assuming that makeSet(L) has no repeated elements for all lists $L \prec K$. We need to show that $P(K)$ is true. In other words, we need to show that makeSet(K) has no repeated elements. Since K is nonempty, we can write $K = a :: L$. There are two cases to consider. If isMember(a, L) is true, then the definition of makeSet gives

$$\text{makeSet}(K) = \text{makeSet}(a :: L) = \text{makeSet}(L).$$

Since $L \prec K$, it follows that $P(L)$ is true. Therefore $P(K)$ is true. If isMember(a, L) is false, then the definition of makeSet gives

$$\text{makeSet}(K) = \text{makeSet}(a :: L) = a :: \text{makeSet}(L).$$

Since $L \prec K$, it follows that $P(L)$ is true. Since isMember(a, L) is false, it follows that the list a :: makeSet(L) has no repeated elements. Thus $P(K)$ is true. Therefore, (4.29) implies that $P(K)$ is true for all lists K. QED.

end example

example **4.43 Using a Lexicographic Ordering**

We'll prove that the following function computes the number $|x - y|$ for any natural numbers x and y:

$$f(x, y) = \text{if } x = 0 \text{ then } y \text{ else if } y = 0 \text{ then } x \text{ else } f(x - 1, y - 1).$$

In other words, we'll prove that $f(x, y) = |x - y|$ for all (x, y) in $\mathbb{N} \times \mathbb{N}$.

Proof: We'll use the well-founded set $\mathbb{N} \times \mathbb{N}$ with the lexicographic ordering. For the basis case, we'll check the formula for the least element (0, 0) to get $f(0, 0) = 0 = |0 - 0|$. For the induction case, let $(x, y) \in \mathbb{N} \times \mathbb{N}$ and assume that $f(u, v) = |u - v|$ for all $(u, v) \prec (x, y)$. We must show $f(x, y) = |x - y|$. The case where $x = 0$ is taken care of by observing that $f(0, y) = y = |0 - y|$. Similarly, if $y = 0$, then $f(x, 0) = x = |x - 0|$. The only case remaining is $x \neq 0$ and $y \neq 0$. In this case the definition of f gives $f(x, y) = f(x - 1, y - 1)$. The lexicographic ordering gives $(x - 1, y - 1) \prec (x, y)$. So it follows by induction

that $f(x-1, y-1) = |(x-1)-(y-1)|$. Putting the two equations together we obtain the following result.

$$\begin{aligned} f(x,y) &= f(x-1,y-1) && \text{(definition of } f\text{)} \\ &= |(x-1)-(y-1)| && \text{(induction assumption)} \\ &= |x-y|. \end{aligned}$$

The result now follows from (4.29). QED.

end example

Inductive Proof with One of Several Variables

Sometimes the claims that we wish to prove involve two or more variables, but we only need one of the variables in the proof. For example, suppose we need to show that $P(x, y)$ is true for all $(x, y) \in A \times B$ where the set A is inductively defined. We can perform the following steps, where y denotes an arbitrary element in B:

1. Show that $P(m, y)$ is true for minimal elements $m \in A$.
2. Assume that $P(a, y)$ is true for all predecessors a of x. Show that $P(x, y)$ is true.

The form of the statement $P(x, y)$ often gives us a clue as to whether we can use induction with a single variable. Here are some examples.

example **4.44 Induction with a Single Variable**

Suppose we want to prove that the following function computes the number y^{x+1} for any natural numbers x and y:

$$f(x, y) = \text{if } x = 0 \text{ then } y \text{ else } f(x-1, y) \cdot y.$$

In other words, we want to prove that $f(x, y) = y^{x+1}$ for all (x, y) in $\mathbb{N} \times \mathbb{N}$. We'll use induction with the variable x because it's changing in the definition of f.

Proof: For the basis case the definition of f gives $f(0, y) = y = y^{0+1}$. So the basis case is proved. For the induction case, assume that $x > 0$ and $f(n, y) = y^{n+1}$ for $n < x$. We must show that $f(x, y) = y^{x+1}$. The definition of f and the induction assumption give us the following equations:

$$\begin{aligned} f(x,y) &= f(x-1,y) \cdot y && \text{(definition of } f\text{)} \\ &= y^{x-1+1} \cdot y && \text{(induction assumption)} \\ &= y^{x+1}. \end{aligned}$$

The result now follows from (4.29). QED.

end example

example

4.45 Inserting an Element in a Binary Search Tree

Let's prove that the following insert function does its job. Given a number x and a binary search tree T, the function returns a binary search tree obtained by inserting x in T.

$$\begin{aligned}\text{insert}\,(x,T) = {} & \text{if } T = \langle\,\rangle \text{ then tree}\,(\langle\,\rangle, x, \langle\,\rangle)\\ & \text{else if } x < \text{root}\,(T) \text{ then}\\ & \quad \text{tree}\,(\text{insert}\,(x, \text{left}\,(T)), \text{root}\,(T), \text{right}\,(T))\\ & \text{else}\\ & \quad \text{tree}\,(\text{left}\,(T), \text{root}\,(T), \text{insert}\,(x, \text{right}\,(T)))\,.\end{aligned}$$

The claim that we wish to prove is,

insert(x, T) is a binary search tree for all binary search trees T.

Proof: We'll use induction on the binary tree variable T. Our ordering of binary search trees will be based on the number of nodes in a tree. The empty binary tree $\langle\,\rangle$ is a binary search tree with no nodes. In this case, we have insert(x, $\langle\,\rangle$) = tree($\langle\,\rangle$, x, $\langle\,\rangle$), which is a single node binary tree and thus also a binary search tree. So the basis case is true. Now let T be a nonempty binary search tree of the form $T = \text{tree}(L, y, R)$ and, since L and R each have fewer nodes than T, we can assume that insert(x, L) and insert(x, R) are binary search trees. With these assumptions we must show that insert(x, T) is a binary search tree. There are two cases to consider, depending on whether $x < y$. First, suppose $x < y$. Then we have

$$\text{insert}(x,\ T) = \text{tree}(\text{insert}(x,\ L),\ y,\ R).$$

By the induction assumption it follows that insert(x, L) is a binary search tree. Thus insert(x, T) is a binary search tree. We obtain a similar result if $x \geq y$. It follows from (4.29) that insert(x, T) is a binary search tree for all binary search trees T. QED.

end example

On Using "Well-Founded"

We often see induction proofs that don't mention the word "well-founded." For example, we might see a statement such as: "We will use induction on the depth of the trees." In such a case the induction assumption might be stated something like "Assume that $P(T)$ is true for all trees T with depth less than n." Then a proof is given that uses the assumption to prove that $P(T)$ is true for an arbitrary tree of depth n. Even though the term "well-founded" may not be mentioned in a proof, there is always a well-founded ordering lurking underneath the surface.

Before we leave the subject of inductive proof, let's discuss how we can use inductive proof to help us tell whether inductive definitions of sets are correct.

Proofs about Inductively Defined Sets

Recall that a set S is inductively defined by a basis case, an inductive case, and a closure case (which we never state explicitly). The closure case says that S is the smallest set satisfying the basis and inductive cases. The closure case can also be stated in practical terms as follows.

Closure Property of Inductive Definitions (4.31)

If S is an inductively defined set and T is a set that also satisfies the basis and inductive cases for the definition of S, and if $T \subseteq S$, then it must be the case that $T = S$.

We can use this closure property to see whether an inductive definition correctly characterizes a given set. For example, suppose we have an inductive definition for a set named S, we have some other description of a set named T, and we wish to prove that T and S are the same set. Then we must prove three things:

1. Prove that T satisfies the basis case of the inductive definition.
2. Prove that T satisfies the inductive case of the inductive definition.
3. Prove that $T \subseteq S$. This can often be accomplished with an induction proof.

example 4.46 Describing an Inductive Set

Suppose we have the following inductive definition for a set S:

Basis: $1 \in S$.

Induction: If $x \in S$, then $x + 2 \in S$.

This gives us a pretty good description of S. For example, suppose someone tells us that $S = \{2k + 1 \mid k \in \mathbb{N}\}$. It seems reasonable. Can we prove it? Let's give it a try. To clarify the situation, we'll let $T = \{2k + 1 \mid k \in \mathbb{N}\}$ and prove that $T = S$. We'll be done if we can show that T satisfies the basis and induction cases for S and that $T \subseteq S$. Then the closure property of inductive definitions will tell us that $T = S$.

Proof: Observe that $1 = 2 \cdot 0 + 1 \in T$ and if $x \in T$, then $x = 2k + 1$ and it follows that $x + 2 = 2(k + 1) + 1 \in T$. So T satisfies the inductive definition. Next we need to show that $T \subseteq S$. In other words, show that $2k + 1 \in S$ for all $k \in \mathbb{N}$. This calls for an induction proof. If $k = 0$, we have $2 \cdot 0 + 1 = 1 \in S$. Now assume that $2k + 1 \in S$ and show that $2(k + 1) + 1 \in S$. Since $2k + 1 \in S$, the inductive definition tells us that $(2k + 1) + 2 \in S$

so that we have $2(k + 1) + 1 = (2k + 1) + 2 \in S$. Therefore, $2k + 1 \in S$ for all $k \in \mathbb{N}$, which proves that $T \subseteq S$. So we've proven the three things that allow us to conclude—by the closure property of inductive definitions—that $T = S$. QED.

end example

example **4.47 A Correct Grammar**

Suppose we're asked to find a grammar for the language $\{ab^n \mid n \in \mathbb{N}\}$, and we write the following grammar G.

$$S \to a \mid Sb.$$

This grammar seems to do the job. But how do we know for sure? One way is to use (3.15) to create an inductive definition for $L(G)$, the language of G. Then we can try to prove that $L(G) = \{ab^n \mid n \in \mathbb{N}\}$. Using (3.15) we see that the basis case is $a \in L(G)$ because of the derivation $S \Rightarrow a$.

For the induction case, if $x \in L(G)$ with derivation $S \Rightarrow^+ x$, then we can add one step to the derivation by using the recursive production $S \to Sb$ to obtain the derivation $S \Rightarrow Sb \Rightarrow^+ xb$. So we obtain the following inductive definition for $L(G)$.

Basis: $a \in L(G)$.

Induction: If $x \in L(G)$, then $xb \in L(G)$.

Now we'll prove that $\{ab^n \mid n \in \mathbb{N}\} = L(G)$. For ease of notation we'll let $M = \{ab^n \mid n \in \mathbb{N}\}$. So we must prove that $M = L(G)$. By (4.31) we must show that M satisfies the basis and induction cases and that $M \subseteq L(G)$. Then by the closure property of inductive definitions we will infer that $M = L(G)$.

Proof: Since $a = ab^0 \in M$, it follows that the basis case of the inductive definition holds. For the induction case, let $x \in M$. Then $x = ab^n$ for some number $n \in \mathbb{N}$. Thus $xb = ab^{n+1} \in M$. Therefore, M satisfies the inductive definition. Now we'll show that $M \subseteq L(G)$ with an induction proof. For $n = 0$, we have $ab^0 = a \in L(G)$. Now assume that $ab^n \in L(G)$. Then the definition of $L(G)$ tells us that $ab^n b \in L(G)$. But $ab^{n+1} = ab^n b$. So $ab^{n+1} \in L(G)$. Therefore, $M \subseteq L(G)$. Now the closure property of inductive definitions gives us our conclusion $M = L(G)$. QED.

end example

Exercises

Numbers

1. Find the sum of the arithmetic progression 12, 26, 40, 54, 68, ..., 278.

2. Use induction to prove each of the following equations.

   a. $1 + 3 + 5 + \cdots + (2n - 1) = n^2$.

   b. $5 + 9 + 11 + \cdots + (2n + 3) = n^2 + 4n$.

   c. $3 + 7 + 11 + \cdots + (4n - 1) = n(2n + 1)$.

   d. $2 + 6 + 10 + \cdots + (4n - 2) = 2n^2$.

   e. $1 + 5 + 9 + \cdots + (4n + 1) = (n + 1)(2n + 1)$.

   f. $2 + 8 + 24 + \cdots + n2^n = (n - 1)2^{n+1} + 2$.

   g. $1^2 + 2^2 + \cdots + n^2 = \dfrac{n\,(n+1)\,(2n+1)}{6}$.

   h. $2 + 6 + 12 + \cdots + n(n + 1) = \dfrac{n\,(n+1)\,(n+2)}{3}$.

   i. $2 + 6 + 12 + \cdots + (n^2 - n) = \dfrac{n\,(n^2-1)}{3}$.

   j. $(1 + 2 + \cdots + n)^2 = 1^3 + 2^3 + \cdots + n^3$.

3. The *Fibonacci numbers* are defined by $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. Use induction to prove each of the following statements.

   a. $F_0 + F_1 + \cdots + F_n = F_{n+2} - 1$.

   b. $F_{n-1}F_{n+1} - F_n^2 = (-1)^n$.

   c. $F_{m+n} = F_{m-1}F_n + F_mF_{n+1}$. *Hint:* Use the lexicographic ordering of $\mathbb{N} \times \mathbb{N}$.

   d. If $m|n$ then $F_m \,|F_n$. *Hint:* Use the fact that $n = mk$ for some k and show the result by induction on k with the help of Part (c).

4. The *Lucas numbers* are defined by $L_0 = 2$, $L_1 = 1$, and $L_n = L_{n-1} + L_{n-2}$ for $n \geq 2$. The sequence begins as 2, 1, 3, 4, 7, 11, 18, ... These numbers are named after the mathematician Édouard Lucas (1842–1891). Use induction to prove each of the following statements.

   a. $L_0 + L_1 + \cdots + L_n = L_{n+2} - 1$.

   b. $L_n = F_{n-1} + F_{n+1}$ for $n \geq 1$, where F_n is the nth Fibonacci number.

5. Let $\text{sum}(n) = 1 + 2 + \cdots + n$ for all natural numbers n. Give an induction proof to show that the following equation is true for all natural numbers m and n: $\text{sum}(m + n) = \text{sum}(m) + \text{sum}(n) + mn$.

6. We know that $1 + 2 = 3$, $4 + 5 + 6 = 7 + 8$, and $9 + 10 + 11 + 12 = 13 + 14 + 15$. Show that we can continue these equations forever. *Hint:* The left side of each equation starts with a number of the form n^2. Formulate a general summation for each side, and then prove that the two sums are equal.

Structures

7. Let $R = \{(x, x + 1) \mid x \in \mathbb{N}\}$ and let L be the "less than" relation on $\mathbb{N}$. Prove that $t(R) = L$.

8. Use induction to prove that a finite set with n elements has 2^n subsets.

9. Use induction to prove that the function f computes the length of a list:

$$f(L) = \text{if } L = \langle\,\rangle \text{ then } 0 \text{ else } 1 + f(\text{tail}(L)).$$

10. Use induction to prove that each function performs its stated task.

a. The function g computes the number of nodes in a binary tree:

$$\begin{aligned} g(T) = \text{if } T = \langle\,\rangle \text{ then } & 0 \\ \text{else } & 1 + g(\text{left}(T)) + g(\text{right}(T)). \end{aligned}$$

b. The function h computes the number of leaves in a binary tree:

$$\begin{aligned} h(T) = \text{if } T = \langle\,\rangle \text{ then } & 0 \\ \text{else if } T = \text{tree}(\langle\,\rangle, x, \langle\,\rangle) \text{ then } & 1 \\ \text{else } & h(\text{left}(T)) + h(\text{right}(T)). \end{aligned}$$

11. Suppose we have the following two procedures to write out the elements of a list. One claims to write the elements in the order listed, and one writes out the elements in reverse order. Prove that each is correct.

a. forward(L): if $L \neq \langle\,\rangle$ then {print(head(L)); forward(tail(L))}.

b. back(L): if $L \neq \langle\,\rangle$ then {back(tail(L)); print(head(L))}.

12. The following function "sort" takes a list of numbers and returns a sorted version of the list (from lowest to highest), where "insert" places an element correctly into a sorted list:

$$\begin{aligned} &\text{sort}(\langle\,\rangle) = \langle\,\rangle, \\ &\text{sort}(x :: L) = \text{insert}(x, \text{sort}(L)). \end{aligned}$$

a. Assume that the function insert is correct. That is, if S is sorted, then insert(x, S) is also sorted. Prove that sort is correct.

b. Prove that the following definition for insert is correct. That is, prove that insert(x, S) is sorted for all sorted lists S.

$$\begin{aligned} \text{insert}(x, S) = \text{if } S = \langle\,\rangle \text{ then } & \langle x \rangle \\ \text{else if } x \leq \text{head}(S) \text{ then } & x :: S \\ \text{else } & \text{head}(S) :: \text{insert}(x, \text{tail}(S)). \end{aligned}$$

13. Show that the following function g correctly computes the greatest common divisor for each pair of positive integers x and y: *Hint:* (2.2a and 2.2b) might be useful.

$$\begin{aligned} g(x,y) = \text{if } x = y \text{ then } & x \\ \text{else if } x > y \text{ then } & g(x-y,y) \\ \text{else } & g(x,y-x). \end{aligned}$$

14. The following program is supposed to input a list of numbers L and output a binary search tree containing the numbers in L:

$$\begin{aligned} f(L) = \text{if } L = \langle\,\rangle \text{ then } & \langle\,\rangle \\ \text{else insert}(\text{head}(L), & f(\text{tail}(L))). \end{aligned}$$

Assume that insert(x, T) correctly returns the binary search tree obtained by inserting the number x in the binary search tree T. Prove the following claim: $f(M)$ is a binary search tree for all lists M.

15. The following program is supposed to return the list obtained by removing the first occurrence of x from the list L.

$$\begin{aligned} \text{delete}(x,L) = \text{if } L = \langle\,\rangle \text{ then } & \langle\,\rangle \\ \text{else if } x = \text{head}(L) \text{ then } & \text{tail}(L) \\ \text{else } & \text{head}(L) :: \text{delete}(x, \text{tail}(L)). \end{aligned}$$

Prove that delete performs as expected.

16. The following function claims to remove all occurrences of an element from a list:

$$\begin{aligned} \text{removeAll}(a,L) = \text{if } L = \langle\,\rangle \text{ then } & L \\ \text{else if } a = \text{head}(L) \text{ then } & \text{removeAll}(a, \text{tail}(L)) \\ \text{else } & \text{head}(L) :: \text{removeAll}(a, \text{tail}(L)). \end{aligned}$$

Prove that removeAll satisfies the claim.

17. Let r stand for the removeAll function from Exercise 16. Prove the following property of r for all elements a, b, and all lists L:

$$r(a,\, r(b,\, L)) = r(b,\, r(a,\, L)).$$

18. The following program computes a well-known function called *Ackermann's function. Note:* If you try out this function, don't let x and y get too large.

$$\begin{aligned} f(x,y) = \text{if } x = 0 \text{ then } & y+1 \\ \text{else if } y = 0 \text{ then } & f(x-1,1) \\ \text{else } & f(x-1, f(x,y-1)). \end{aligned}$$

Prove that f is defined for all pairs (x, y) in $\mathbb{N} \times \mathbb{N}$. *Hint:* Use the lexicographic ordering on $\mathbb{N} \times \mathbb{N}$. This gives the single basis element $(0, 0)$. For the induction assumption, assume that $f(x', y')$ is defined for all (x', y') such that $(x', y') \prec (x, y)$. Then show that $f(x, y)$ is defined.

19. Let the function "isMember" be defined as follows for any list L:

$$\begin{aligned}\text{isMember}\,(a, L) = \text{if } L = \langle\,\rangle \text{ then False}\\ \text{else if } a = \text{head}\,(L) \text{ then True}\\ \text{else isMember}\,(a, \text{tail}\,(L))\,.\end{aligned}$$

a. Prove that isMember is correct. That is, show that isMember(a, L) is true if and only if a occurs as an element of L.

b. Prove that the following equation is true for all lists L when $a \neq b$, where removeAll is the function from Exercise 16:

$$\text{isMember}(a, \text{removeAll}(b, L)) = \text{isMember}(a, L).$$

20. Use induction to prove that the following concatenation function is associative.

$$\begin{aligned}\text{cat}\,(x, y) = \text{if } x = \langle\,\rangle \text{ then } y\\ \text{else head}\,(x) :: \text{cat}\,(\text{tail}\,(x)\,, y)\,.\end{aligned}$$

In other words, show that cat(x, cat(y, z)) = cat(cat(x, y), z) for all lists x, y, and z.

21. Two students came up with the following two solutions to a problem. Both students used the removeAll function from Exercise 16, which we abbreviate to r.

$$\begin{aligned}\text{Student A:}\quad f(L) = \text{if } L = \langle\,\rangle \text{ then } \langle\,\rangle\\ \text{else head}(L) :: r(\text{head}(L), f(\text{tail}(L))).\\ \text{Student B:}\quad g(L) = \text{if } L = \langle\,\rangle \text{ then } \langle\,\rangle\\ \text{else head}(L) :: g(r(\text{head}(L), \text{tail}(L))).\end{aligned}$$

a. Prove that $r(a, g(L)) = g(r(a, L))$ for all elements a and all lists L. *Hint:* Exercise 17 might be useful in the proof.

b. Prove that $f(L) = g(L)$ for all lists L. *Hint:* Part (a) could be helpful.

c. Can you find an appropriate name for f and g? Can you prove that the name you choose is correct?

Challenges

22. Prove that Condition 1 of (4.28) is a consequence of Condition 2 of (4.28).

23. Let G be the grammar $S \to a \mid abS$, and let $M = \{(ab)^n a \mid n \in \mathbb{N}\}$. Use (3.15) to construct an inductive definition for $L(G)$. Then use (4.31) to prove that $M = L(G)$.

24. A useful technique for recursively defined functions involves keeping—or accumulating—the results of function calls in *accumulating parameters*: The values in the accumulating parameters can then be used to compute subsequent values of the function that are then used to replace the old values in the accumulating parameters. We call the function by giving initial values to the accumulating parameters. Often these initial values are basis values for an inductively defined set of elements.

For example, suppose we define the function f as follows:

$$f(n, u, v) = \text{if } n = 0 \text{ then } u \text{ else } f(n - 1, v, u + v).$$

The second and third arguments to f are accumulating parameters because they always hold two possible values of the function. Prove each of the following statements.

a. $f(n, 0, 1) = F_n$, the nth Fibonacci number.

b. $f(n, 2, 1) = L_n$, the nth Lucas number.

Hint: For Part (a), show that $f(n, 0, 1) = f(k, F_{n-k}, F_{n-k+1})$ for $0 \leq k \leq n$. A similar hint applies to Part (b).

25. A *derangement* of a string is an arrangement of the letters of the string such that no letter remains in the same position. In terms of bijections, a derangement of a set S is a bijection f on S such that $f(x) \neq x$ for all x in S. The number of derangements of an n-element set can be given by the following recursively defined function:

$$\begin{aligned} d(1) &= 0, \\ d(2) &= 1, \\ d(n) &= (n-1)\,(d(n-1) + d(n-2)) \quad (n \geq 3). \end{aligned}$$

Give an inductive proof that $d(n) = nd(n-1) + (-1)^n$ for $n \geq 2$.

4.5 Chapter Summary

Binary relations are common denominators for describing the ideas of equivalence, order, and inductive proof. The basic properties that a binary relation may or may not possess are reflexive, symmetric, transitive, irreflexive, and antisymmetric. Binary relations can be constructed from other binary relations by composition and closure, and by the usual set operations. Transitive closure plays an important part in algorithms for solving path problems—Warshall's algorithm, Floyd's algorithm, and the modification of Floyd's algorithm to find shortest paths.

Equivalence relations are characterized by being reflexive, symmetric, and transitive. These relations generalize the idea of basic equality by partitioning a set into classes of equivalent elements. Any set has a hierarchy of partitions ranging from fine to coarse. Equivalence relations can be generated from other relations by taking the transitive symmetric reflexive closure. They can also be generated from functions by the kernel relation. The equivalence problem can be solved by a novel tree structure. Kruskal's algorithm uses an equivalence relation to find a minimal spanning tree for a weighted undirected graph.

Order relations are characterized by being transitive and antisymmetric. Sets with these properties are called posets—for partially ordered sets—because it may be the case that not all pairs of elements are related. The ideas of successor and predecessor apply to posets. Posets can also be topologically sorted. A well-founded poset is characterized by the condition that no descending chain of elements can go on forever. This is equivalent to the condition that any nonempty subset has a minimal element. Well-founded sets can be constructed by mapping objects into a known well-founded set such as the natural numbers. Inductively defined sets are well-founded.

Inductive proof is a powerful technique that can be used to prove infinitely many statements. The most basic inductive proof technique is the principle of mathematical induction. Another useful inductive proof technique is well-founded induction. The important thing to remember about applying inductive proof techniques is to *make an assumption* and then *use the assumption* that you made. Inductive proof techniques can be used to prove properties of recursively defined functions and inductively defined sets.