Chapter 2

Sets, Functions, and Relations

**We deal with the complexity of the world by putting things into categories.** There are not just hordes of individual creatures. There are dogs, cats, elephants, and mice. There are mammals, insects, and fish. Animals, vegetables, and minerals. Solids, liquids, and gases. Things that are red. Big cities. Pleasant memories. . . . Categories build on categories. They are the subject and the substance of thought.

In mathematics, which operates in its own abstract and rigorous world, categories are modeled by sets. A set is just a collection of elements. Along with logic, sets form the “foundation” of mathematics, just as categories are part of the foundation of day-to-day thought. In this chapter, we study sets and relationships among sets.

### 2.1 Basic Concepts

A **set** is a collection of **elements**. A set is defined entirely by the elements that it contains. An element can be anything, including another set. You will notice that this is not a precise mathematical definition. Instead, it is an intuitive description of what the word “set” is supposed to mean: Any time you have a bunch of entities and you consider them as a unit, you have a set. Mathematically, sets are really defined by the operations that can be performed on them. These operations model things that can be done with collections of objects in the real world. These operations are the subject of the branch of mathematics known as **set theory**.

The most basic operation in set theory is forming a set from a given list
of specific entities. The set that is formed in this way is denoted by enclosing the list of entities between a left brace, “{”, and a right brace, “}”. The entities in the list are separated by commas. For example, the set denoted by

\[
\{17, \pi, \text{New York City}, \text{Barack Obama}, \text{Big Ben}\}
\]

is the set that contains the entities 17, \(\pi\), New York City, Barack Obama, and Big Ben. These entities are the elements of the set. Since we assume that a set is completely defined by the elements that it contains, the set is well-defined. Of course, we still haven’t said what it means to be an “entity.” Something as definite as “New York City” should qualify, except that it doesn’t seem like New York City really belongs in the world of Mathematics. The problem is that mathematics is supposed to be its own self-contained world, but it is supposed to model the real world. When we use mathematics to model the real world, we admit entities such as New York City and even Big Ben. But when we are doing mathematics per se, we’ll generally stick to obviously mathematical entities such as the integer 17 or the real number \(\pi\). We will also use letters such as \(a\) and \(b\) to refer to entities. For example, when I say something like “Let \(A\) be the set \(\{a, b, c\}\),” I mean \(a\), \(b\), and \(c\) to be particular, but unspecified, entities.

It’s important to understand that a set is defined by the elements that it contains, and not by the order in which those elements might be listed. For example, the notations \(\{a, b, c, d\}\) and \(\{b, c, a, d\}\) define the same set. Furthermore, a set can only contain one copy of a given element, even if the notation that specifies the set lists the element twice. This means that \(\{a, b, a, b, c, a\}\) and \(\{a, b, c\}\) specify exactly the same set. Note in particular that it’s incorrect to say that the set \(\{a, b, a, b, c, a\}\) contains seven elements, since some of the elements in the list are identical. The notation \(\{a, b, c\}\) can lead to some confusion, since it might not be clear whether the letters \(a\), \(b\), and \(c\) are assumed to refer to three different entities. A mathematician would generally not make this assumption without stating it explicitly, so that the set denoted by \(\{a, b, c\}\) could actually contain either one, two, or three elements. When it is important that different letters refer to different entities, I will say so explicitly, as in “Consider the set \(\{a, b, c\}\), where \(a\), \(b\), and \(c\) are distinct.”

The symbol \(\in\) is used to express the relation “is an element of.” That is, if \(a\) is an entity and \(A\) is a set, then \(a \in A\) is a statement that is true if and only if \(a\) is one of the elements of \(A\). In that case, we also say that \(a\) is a member of the set \(A\). The assertion that \(a\) is not an element of \(A\) is expressed by the notation \(a \notin A\). Note that both \(a \in A\) and \(a \notin A\) are statements in the sense of propositional logic. That is, they are assertions which can be either true or false. The statement \(a \notin A\) is equivalent to
2.1. BASIC CONCEPTS

It is possible for a set to be empty, that is, to contain no elements whatsoever. Since a set is completely determined by the elements that it contains, there is only one set that contains no elements. This set is called the empty set, and it is denoted by the symbol $\emptyset$. Note that for any element $a$, the statement $a \in \emptyset$ is false. The empty set, $\emptyset$, can also be denoted by an empty pair of braces, \{\}. If $A$ and $B$ are sets, then, by definition, $A$ is equal to $B$ if and only if they contain exactly the same elements. In this case, we write $A = B$. Using the notation of predicate logic, we can say that $A = B$ if and only if $\forall x (x \in A \leftrightarrow x \in B)$. Suppose now that $A$ and $B$ are sets such that every element of $A$ is an element of $B$. In that case, we say that $A$ is a subset of $B$, i.e. $A$ is a subset of $B$ if and only if $\forall x (x \in A \rightarrow x \in B)$. The fact that $A$ is a subset of $B$ is denoted by $A \subseteq B$. Note that $\emptyset$ is a subset of every set $B$: $x \in \emptyset$ is false for any $x$, and so given any $B$, $(x \in \emptyset \rightarrow x \in B)$ is true for all $x$. If $A = B$, then it is automatically true that $A \subseteq B$ and that $B \subseteq A$. The converse is also true: If $A \subseteq B$ and $B \subseteq A$, then $A = B$. This follows from the fact that for any $x$, the statement $(x \in A \rightarrow x \in B)$ is logically equivalent to the statement $(x \in A \rightarrow x \in B) \land (x \in B \rightarrow x \in A)$. This fact is important enough to state as a theorem.

**Theorem 2.1.** Let $A$ and $B$ be sets. Then $A = B$ if and only if both $A \subseteq B$ and $B \subseteq A$.

This theorem expresses the following advice: If you want to check that two sets, $A$ and $B$, are equal, you can do so in two steps. First check that every element of $A$ is also an element of $B$, and then check that every element of $B$ is also an element of $A$.

If $A \subseteq B$ but $A \neq B$, we say that $A$ is a proper subset of $B$. We use the notation $A \subset B$ to mean that $A$ is a proper subset of $B$. That is, $A \subset B$ if and only if $A \subseteq B \land A \neq B$. We will sometimes use $A \supseteq B$ as an equivalent notation for $B \subseteq A$, and $A \supset B$ as an equivalent for $B \subset A$.

A set can contain an infinite number of elements. In such a case, it is not possible to list all the elements in the set. Sometimes the ellipsis “...” is used to indicate a list that continues on infinitely. For example, $\mathbb{N}$, the set of natural numbers, can be specified as

$$\mathbb{N} = \{0, 1, 2, 3, \ldots\}$$

However, this is an informal notation, which is not really well-defined, and it should only be used in cases where it is clear what it means. It’s not
very useful to say that “the set of prime numbers is \(\{2, 3, 5, 7, 11, 13, \ldots\}\),” and it is completely meaningless to talk about “the set \(\{17, 42, 105, \ldots\}\).” Clearly, we need another way to specify sets besides listing their elements. The need is fulfilled by predicates.

If \(P(x)\) is a predicate, then we can form the set that contains all entities \(a\) such that \(a\) is in the domain of discourse for \(P\) and \(P(a)\) is true. The notation \(\{x \mid P(x)\}\) is used to denote this set. The name of the variable, \(x\), is arbitrary, so the same set could equally well be denoted as \(\{z \mid P(z)\}\) or \(\{r \mid P(r)\}\). The notation \(\{x \mid P(x)\}\) can be read “the set of \(x\) such that \(P(x)\).” For example, if \(E(x)\) is the predicate “\(x\) is an even number,” and if the domain of discourse for \(E\) is the set \(\mathbb{N}\) of natural numbers, then the notation \(\{x \mid E(x)\}\) specifies the set of even natural numbers. That is,

\[
\{x \mid E(x)\} = \{0, 2, 4, 6, 8, \ldots\}
\]

It turns out, for deep and surprising reasons that we will discuss later in this section, that we have to be a little careful about what counts as a predicate. In order for the notation \(\{x \mid P(x)\}\) to be valid, we have to assume that the domain of discourse of \(P\) is in fact a set. (You might wonder how it could be anything else. That’s the surprise!) Often, it is useful to specify the domain of discourse explicitly in the notation that defines a set. In the above example, to make it clear that \(x\) must be a natural number, we could write the set as \(\{x \in \mathbb{N} \mid E(x)\}\). This notation can be read as “the set of all \(x\) in \(\mathbb{N}\) such that \(E(x)\).” More generally, if \(X\) is a set and \(P\) is a predicate whose domain of discourse includes all the elements of \(X\), then the notation

\[
\{x \in X \mid P(x)\}
\]

is the set that consists of all entities \(a\) that are members of the set \(X\) and for which \(P(a)\) is true. In this notation, we don’t have to assume that the domain of discourse for \(P\) is a set, since we are effectively limiting the domain of discourse to the set \(X\). The set denoted by \(\{x \in X \mid P(x)\}\) could also be written as \(\{x \mid x \in X \land P(x)\}\).

We can use this notation to define the set of prime numbers in a rigorous way. A prime number is a natural number \(n\) which is greater than 1 and which satisfies the property that for any factorization \(n = xy\), where \(x\) and \(y\) are natural numbers, either \(x\) or \(y\) must be \(n\). We can express this definition as a predicate and define the set of prime numbers as

\[
\{n \in \mathbb{N} \mid (n > 1) \land \forall x \forall y ((x \in \mathbb{N} \land y \in \mathbb{N} \land n = xy) \rightarrow (x = n \lor y = n))\}.
\]

Admittedly, this definition is hard to take in in one gulp. But this example shows that it is possible to define complex sets using predicates.
2.1. BASIC CONCEPTS

Now that we have a way to express a wide variety of sets, we turn to operations that can be performed on sets. The most basic operations on sets are union and intersection. If $A$ and $B$ are sets, then we define the union of $A$ and $B$ to be the set that contains all the elements of $A$ together with all the elements of $B$. The union of $A$ and $B$ is denoted by $A \cup B$. The union can be defined formally as

$$A \cup B = \{ x \mid x \in A \lor x \in B \}.$$ 

The intersection of $A$ and $B$ is defined to be the set that contains every entity that is both a member of $A$ and a member of $B$. The intersection of $A$ and $B$ is denoted by $A \cap B$. Formally,

$$A \cap B = \{ x \mid x \in A \land x \in B \}.$$ 

An entity gets into $A \cup B$ if it is in either $A$ or $B$. It gets into $A \cap B$ if it is in both $A$ and $B$. Note that the symbol for the logical “or” operator, $\lor$, is similar to the symbol for the union operator, $\cup$, while the logical “and” operator, $\land$, is similar to the intersection operator, $\cap$.

The set difference of two sets, $A$ and $B$, is defined to be the set of all entities that are members of $A$ but are not members of $B$. The set difference of $A$ and $B$ is denoted $A \setminus B$. The idea is that $A \setminus B$ is formed by starting with $A$ and then removing any element that is also found in $B$. Formally,

$$A \setminus B = \{ x \mid x \in A \land x \not\in B \}.$$ 

Union and intersection are clearly commutative operations. That is, $A \cup B = B \cup A$ and $A \cap B = B \cap A$ for any sets $A$ and $B$. However, set difference is not commutative. In general, $A \setminus B \not= B \setminus A$.

Suppose that $A = \{a, b, c\}$, that $B = \{b, d\}$, and that $C = \{d, e, f\}$. Then we can apply the definitions of union, intersection, and set difference to compute, for example, that:

$$A \cup B = \{a, b, c, d\} \quad A \cap B = \{b\} \quad A \setminus B = \{a, c\}$$

$$A \cup C = \{a, b, c, d, e, f\} \quad A \cap C = \emptyset \quad A \setminus C = \{a, b, c\}$$

In this example, the sets $A$ and $C$ have no elements in common, so that $A \cap C = \emptyset$. There is a term for this: Two sets are said to be disjoint if they have no elements in common. That is, for any sets $A$ and $B$, $A$ and $B$ are said to be disjoint if and only if $A \cap B = \emptyset$.

Of course, the set operations can also be applied to sets that are defined by predicates. For example, let $L(x)$ be the predicate “$x$ is lucky,” and let $W(x)$ be the predicate “$x$ is wise,” where the domain of discourse for each
CHAPTER 2. SETS, FUNCTIONS, AND RELATIONS

Notation | Definition
---|---
$a \in A$ | $a$ is a member (or element) of $A$
$a \notin A$ | $\neg(a \in A)$, $a$ is not a member of $A$
$\emptyset$ | the empty set, which contains no elements
$A \subseteq B$ | $A$ is a subset of $B$, $\forall x(x \in A \rightarrow x \in B)$
$A \subset B$ | $A$ is a proper subset of $B$, $A \subseteq B \land A \neq B$
$A \supseteq B$ | $A$ is a superset of $B$, same as $B \subseteq A$
$A \supset B$ | $A$ is a proper superset of $B$, same as $B \supset A$
$A = B$ | $A$ and $B$ have the same members, $A \subseteq B \land B \subseteq A$
$A \cup B$ | union of $A$ and $B$, $\{x \mid x \in A \lor x \in B\}$
$A \cap B$ | intersection of $A$ and $B$, $\{x \mid x \in A \land x \in B\}$
$A \setminus B$ | set difference of $A$ and $B$, $\{x \mid x \in A \land x \notin B\}$
$\mathcal{P}(A)$ | power set of $A$, $\{X \mid X \subseteq A\}$

Figure 2.1: Some of the notations that are defined in this section. $A$ and $B$ are sets, and $a$ is an entity.

Predicate is the set of people. Let $X = \{x \mid L(x)\}$, and let $Y = \{x \mid W(x)\}$. Then

\[
X \cup Y = \{x \mid L(x) \lor W(x)\} = \{\text{people who are lucky or wise}\}
\]
\[
X \cap Y = \{x \mid L(x) \land W(x)\} = \{\text{people who are lucky and wise}\}
\]
\[
X \setminus Y = \{x \mid L(x) \land \neg W(x)\} = \{\text{people who are lucky but not wise}\}
\]
\[
Y \setminus X = \{x \mid W(x) \land \neg L(x)\} = \{\text{people who are wise but not lucky}\}
\]

You have to be a little careful with the English word “and.” We might say that the set $X \cup Y$ contains people who are lucky and people who are wise. But what this means is that a person gets into the set $X \cup Y$ either by being lucky or by being wise, so $X \cup Y$ is defined using the logical “or” operator, $\lor$.

Sets can contain other sets as elements. For example, the notation $\{a, \{b\}\}$ defines a set that contains two elements, the entity $a$ and the set $\{b\}$. Since the set $\{b\}$ is a member of the set $\{a, \{b\}\}$, we have that $\{b\} \in \{a, \{b\}\}$. On the other hand, provided that $a \neq b$, the statement $\{b\} \subseteq \{a, \{b\}\}$ is false, since saying $\{b\} \subseteq \{a, \{b\}\}$ is equivalent to saying
that $b \in \{a, \{b\}\}$, and the entity $b$ is not one of the two members of $\{a, \{b\}\}$. For the entity $a$, it is true that $\{a\} \subseteq \{a, \{b\}\}$.

Given a set $A$, we can construct the set that contains all the subsets of $A$. This set is called the **power set** of $A$, and is denoted $\mathcal{P}(A)$. Formally, we define

$$\mathcal{P}(A) = \{X \mid X \subseteq A\}.$$  

For example, if $A = \{a, b\}$, then the subsets of $A$ are the empty set, $\{a\}$, $\{b\}$, and $\{a, b\}$, so the power set of $A$ is set given by

$$\mathcal{P}(A) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}.$$ 

Note that since the empty set is a *subset* of any set, the empty set is an *element* of the power set of any set. That is, for any set $A$, $\emptyset \subseteq A$ and $\emptyset \in \mathcal{P}(A)$. Since the empty set is a subset of itself, and is its only subset, we have that $\mathcal{P}(\emptyset) = \{\emptyset\}$. The set $\{\emptyset\}$ is not empty. It contains one element, namely $\emptyset$.

We remarked earlier in this section that the notation $\{x \mid P(x)\}$ is only valid if the domain of discourse of $P$ is a set. This might seem a rather puzzling thing to say—after all, why and how would the domain of discourse be anything else? The answer is related to Russell’s Paradox, which we mentioned briefly in Chapter 1 and which shows that it is logically impossible for the set of all sets to exist. This impossibility can be demonstrated using a proof by contradiction. In the proof, we use the existence of the set of all sets to define another set which cannot exist because its existence would lead to a logical contradiction.

**Theorem 2.2.** There is no set of all sets.

**Proof.** Suppose that the set of all sets exists. We will show that this assumption leads to a contradiction. Let $V$ be the set of all sets. We can then define the set $R$ to be the set which contains every set that does not contain itself. That is,

$$R = \{X \in V \mid X \not\in X\}.$$  

Now, we must have either $R \in R$ or $R \not\in R$. We will show that either case leads to a contradiction.

Consider the case where $R \in R$. Since $R \in R$, $R$ must satisfy the condition for membership in $R$. A set $X$ is in $R$ iff $X \not\in X$. To say that $R$ satisfies this condition means that $R \not\in R$. That is, from the fact that $R \in R$, we deduce the contradiction that $R \not\in R$. 

CHAPTER 2. SETS, FUNCTIONS, AND RELATIONS

Now consider the remaining case, where \( R \not\in R \). Since \( R \not\in R \), \( R \) does not satisfy the condition for membership in \( R \). Since the condition for membership is that \( R \not\in R \), and this condition is false, the statement \( R \not\in R \) must be false. But this means that the statement \( R \in R \) is true. From the fact that \( R \not\in R \), we deduce the contradiction that \( R \in R \).

Since both possible cases, \( R \in R \) and \( R \not\in R \), lead to contradictions, we see that it is not possible for \( R \) to exist. Since the existence of \( R \) follows from the existence of \( V \), we see that \( V \) also cannot exist.

To avoid Russell’s paradox, we must put limitations on the construction of new sets. We can’t force the set of all sets into existence simply by thinking of it. We can’t form the set \( \{ x \mid P(x) \} \) unless the domain of discourse of \( P \) is a set. Any predicate \( Q \) can be used to form a set \( \{ x \in X \mid Q(x) \} \), but this notation requires a pre-existing set \( X \). Predicates can be used to form subsets of existing sets, but they can’t be used to form new sets completely from scratch.

The notation \( \{ x \in A \mid P(x) \} \) is a convenient way to effectively limit the domain of discourse of a predicate, \( P \), to members of a set, \( A \), that we are actually interested in. We will use a similar notation with the quantifiers \( \forall \) and \( \exists \). The proposition \( (\forall x \in A)(P(x)) \) is true if and only if \( P(a) \) is true for every element \( a \) of the set \( A \). And the proposition \( (\exists x \in A)(P(x)) \) is true if and only if there is some element \( a \) of the set \( A \) for which \( P(a) \) is true. These notations are valid only when \( A \) is contained in the domain of discourse for \( P \). As usual, we can leave out parentheses when doing so introduces no ambiguity. So, for example, we might write \( \forall x \in A \ P(x) \).

We end this section with proofs of the two forms of the principle of mathematical induction. These proofs were omitted from the previous chapter, but only for the lack of a bit of set notation. In fact, the principle of mathematical induction is valid only because it follows from one of the basic axioms that define the natural numbers, namely the fact that any non-empty set of natural numbers has a smallest element. Given this axiom, we can use it to prove the following two theorems:

**Theorem 2.3.** Let \( P \) be a one-place predicate whose domain of discourse includes the natural numbers. Suppose that \( P(0) \land (\forall k \in \mathbb{N} \ (P(k) \rightarrow P(k + 1))) \). Then \( \forall n \in \mathbb{N}, P(n) \).

**Proof.** Suppose that both \( P(0) \) and \( (\forall k \in \mathbb{N} \ (P(k) \rightarrow P(k + 1))) \) are true, but that \( (\forall n \in \mathbb{N}, P(n)) \) is false. We show that this assumption leads to a contradiction.
2.1. BASIC CONCEPTS

Since the statement $\forall n \in \mathbb{N}, P(n)$ is false, its negation, $\neg(\forall n \in \mathbb{N}, P(n))$, is true. The negation is equivalent to $\exists n \in \mathbb{N}, \neg P(n)$. Let $X = \{n \in \mathbb{N} | \neg P(n)\}$. Since $\exists n \in \mathbb{N}, \neg P(n)$ is true, we know that $X$ is not empty. Since $X$ is a non-empty set of natural numbers, it has a smallest element. Let $x$ be the smallest element of $X$. That is, $x$ is the smallest natural number such that $P(x)$ is false. Since we know that $P(0)$ is true, $x$ cannot be 0. Let $y = x - 1$. Since $x \neq 0$, $y$ is a natural number. Since $y < x$, we know, by the definition of $x$, that $P(y)$ is true. We also know that $\forall k \in \mathbb{N} (P(k) \rightarrow P(k + 1))$ is true. In particular, taking $k = y$, we know that $P(y) \rightarrow P(y + 1)$. Since $P(y)$ and $P(y) \rightarrow P(y + 1)$, we deduce by modus ponens that $P(y + 1)$ is true. But $y + 1 = x$, so we have deduced that $P(x)$ is true. This contradicts the fact that $P(x)$ is false. This contradiction proves the theorem.

**Theorem 2.4.** Let $P$ be a one-place predicate whose domain of discourse includes the natural numbers. Suppose that $P(0)$ is true and that

$$(P(0) \land P(1) \land \cdots \land P(k)) \rightarrow P(k + 1)$$

is true for each natural number $k \geq 0$. Then it is true that $\forall n \in \mathbb{N}, P(n)$.

**Proof.** Suppose that $P$ is a predicate that satisfies the hypotheses of the theorem, and suppose that the statement $\forall n \in \mathbb{N}, P(n)$ is false. We show that this assumption leads to a contradiction.

Let $X = \{n \in \mathbb{N} | \neg P(n)\}$. Because of the assumption that $\forall n \in \mathbb{N}, P(n)$ is false, $X$ is non-empty. It follows that $X$ has a smallest element. Let $x$ be the smallest element of $X$. The assumption that $P(0)$ is true means that $0 \notin X$, so we must have $x > 0$. Since $x$ is the smallest natural number for which $P(x)$ is false, we know that $P(0)$, $P(1)$, $\ldots$, and $P(x - 1)$ are all true. From this and the fact that $(P(0) \land P(1) \land \cdots \land P(x - 1)) \rightarrow P(x)$, we deduce that $P(x)$ is true. But this contradicts the fact that $P(x)$ is false. This contradiction proves the theorem.

**Exercises**

1. If we don’t make the assumption that $a$, $b$, and $c$ are distinct, then the set denoted by $\{a, b, c\}$ might actually contain either 1, 2, or 3 elements. How many different elements might the set $\{a, b, \{a, c\}, \{a, c\}, \{a, b, c\}\}$ contain? Explain your answer.

2. Compute $A \cup B$, $A \cap B$, and $A \setminus B$ for each of the following pairs of sets

   a) $A = \{a, b, c\}$, $B = \emptyset$
   b) $A = \{1, 2, 3, 4, 5\}$, $B = \{2, 4, 6, 8, 10\}$
c) $A = \{a, b\}, \ B = \{a, b, c, d\}$
d) $A = \{a, b, \{a, b\}\}, \ B = \{\{a\}, \{a, b\}\}$

3. Recall that $\mathbb{N}$ represents the set of natural numbers. That is, $\mathbb{N} = \{0, 1, 2, 3, \ldots \}$. Let $X = \{n \in \mathbb{N} | n \geq 5\}$, let $Y = \{n \in \mathbb{N} | n \leq 10\}$, and let $Z = \{n \in \mathbb{N} | n$ is an even number}. Find each of the following sets:

- $a) \ X \cap Y$
- $b) \ X \cup Y$
- $c) \ X \setminus Y$
- $d) \ N \setminus Z$
- $e) \ X \cap Z$
- $f) \ Y \cap Z$
- $g) \ Y \cup Z$
- $h) \ Z \setminus N$

4. Find $\mathcal{P}(\{1, 2, 3\})$. (It has eight elements.)

5. Assume that $a$ and $b$ are entities and that $a \neq b$. Let $A$ and $B$ be the sets defined by $A = \{a, \{b\}\}$ and $B = \{a, b, \{a, \{b\}\}\}$. Determine whether each of the following statements is true or false. Explain your answers.

- $a) \ b \in A$
- $b) \ \{a, b\} \subseteq A$
- $c) \ \{a, b\} \subseteq B$
- $d) \ \{a, b\} \in B$
- $e) \ \{a, \{b\}\} \in A$
- $f) \ \{a, \{b\}\} \in B$

6. Since $\mathcal{P}(A)$ is a set, it is possible to form the set $\mathcal{P}(\mathcal{P}(A))$. What is $\mathcal{P}(\mathcal{P}(\emptyset))$? What is $\mathcal{P}(\mathcal{P}(\{a, b\}))$? (It has sixteen elements.)

7. In the English sentence, “She likes men who are tall, dark, and handsome,” does she like an intersection or a union of sets of men? How about in the sentence, “She likes men who are tall, men who are dark, and men who are handsome?”

8. If $A$ is any set, what can you say about $A \cup A$? About $A \cap A$? About $A \setminus A$? Why?

9. Suppose that $A$ and $B$ are sets such that $A \subseteq B$. What can you say about $A \cup B$? About $A \cap B$? About $A \setminus B$? Why?

10. Suppose that $A$, $B$, and $C$ are sets. Show that $C \subseteq A \cap B$ if and only if $(C \subseteq A) \land (C \subseteq B)$.

11. Suppose that $A$, $B$, and $C$ are sets, and that $A \subseteq B$ and $B \subseteq C$. Show that $A \subseteq C$.

12. Suppose that $A$ and $B$ are sets such that $A \subseteq B$. Is it necessarily true that $\mathcal{P}(A) \subseteq \mathcal{P}(B)$? Why or why not?

13. Let $M$ be any natural number, and let $P(n)$ be a predicate whose domain of discourse includes all natural numbers greater than or equal to $M$. Suppose that $P(M)$ is true, and suppose that $P(k) \rightarrow P(k + 1)$ for all $k \geq M$. Show that $P(n)$ is true for all $n \geq M$.

2.2 The Boolean Algebra of Sets

It is clear that set theory is closely related to logic. The intersection and union of sets can be defined in terms of the logical “and” and logical “or” operators. The notation $\{x | P(x)\}$ makes it possible to use predicates to specify sets. And if $A$ is any set, then the formula $x \in A$ defines a one place predicate that is true for an entity $x$ if and only if $x$ is a member of $A$. So
it should not be a surprise that many of the rules of logic have analogs in set theory.

For example, we have already noted that \( \cup \) and \( \cap \) are commutative operations. This fact can be verified using the rules of logic. Let \( A \) and \( B \) be sets. According to the definition of equality of sets, we can show that \( A \cup B = B \cup A \) by showing that \( \forall x \left( (x \in A \cup B) \iff (x \in B \cup A) \right) \). But for any \( x \),

\[
x \in A \cup B \iff x \in A \lor x \in B \quad \text{(definition of } \cup) \\
\iff x \in B \lor x \in A \quad \text{(commutativity of } \lor) \\
\iff x \in B \cup A \quad \text{(definition of } \cup)
\]

The commutativity of \( \cap \) follows in the same way from the definition of \( \cap \) in terms of \( \land \) and the commutativity of \( \land \), and a similar argument shows that union and intersection are associative operations.

The distributive laws for propositional logic give rise to two similar rules in set theory. Let \( A \), \( B \), and \( C \) be any sets. Then

\[
A \cup (B \cap C) = (A \cup B) \cap (A \cup C)
\]

and

\[
A \cap (B \cup C) = (A \cap B) \cup (A \cap C)
\]

These rules are called the distributive laws for set theory. To verify the first of these laws, we just have to note that for any \( x \),

\[
x \in A \cup (B \cap C) \\
\iff (x \in A) \lor ((x \in B) \land (x \in C)) \quad \text{(definition of } \cup, \land) \\
\iff ((x \in A) \lor (x \in B)) \land ((x \in A) \lor (x \in C)) \quad \text{(distributivity of } \lor) \\
\iff (x \in A \cup B) \land (x \in A \cup C) \quad \text{(definition of } \cup) \\
\iff x \in ((A \cup B) \cap (A \cup C)) \quad \text{(definition of } \cap)
\]

The second distributive law for sets follows in exactly the same way.

While \( \cup \) is analogous to \( \lor \) and \( \cap \) is analogous to \( \land \), we have not yet seen any operation is set theory that is analogous to the logical “not” operator, \( \neg \). Given a set \( A \), it is tempting to try to define \( \{ x \mid \neg (x \in A) \} \), the set that contains everything that does not belong to \( A \). Unfortunately, the rules of set theory do not allow us to define such a set. The notation \( \{ x \mid P(x) \} \) can only be used when the domain of discourse of \( P \) is a set, so there
must be an underlying set from which the elements that are/are not in \( A \) are chosen, i.e. some underlying set of which \( A \) is a subset. We can get around this problem by restricting the discussion to subsets of some fixed set. This set will be known as the **universal set**. Keep in mind that the universal set is only universal for some particular discussion. It is simply some set that is large enough to contain all the sets under discussion as subsets. Given a universal set \( U \) and any subset \( A \) of \( U \), we can define the set \( \{ x \in U \mid \neg(x \in A) \} \).

**Definition 2.1.** Let \( U \) be a given universal set, and let \( A \) be any subset of \( U \). We define the **complement** of \( A \) in \( U \) to be the set \( \overline{A} \) that is defined by \( \overline{A} = \{ x \in U \mid x \notin A \} \).

Usually, we will refer to the complement of \( A \) in \( U \) simply as the complement of \( A \), but you should remember that whenever complements of sets are used, there must be some universal set in the background.

Given the complement operation on sets, we can look for analogs to the rules of logic that involve negation. For example, we know that \( p \land \neg p = \bot \) for any proposition \( p \). It follows that for any subset \( A \) of \( U \),

\[
A \cap \overline{A} = \{ x \in U \mid (x \in A) \land (x \notin A) \} \quad \text{(definition of } \cap) \\
= \{ x \in U \mid (x \in A) \land (x \notin A) \} \quad \text{(definition of complement)} \\
= \{ x \in U \mid (x \in A) \land \neg(x \in A) \} \quad \text{(definition of } \notin) \\
= \emptyset
\]

the last equality following because the proposition \((x \in A) \land \neg(x \in A)\) is false for any \( x \). Similarly, we can show that \( A \cup \overline{A} = U \) and that \( \overline{\overline{A}} = A \) (where \( \overline{A} \) is the complement of the complement of \( A \), that is, the set obtained by taking the complement of \( \overline{A} \)).

The most important laws for working with complements of sets are DeMorgan’s Laws for sets. These laws, which follow directly from DeMorgan’s Laws for logic, state that for any subsets \( A \) and \( B \) of a universal set \( U \),

\[
\overline{A \cup B} = \overline{A} \cap \overline{B}
\]

and

\[
\overline{A \cap B} = \overline{A} \cup \overline{B}
\]
Double complement \(\overline{A} = A\)

Miscellaneous laws
\(A \cup \overline{A} = U\)
\(A \cap \overline{A} = \emptyset\)
\(\emptyset \cup A = A\)
\(\emptyset \cap A = \emptyset\)

Idempotent laws
\(A \cap A = A\)
\(A \cup A = A\)

Commutative laws
\(A \cap B = B \cap A\)
\(A \cup B = B \cup A\)

Associative laws
\(A \cap (B \cap C) = (A \cap B) \cap C\)
\(A \cup (B \cup C) = (A \cup B) \cup C\)

Distributive laws
\(A \cap (B \cup C) = (A \cap B) \cup (A \cap C)\)
\(A \cup (B \cap C) = (A \cup B) \cap (A \cup C)\)

DeMorgan’s laws
\(\overline{A \cup B} = \overline{A} \cap \overline{B}\)
\(\overline{A \cap B} = \overline{A} \cup \overline{B}\)

Figure 2.2: Some Laws of Boolean Algebra for sets. \(A, B,\) and \(C\) are sets. For the laws that involve the complement operator, they are assumed to be subsets of some universal set, \(U\). For the most part, these laws correspond directly to laws of Boolean Algebra for propositional logic as given in Figure 1.2.

For example, we can verify the first of these laws with the calculation

\[
\overline{A \cup B} = \{x \in U \mid x \not\in (A \cup B)\} \\
= \{x \in U \mid \neg(x \in A \cup B)\} \\
= \{x \in U \mid \neg(x \in A \lor x \in B)\} \\
= \{x \in U \mid (\neg(x \in A)) \land (\neg(x \in B))\} \\
= \{x \in U \mid (x \not\in A) \land (x \not\in B)\} \\
= \overline{A \cap B}
\]

An easy inductive proof can be used to verify generalized versions of DeMorgan’s Laws for set theory. (In this context, all sets are assumed to
be subsets of some unnamed universal set.) A simple calculation verifies DeMorgan’s Law for three sets:

\[
A \cup B \cup C = (A \cup B) \cup C = (A \cup B) \cap C = (A \cap B) \cap C
\]

(by DeMorgan’s Law for two sets)

From there, we can derive similar laws for four sets, five sets, and so on. However, just saying “and so on” is not a rigorous proof of this fact. Here is a rigorous inductive proof of a generalized DeMorgan’s Law:

**Theorem 2.5.** For any natural number \( n \geq 2 \) and for any sets \( X_1, X_2, \ldots, X_n \),

\[
X_1 \cup X_2 \cup \cdots \cup X_n = X_1 \cap X_2 \cap \cdots \cap X_n
\]

**Proof.** We give a proof by induction. In the base case, \( n = 2 \), the statement is that \( \overline{X_1 \cup X_2} = \overline{X_1} \cap \overline{X_2} \). This is true since it is just an application of DeMorgan’s law for two sets.

For the inductive case, suppose that the statement is true for \( n = k \). We want to show that it is true for \( n = k + 1 \). Let \( X_1, X_2, \ldots, X_{k+1} \) be any \( k \) sets. Then we have:

\[
\overline{X_1 \cup X_2 \cup \cdots \cup X_{k+1}} = \overline{(X_1 \cup X_2 \cup \cdots \cup X_k) \cup X_{k+1}} = (X_1 \cap X_2 \cap \cdots \cap X_k) \cap X_{k+1} = \overline{X_1} \cap \overline{X_2} \cap \cdots \cap \overline{X_{k+1}}
\]

In this computation, the second step follows by DeMorgan’s Law for two sets, while the third step follows from the induction hypothesis.

Just as the laws of logic allow us to do algebra with logical formulas, the laws of set theory allow us to do algebra with sets. Because of the close relationship between logic and set theory, their algebras are very similar. The algebra of sets, like the algebra of logic, is Boolean algebra. When George Boole wrote his 1854 book about logic, it was really as much about set theory as logic. In fact, Boole did not make a clear distinction between a predicate and the set of objects for which that predicate is true. His algebraic laws and formulas apply equally to both cases. More exactly, if we consider only subsets of some given universal set \( U \), then there is a direct correspondence between the basic symbols and operations of propositional
logic and certain symbols and operations in set theory, as shown in this table:

<table>
<thead>
<tr>
<th>Logic</th>
<th>Set Theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>$U$</td>
</tr>
<tr>
<td>$F$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$p \land q$</td>
<td>$A \cap B$</td>
</tr>
<tr>
<td>$p \lor q$</td>
<td>$A \cup B$</td>
</tr>
<tr>
<td>$\neg p$</td>
<td>$\overline{A}$</td>
</tr>
</tbody>
</table>

Any valid logical formula or computation involving propositional variables and the symbols $T$, $F$, $\land$, $\lor$, and $\neg$ can be transformed into a valid formula or computation in set theory by replacing the propositions in the formula with subsets of $U$ and replacing the logical symbols with $U$, $\emptyset$, $\cap$, $\cup$, and the complement operator.

Just as in logic, the operations of set theory can be combined to form complex expressions such as $(A \cup C) \cap (B \cup C \cup D)$. Parentheses can always be used in such expressions to specify the order in which the operations are to be performed. In the absence of parentheses, we need precedence rules to determine the order of operation. The precedence rules for the Boolean algebra of sets are carried over directly from the Boolean algebra of propositions. When union and intersection are used together without parentheses, intersection has precedence over union. Furthermore, when several operators of the same type are used without parentheses, then they are evaluated in order from left to right. (Of course, since $\cup$ and $\cap$ are both associative operations, it really doesn’t matter whether the order of evaluation is left-to-right or right-to-left.) For example, $A \cup B \cap C \cup D$ is evaluated as $(A \cup ((B \cap C)) \cup D$. The complement operation is a special case. Since it is denoted by drawing a line over its operand, there is never any ambiguity about which part of a formula it applies to.

The laws of set theory can be used to simplify complex expressions involving sets. (As usual, of course, the meaning of “simplification” is partly in the eye of the beholder.) For example, for any sets $X$ and $Y$,

$$(X \cup Y) \cap (Y \cup X) = (X \cup Y) \cap (X \cup Y)$$  \hspace{1cm} \text{(Commutative Law)}

$$= (X \cup Y)$$  \hspace{1cm} \text{(Idempotent Law)}

where in the second step, the Idempotent Law, which says that $A \cap A = A$, is applied with $A = X \cup Y$. For expressions that use the complement operation, it is usually considered to be simpler to apply the operation to an
individual set, as in \( A \), rather than to a formula, as in \( A \cap B \). DeMorgan’s Laws can always be used to simplify an expression in which the complement operation is applied to a formula. For example,

\[
A \cap B \cup \overline{A} = A \cap (B \cap \overline{A}) \quad \text{(DeMorgan’s Law)}
\]
\[
= A \cap (B \cap A) \quad \text{(Double Complement)}
\]
\[
= A \cap (A \cap B) \quad \text{(Commutative Law)}
\]
\[
= (A \cap A) \cap B \quad \text{(Associative Law)}
\]
\[
= A \cap B \quad \text{(Idempotent Law)}
\]

As a final example of the relationship between set theory and logic, consider the set-theoretical expression \( A \cap (A \cup B) \) and the corresponding compound proposition \( p \land (p \lor q) \). (These correspond since for any \( x \), \( x \in A \cap (A \cup B) \equiv (x \in A) \land ((x \in A) \lor (x \in B)) \).) You might find it intuitively clear that \( A \cap (A \cup B) = A \). Formally, this follows from the fact that \( p \land (p \lor q) \equiv p \), which might be less intuitively clear and is surprising difficult to prove algebraically from the laws of logic. However, there is another way to check that a logical equivalence is valid: Make a truth table. Consider a truth table for \( p \land (p \lor q) \):

<table>
<thead>
<tr>
<th>( p )</th>
<th>( q )</th>
<th>( p \lor q )</th>
<th>( p \land (p \lor q) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

The fact that the first column and the last column of this table are identical shows that \( p \land (p \lor q) \equiv p \). Taking \( p \) to be the proposition \( x \in A \) and \( q \) to be the proposition \( x \in B \), it follows that the sets \( A \) and \( A \cap (A \cup B) \) have the same members and therefore are equal.

**Exercises**

1. Use the laws of logic to verify the associative laws for union and intersection. That is, show that if \( A \), \( B \), and \( C \) are sets, then \( A \cup (B \cup C) = (A \cup B) \cup C \) and \( A \cap (B \cap C) = (A \cap B) \cap C \).
2. Show that for any sets \( A \) and \( B \), \( A \subseteq A \cup B \) and \( A \cap B \subseteq A \).
3. Recall that the symbol \( \oplus \) denotes the logical exclusive or operation. If \( A \) and \( B \) sets, define the set \( A \triangle B \) by \( A \triangle B = \{ x \mid (x \in A) \oplus (x \in B) \} \). Show that \( A \triangle B = (A \setminus B) \cup (B \setminus A) \). \( A \triangle B \) is known as the symmetric difference of \( A \) and \( B \).
4. Let $A$ be a subset of some given universal set $U$. Verify that $\overline{A} = A$ and that $A \cup \overline{A} = U$.

5. Verify the second of DeMorgan’s Laws for sets, $\overline{A \cap B} = \overline{A} \cup \overline{B}$. For each step in your verification, state why that step is valid.

6. The subset operator, $\subseteq$, is defined in terms of the logical implication operator, $\rightarrow$. However, $\subseteq$ differs from the $\cap$ and $\cup$ operators in that $A \cap B$ and $A \cup B$ are sets, while $A \subseteq B$ is a statement. So the relationship between $\subseteq$ and $\rightarrow$ isn’t quite the same as the relationship between $\cup$ and $\lor$ or between $\cap$ and $\land$. Nevertheless, $\subseteq$ and $\rightarrow$ do share some similar properties. This problem shows one example.
   a) Show that the following three compound propositions are logically equivalent: $p \rightarrow q$, $(p \land q) \leftrightarrow p$, and $(p \lor q) \leftrightarrow q$.
   b) Show that for any sets $A$ and $B$, the following three statements are equivalent: $A \subseteq B$, $A \cap B = A$, and $A \cup B = B$.

7. DeMorgan’s Laws apply to subsets of some given universal set $U$. Show that for a subset $X$ of $U$, $\overline{X} = U \setminus X$. It follows that DeMorgan’s Laws can be written as $U \setminus (A \cup B) = (U \setminus A) \cap (U \setminus B)$ and $U \setminus (A \cap B) = (U \setminus A) \cup (U \setminus B)$. Show that these laws hold whether or not $A$ and $B$ are subsets of $U$. That is, show that for any sets $A$, $B$, and $C$, $C \setminus (A \cup B) = (C \setminus A) \cap (C \setminus B)$ and $C \setminus (A \cap B) = (C \setminus A) \cup (C \setminus B)$.

8. Show that $A \cup (A \cap B) = A$ for any sets $A$ and $B$.

9. Let $X$ and $Y$ be sets. Simplify each of the following expressions. Justify each step in the simplification with one of the rules of set theory.
   a) $X \cup (Y \cup X)$  
   b) $(X \cap Y) \cap \overline{X}$
   c) $(X \cup Y) \cap \overline{Y}$  
   d) $(X \cup Y) \cup (X \cap Y)$

10. Let $A$, $B$, and $C$ be sets. Simplify each of the following expressions. In your answer, the complement operator should only be applied to the individual sets $A$, $B$, and $C$.
    a) $A \cup B \cup C$  
    b) $A \cup B \cap C$  
    c) $A \cup \overline{B}$
    d) $\overline{B} \cap \overline{C}$  
    e) $A \cap B \cap \overline{C}$  
    f) $A \cap \overline{A} \cup \overline{B}$

11. Use induction to prove the following generalized DeMorgan’s Law for set theory: For any natural number $n \geq 2$ and for any sets $X_1$, $X_2$, $\ldots$, $X_n$,
    
    \[ \overline{X_1 \cap X_2 \cap \cdots \cap X_n} = X_1 \cup X_2 \cup \cdots \cup X_n \]

12. State and prove generalized distributive laws for set theory.

### 2.3 Application: Programming with Sets

On a computer, all data are represented, ultimately, as strings of zeros and ones. At times, computers need to work with sets. How can sets be represented as strings of zeros and ones?
A set is determined by its elements. Given a set \( A \) and an entity \( x \), the fundamental question is, does \( x \) belong to \( A \) or not? If we know the answer to this question for each possible \( x \), then we know the set. For a given \( x \), the answer to the question, “Is \( x \) a member of \( A \)?” is either yes or no. The answer, then, is a single bit, that is, a value that can be either zero or one. To represent the set \( A \) as a string of zeros and ones, we could use one bit for each possible member of \( A \). If a possible member \( x \) is in the set, then the corresponding bit has the value one. If \( x \) is not in the set, then the corresponding bit has the value zero.

Now, in cases where the number of possible elements of the set is very large or infinite, it is not practical to represent the set in this way. It would require too many bits, perhaps an infinite number. In such cases, some other representation for the set can be used. However, suppose we are only interested in subsets of some specified small set. Since this set plays the role of a universal set, let’s call it \( U \). To represent a subset of \( U \), we just need one bit for each member of \( U \). If the number of members of \( U \) is \( n \), then a subset of \( U \) is represented by a string of \( n \) zeros and ones. Furthermore, every string of \( n \) zeros and ones determines a subset of \( U \), namely that subset that contains exactly the elements of \( U \) that correspond to ones in the string. A string of \( n \) zeros and ones is called an \( n \)-bit binary number. So, we see that if \( U \) is a set with \( n \) elements, then the subsets of \( U \) correspond to \( n \)-bit binary numbers.

To make things more definite, let \( U \) be the set \( \{0, 1, 2, \ldots, 31\} \). This set consists of the 32 integers between 0 and 31, inclusive. Then each subset of \( U \) can be represented by a 32-bit binary number. We use 32 bits because most computer languages can work directly with 32-bit numbers. For example, the programming languages Java, C, and C++ have a data type named int. A value of type int is a 32-bit binary number.\(^1\) Before we get a definite correspondence between subsets of \( U \) and 32-bit numbers, we have to decide which bit in the number will correspond to each member of \( U \). Following tradition, we assume that the bits are numbered from right to left. That is, the rightmost bit corresponds to the element 0 in \( U \), the second bit from the right corresponds to 1, the third bit from the right to 2, and so on. For example, the 32-bit number

\[
10000000000000000000100110110
\]

corresponds to the subset \( \{1, 2, 4, 5, 6, 9, 31\} \). Since the leftmost bit of the

\(^1\)Actually, in some versions of C and C++, a value of type int is a 16-bit number. A 16-bit number can be used to represent a subset of the set \( \{0, 1, 2, \ldots, 15\} \). The principle, of course, is the same.
2.3. APPLICATION: PROGRAMMING WITH SETS

<table>
<thead>
<tr>
<th>Hex.</th>
<th>Binary</th>
<th>Hex.</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000₂</td>
<td>8</td>
<td>1000₂</td>
</tr>
<tr>
<td>1</td>
<td>0001₂</td>
<td>9</td>
<td>1001₂</td>
</tr>
<tr>
<td>2</td>
<td>0010₂</td>
<td>A</td>
<td>1010₂</td>
</tr>
<tr>
<td>3</td>
<td>0011₂</td>
<td>B</td>
<td>1011₂</td>
</tr>
<tr>
<td>4</td>
<td>0100₂</td>
<td>C</td>
<td>1100₂</td>
</tr>
<tr>
<td>5</td>
<td>0101₂</td>
<td>D</td>
<td>1101₂</td>
</tr>
<tr>
<td>6</td>
<td>0110₂</td>
<td>E</td>
<td>1110₂</td>
</tr>
<tr>
<td>7</td>
<td>0111₂</td>
<td>F</td>
<td>1111₂</td>
</tr>
</tbody>
</table>

Figure 2.3: The 16 hexadecimal digits and the corresponding binary numbers. Each hexadecimal digit corresponds to a 4-bit binary number. Longer binary numbers can be written using two or more hexadecimal digits. For example, 10100011111₂ = 0xA1F.

number is 1, the number 31 is in the set; since the next bit is 0, the number 30 is not in the set; and so on.

From now on, I will write binary numbers with a subscript of 2 to avoid confusion with ordinary numbers. Furthermore, I will often leave out leading zeros. For example, 1101₂ is the binary number that would be written out in full as

```
0000000000000000000000000001101
```

and which corresponds to the set \{0, 2, 3\}. On the other hand 1101 represents the ordinary number one thousand one hundred and one.

Even with this notation, it can be very annoying to write out long binary numbers—and almost impossible to read them. So binary numbers are never written out as sequences of zeros and ones in computer programs. An alternative is to use hexadecimal numbers. Hexadecimal numbers are written using the sixteen symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. These symbols are known as the hexadecimal digits. Each hexadecimal digit corresponds to a 4-bit binary number, as shown in Figure 2.3. To represent a longer binary number, several hexadecimal digits can be strung together. For example, the hexadecimal number C7 represents the binary number 11000111₂. In Java and many related languages, a hexadecimal number is written with the prefix “0x”. Thus, the hexadecimal number C7 would appear in the program as 0xC7. I will follow the
same convention here. Any 32-bit binary number can be written using eight hexadecimal digits (or fewer if leading zeros are omitted). Thus, subsets of \{0, 1, 2, \ldots, 31\} correspond to 8-digit hexadecimal numbers. For example, the subset \{1, 2, 4, 5, 6, 9, 31\} corresponds to 0x80000276, which represents the binary number 10000000000000000000010011101102. Similarly, 0xFF corresponds to \{0, 1, 2, 3, 4, 5, 6, 7\} and 0x1101 corresponds to the binary number 00010001000000012 and to the set \{0, 8, 12\}.

Now, if you have worked with binary numbers or with hexadecimal numbers, you know that they have another, more common interpretation. They represent ordinary integers. Just as 342 represents the integer \(3 \cdot 10^2 + 4 \cdot 10^1 + 2 \cdot 10^0\), the binary number 11012 represents the integer \(1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0\), or 13. When used in this way, binary numbers are known as **base-2 numbers**, just as ordinary numbers are base-10 numbers. Hexadecimal numbers can be interpreted as base-16 numbers. For example, 0x3C7 represents the integer \(3 \cdot 16^2 + 12 \cdot 16^1 + 7 \cdot 16^0\), or 874. So, does 11012 really represent the integer 13, or does it represent the set \{0, 2, 3\}? The answer is that to a person, 11012 can represent either. Both are valid interpretations, and the only real question is which interpretation is useful in a given circumstance. On the other hand, to the computer, 11012 doesn’t represent anything. It’s just a string of bits, and the computer manipulates the bits according to its program, without regard to their interpretation.

Of course, we still have to answer the question of whether it is ever useful to interpret strings of bits in a computer as representing sets.

If all we could do with sets were to “represent” them, it wouldn’t be very useful. We need to be able to compute with sets. That is, we need to be able to perform set operations such as union and complement.

Many programming languages provide operators that perform set operations. In Java and related languages, the operators that perform union, intersection, and complement are written as |, &, and ~. For example, if \(x\) and \(y\) are 32-bit integers representing two subsets, \(X\) and \(Y\), of \{0, 1, 2, \ldots, 31\}, then \(x | y\) is a 32-bit integer that represents the set \(X \cup Y\). Similarly, \(x \& y\) represents the set \(X \cap Y\), and \(^\sim x\) represents the complement, \(^\sim X\).

The operators |, &, and ~ are called **bitwise logical operators** because of the way they operate on the individual bits of the numbers to which they are applied. If 0 and 1 are interpreted as the logical values false and true, then the bitwise logical operators perform the logical operations \(\lor\), \(\land\), and \(^\sim\) on individual bits. To see why this is true, let’s look at the computations that these operators have to perform.

Let \(k\) be one of the members of \{0, 1, 2, \ldots, 31\}. In the binary numbers \(x, y, x | y, x \& y,\) and \(^\sim x\), the number \(k\) corresponds to the bit in position \(k\).
That is, \( k \) is in the set represented by a binary number if and only if the bit
in position \( k \) in that binary number is 1. Considered as sets, \( x \& y \) is the
intersection of \( x \) and \( y \), so \( k \) is a member of the set represented by \( x \& y \) if
and only if \( k \) is a member of both of the sets represented by \( x \) and \( y \). That
is, bit \( k \) is 1 in the binary number \( x \& y \) if and only if bit \( k \) is 1 in \( x \) and bit
\( k \) is 1 in \( y \). When we interpret 1 as \textit{true} and 0 as \textit{false}, we see that bit \( k \)
of \( x \& y \) is computed by applying the logical “and” operator, \( \land \), to bit \( k \) of \( x \)
and bit \( k \) of \( y \). Similarly, bit \( k \) of \( x | y \) is computed by applying the logical
“or” operator, \( \lor \), to bit \( k \) of \( x \) and bit \( k \) of \( y \). And bit \( k \) of \( \sim x \) is computed
by applying the logical “not” operator, \( \neg \), to bit \( k \) of \( x \). In each case, the
logical operator is applied to each bit position separately. (Of course, this
discussion is just a translation to the language of bits of the definitions of
the set operations in terms of logical operators: \( A \cap B = \{ x \mid x \in A \land x \in B \} \),
\( A \cup B = \{ x \mid x \in A \lor x \in B \} \), and \( \overline{A} = \{ x \in U \mid \neg(x \in A) \} \).)

For example, consider the binary numbers \( 1011010_2 \) and \( 10111_2 \), which
represent the sets \( \{1, 3, 4, 6\} \) and \( \{0, 1, 2, 4\} \). Then \( 1011010_2 \& 10111_2 \) is
\( 1011010_2 \). This binary number represents the set \( \{1, 4\} \), which is the inter-
section \( \{1, 3, 4, 6\} \cap \{0, 1, 2, 4\} \). It’s easier to see what’s going on if we write
out the computation in columns, the way you probably first learned to do
addition:

\[
\begin{array}{c|c}
1011010 & \{6, 4, 3, 1\} \\
0010111 & \{4, 2, 1, 0\} \\
\hline
0010010 & \{4, 1\}
\end{array}
\]

Note that in each column in the binary numbers, the bit in the bottom row
is computed as the logical “and” of the two bits that lie above it in the
column. I’ve written out the sets that correspond to the binary numbers
to show how the bits in the numbers correspond to the presence or absence
of elements in the sets. Similarly, we can see how the union of two sets is
computed as a bitwise “or” of the corresponding binary numbers.

\[
\begin{array}{c|c}
1011010 & \{6, 4, 3, 1\} \\
| 0010111 & \{4, 2, 1, 0\} \\
\hline
1011111 & \{6, 4, 3, 2, 1, 0\}
\end{array}
\]

The complement of a set is computed using a bitwise “not” operation. Since
we are working with 32-bit binary numbers, the complement is taken with
respect to the universal set \( \{0, 1, 2, \ldots, 31\} \). So, for example,

\[
\sim1011010_2 = 111111111111111111111111100101_2
\]

Of course, we can apply the operators \&, \|, and \( \sim \) to numbers written in
hexadecimal form, or even in ordinary, base-10 form. When doing such
calculations by hand, it is probably best to translate the numbers into binary form. For example,

$$\begin{align*}
0xAB7 \& 0x168E &= 101010111011_2 \& 1011010001110_2 \\
&= 0001010000110_2 \\
&= 0x286
\end{align*}$$

When computing with sets, it is sometimes necessary to work with individual elements. Typical operations include adding an element to a set, removing an element from a set, and testing whether an element is in a set. However, instead of working with an element itself, it’s convenient to work with the set that contains that element as its only member. For example, testing whether $5 \in A$ is the same as testing whether $\{5\} \cap A \neq \emptyset$. The set $\{5\}$ is represented by the binary number $100000_2$ or by the hexadecimal number $0x20$. Suppose that the set $A$ is represented by the number $x$. Then, testing whether $5 \in A$ is equivalent to testing whether $0x20 \& x \neq 0$. Similarly, the set $A \cup \{5\}$, which is obtained by adding 5 to $A$, can be computed as $x \mid 0x20$. The set $A \setminus \{5\}$, which is the set obtained by removing 5 from $A$ if it occurs in $A$, is represented by $x \& \neg 0x20$.

The sets $\{0\}$, $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{6\}$, $\ldots$, $\{31\}$ are represented by the hexadecimal numbers $0x1$, $0x2$, $0x4$, $0x8$, $0x10$, $0x20$, $\ldots$, $0x8000000$. In typical computer applications, some of these numbers are given names, and these names are thought of as names for the possible elements of a set (although, properly speaking, they are names for sets containing those elements). Suppose, for example, that $a$, $b$, $c$, and $d$ are names for four of the numbers from the above list. Then $a \mid c$ is the set that contains the two elements corresponding to the numbers $a$ and $c$. If $x$ is a set, then $x \& \neg d$ is the set obtained by removing $d$ from $x$. And we can test whether $b$ is in $x$ by testing if $x \& b \neq 0$.

Here is an actual example, which is used in the Macintosh operating system. Characters can be printed or displayed on the screen in various sizes and styles. A font is a collection of pictures of characters in a particular size and style. On the Macintosh, a basic font can be modified by specifying any of the following style attributes: bold, italic, underline, outline, shadow, condense, and extend. The style of a font is a subset of this set of attributes. A style set can be specified by or-ing together individual attributes. For example, an underlined, bold, italic font has style set underline $|$ bold $|$ italic. For a plain font, with none of the style attributes set, the style set is the empty set, which is represented by the number zero.

The Java programming language uses a similar scheme to specify style attributes for fonts, but currently there are only two basic attributes,
2.3. APPLICATION: PROGRAMMING WITH SETS

Font.BOLD and Font.ITALIC. A more interesting example in Java is provided by event types. An event in Java represents some kind of user action, such as pressing a key on the keyboard. Events are associated with “components” such as windows, push buttons, and scroll bars. Components can be set to ignore a given type of event. We then say that that event type is disabled for that component. If a component is set to process events of a given type, then that event type is said to be enabled. Each component keeps track of the set of event types that are currently enabled. It will ignore any event whose type is not in that set. Each event type has an associated constant with a name such as AWTEvent.MOUSE_EVENT_MASK. These constants represent the possible elements of a set of event types. A set of event types can be specified by or-ing together a number of such constants. If \( c \) is a component and \( x \) is a number representing a set of event types, then the command “\( c.\text{enableEvents}(x) \)” enables the events in the set \( x \) for the component \( c \). If \( y \) represents the set of event types that were already enabled for \( c \), then the effect of this command is to replace \( y \) with the union, \( y \cup x \). Another command, “\( c.\text{disableEvents}(x) \)”, will disable the event types in \( x \) for the component \( c \). It does this by replacing the current set, \( y \), with \( y \& \sim x \).

Exercises

1. Suppose that the numbers \( x \) and \( y \) represent the sets \( A \) and \( B \). Show that the set \( A \setminus B \) is represented by \( x \& (\sim y) \).

2. Write each of the following binary numbers in hexadecimal:
   a) 101101102
   b) 102
   c) 111100011112
   d) 1010012

3. Write each of the following hexadecimal numbers in binary:
   a) 0x123
   b) 0xFADE
   c) 0x137F
   d) 0xFF11

4. Give the value of each of the following expressions as a hexadecimal number:
   a) \( 0x73 \mid 0x56A \)
   b) \( \sim 0x3FF0A2FF \)
   c) \((0x44 \mid 0x95) \& 0xE7 \)
   d) \( 0x5C35A7 \& 0xFF00 \)
   e) \( 0x5C35A7 \& \sim 0xFF00 \)
   f) \( \sim (0x1234 \& 0x4321) \)

5. Find a calculator (or a calculator program on a computer) that can work with hexadecimal numbers. Write a short report explaining how to work with hexadecimal numbers on that calculator. You should explain, in particular, how the calculator can be used to do the previous problem.

6. This question assumes that you know how to add binary numbers. Suppose \( x \) and \( y \) are binary numbers. Under what circumstances will the binary numbers \( x + y \) and \( x \cup y \) be the same?

7. In addition to hexadecimal numbers, the programming languages Java, C, and C++ support octal numbers. Look up and report on octal numbers in
Java, C, or C++. Explain what octal numbers are, how they are written, and how they are used.

8. In the UNIX (or Linux) operating system, every file has an associated set of permissions, which determine who can use the file and how it can be used. The set of permissions for a given file is represented by a nine-bit binary number. This number is sometimes written as an octal number. Research and report on the UNIX systems of permissions. What set of permissions is represented by the octal number 752? by the octal number 622? Explain what is done by the UNIX commands “chmod g+rw filename” and “chmod o-w filename” in terms of sets. (Hint: Look at the man page for the chmod command. To see the page, use the UNIX command “man chmod”. If you don’t know what this means, you probably don’t know enough about UNIX to do this exercise.)

9. Java, C, and C++ each have a boolean data type that has the values true and false. The usual logical and, or, and not operators on boolean values are represented by the operators &&, ||, and!. C and C++ allow integer values to be used in places where boolean values are expected. In this case, the integer zero represents the boolean value false while any non-zero integer represents the boolean value true. This means that if x and y are integers, then both x & y and x && y are valid expressions, and both can be considered to represent boolean values. Do the expressions x & y and x && y always represent the same boolean value, for any integers x and y? Do the expressions x | y and x || y always represent the same boolean values? Explain your answers.

10. Suppose that you, as a programmer, want to write a subroutine that will open a window on the computer’s screen. The window can have any of the following options: a close box, a zoom box, a resize box, a minimize box, a vertical scroll bar, a horizontal scroll bar. Design a scheme whereby the options for the window can be specified by a single parameter to the subroutine. The parameter should represent a set of options. How would you use your subroutine to open a window that has a close box and both scroll bars and no other options? Inside your subroutine, how would you determine which options have been specified for the window?

2.4 Functions

Both the real world and the world of mathematics are full of what are called, in mathematics, “functional relationships.” A functional relationship is a relationship between two sets, which associates exactly one element from the second set to each element of the first set.

For example, each item for sale in a store has a price. The first set in this relationship is the set of items in the store. For each item in the store, there is an associated price, so the second set in the relationship is the set of
possible prices. The relationship is a functional relationship because each item has a price. That is, the question “What is the price of this item?” has a single, definite answer for each item in the store.

Similarly, the question “Who is the (biological) mother of this person?” has a single, definite answer for each person. So, the relationship “mother of” defines a functional relationship. In this case, the two sets in the relationship are the same set, namely the set of people. On the other hand, the relationship “child of” is not a functional relationship. The question “Who is the child of this person?” does not have a single, definite answer for each person. A given person might not have any child at all. And a given person might have more than one child. Either of these cases—a person with no child or a person with more than one child—is enough to show that the relationship “child of” is not a functional relationship.

Or consider an ordinary map, such as a map of New York State or a street map of Rome. The whole point of the map, if it is accurate, is that there is a functional relationship between points on the map and points on the surface of the Earth. Perhaps because of this example, a functional relationship is sometimes called a mapping.

There are also many natural examples of functional relationships in mathematics. For example, every rectangle has an associated area. This fact expresses a functional relationship between the set of rectangles and the set of numbers. Every natural number \( n \) has a square, \( n^2 \). The relationship “square of” is a functional relationship from the set of natural numbers to itself.

In mathematics, of course, we need to work with functional relationships in the abstract. To do this, we introduce the idea of function. You should think of a function as a mathematical object that expresses a functional relationship between two sets. The notation \( f : A \to B \) expresses the fact that \( f \) is a function from the set \( A \) to the set \( B \). That is, \( f \) is a name for a mathematical object that expresses a functional relationship from the set \( A \) to the set \( B \). The notation \( f : A \to B \) is read as “\( f \) is a function from \( A \) to \( B \)” or more simply as “\( f \) maps \( A \) to \( B \).”

If \( f : A \to B \) and if \( a \in A \), the fact that \( f \) is a functional relationship from \( A \) to \( B \) means that \( f \) associates some element of \( B \) to \( a \). That element is denoted \( f(a) \). That is, for each \( a \in A \), \( f(a) \in B \) and \( f(a) \) is the single, definite answer to the question “What element of \( B \) is associated to \( a \) by the function \( f \)?” The fact that \( f \) is a function from \( A \) to \( B \) means that this question has a single, well-defined answer. Given \( a \in A \), \( f(a) \) is called the value of the function \( f \) at \( a \).

\(^2\)I’m avoiding here the question of Adam and Eve or of pre-human ape-like ancestors. (Take your pick.)
For example, if \( I \) is the set of items for sale in a given store and \( M \) is the set of possible prices, then there is function \( c: I \rightarrow M \) which is defined by the fact that for each \( x \in I \), \( c(x) \) is the price of the item \( x \). Similarly, if \( P \) is the set of people, then there is a function \( m: P \rightarrow P \) such that for each person \( p \), \( m(p) \) is the mother of \( p \). And if \( \mathbb{N} \) is the set of natural numbers, then the formula \( s(n) = n^2 \) specifies a function \( s: \mathbb{N} \rightarrow \mathbb{N} \). It is in the form of formulas such as \( s(n) = n^2 \) or \( f(x) = x^3 - 3x + 7 \) that most people first encounter functions. But you should note that a formula by itself is not a function, although it might well specify a function between two given sets of numbers. Functions are much more general than formulas, and they apply to all kinds of sets, not just to sets of numbers.

Suppose that \( f: A \rightarrow B \) and \( g: B \rightarrow C \) are functions. Given \( a \in A \), there is an associated element \( f(a) \in B \). Since \( g \) is a function from \( B \) to \( C \), and since \( f(a) \in B \), \( g \) associates some element of \( C \) to \( f(a) \). That element is \( g(f(a)) \). Starting with an element \( a \) of \( A \), we have produced an associated element \( g(f(a)) \) of \( C \). This means that we have defined a new function from the set \( A \) to the set \( C \). This function is called the composition of \( g \) with \( f \), and it is denoted by \( g \circ f \). That is, if \( f: A \rightarrow B \) and \( g: B \rightarrow C \) are functions, then \( g \circ f: A \rightarrow C \) is the function which is defined by

\[
(g \circ f)(a) = g(f(a))
\]

for each \( a \in A \). For example, suppose that \( p \) is the function that associates to each item in a store the price of the item, and suppose that \( t \) is a function that associates the amount of tax on a price to each possible price. The composition, \( t \circ p \), is the function that associates to each item the amount of tax on that item. Or suppose that \( s: \mathbb{N} \rightarrow \mathbb{N} \) and \( r: \mathbb{N} \rightarrow \mathbb{N} \) are the functions defined by the formulas \( s(n) = n^2 \) and \( r(n) = 3n + 1 \), for each \( n \in \mathbb{N} \). Then \( r \circ s \) is a function from \( \mathbb{N} \) to \( \mathbb{N} \), and for \( n \in \mathbb{N} \), \( (r \circ s)(n) = r(s(n)) = r(n^2) = 3n^2 + 1 \). In this case, we also have the function \( s \circ r \), which satisfies \( (s \circ r)(n) = s(r(n)) = s(3n + 1) = (3n + 1)^2 = 9n^2 + 6n + 1 \). Note in particular that \( r \circ s \) and \( s \circ r \) are not the same function. The operation \( \circ \) is not commutative.

If \( A \) is a set and \( f: A \rightarrow A \), then \( f \circ f \), the composition of \( f \) with itself, is defined. For example, using the function \( s \) from the preceding example, \( s \circ s \) is the function from \( \mathbb{N} \) to \( \mathbb{N} \) given by the formula \( (s \circ s)(n) = s(s(n)) = s(n^2) = (n^2)^2 = n^4 \). If \( m \) is the function from the set of people to itself which associates to each person that person’s mother, then \( m \circ m \) is the function that associates to each person that person’s maternal grandmother.

If \( a \) and \( b \) are entities, then \((a, b)\) denotes the ordered pair containing \( a \) and \( b \). The ordered pair \((a, b)\) differs from the set \(\{a, b\}\) because a set is not
ordered. That is, \( \{a, b\} \) and \( \{b, a\} \) denote the same set, but if \( a \neq b \), then \( (a, b) \) and \( (b, a) \) are different ordered pairs. More generally, two ordered pairs \( (a, b) \) and \( (c, d) \) are equal if and only if both \( a = c \) and \( b = d \). If \( (a, b) \) is an ordered pair, then \( a \) and \( b \) are referred to as the coordinates of the ordered pair. In particular, \( a \) is the first coordinate and \( b \) is the second coordinate.

If \( A \) and \( B \) are sets, then we can form the set \( A \times B \) which is defined by

\[
A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}.
\]

This set is called the cross product or Cartesian product of the sets \( A \) and \( B \). The set \( A \times B \) contains every ordered pair whose first coordinate is an element of \( A \) and whose second coordinate is an element of \( B \). For example, if \( X = \{c, d\} \) and \( Y = \{1, 2, 3\} \), then \( X \times Y = \{(c, 1), (c, 2), (c, 3), (d, 1), (d, 2), (d, 3)\} \). It is possible to extend this idea to the cross product of more than two sets. The cross product of the three sets \( A, B, \) and \( C \) is denoted \( A \times B \times C \). It consists of all ordered triples \( (a, b, c) \) where \( a \in A \), \( b \in B \), and \( c \in C \). The definition for four or more sets is similar. The general term for a member of a cross product is tuple or, more specifically, ordered \( n \)-tuple. For example, \((a, b, c, d, e)\) is an ordered 5-tuple.

Given a function \( f: A \rightarrow B \), consider the set \( \{(a, b) \in A \times B \mid a \in A \text{ and } b = f(a)\} \). This set of ordered pairs consists of all pairs \( (a, b) \) such that \( a \in A \) and \( b \) is the element of \( B \) that is associated to \( a \) by the function \( f \). The set \( \{(a, b) \in A \times B \mid a \in A \text{ and } b = f(a)\} \) is called the graph of the function \( f \). Since \( f \) is a function, each element \( a \in A \) occurs once and only once as a first coordinate among the ordered pairs in the graph of \( f \). Given \( a \in A \), we can determine \( f(a) \) by finding that ordered pair and looking at the second coordinate. In fact, it is convenient to consider the function and its graph to be the same thing, and to use this as our official mathematical definition.\(^3\)

**Definition 2.2.** Let \( A \) and \( B \) be sets. A function from \( A \) to \( B \) is a subset of \( A \times B \) which has the property that for each \( a \in A \), the set contains one and only one ordered pair whose first coordinate is \( a \). If \( (a, b) \) is that ordered pair, then \( b \) is called the value of the function at \( a \) and is denoted \( f(a) \). If \( b = f(a) \), then we also say that the function \( f \) maps \( a \) to \( b \). The fact that \( f \) is a function from \( A \) to \( B \) is indicated by the notation \( f: A \rightarrow B \).

---

\(^3\)This is a convenient definition for the mathematical world, but as is often the case in mathematics, it leaves out an awful lot of the real world. Functional relationships in the real world are meaningful, but we model them in mathematics with meaningless sets of ordered pairs. We do this for the usual reason: to have something precise and rigorous enough that we can make logical deductions and prove things about it.
For example, if \( X = \{a, b\} \) and \( Y = \{1, 2, 3\} \), then the set \( \{(a, 2), (b, 1)\} \) is a function from \( X \) to \( Y \), and \( \{(1, a), (2, a), (3, b)\} \) is a function from \( Y \) to \( X \). On the other hand, \( \{(1, a), (2, b)\} \) is not a function from \( Y \) to \( X \), since it does not specify any value for 3. And \( \{(a, 1), (a, 2), (b, 3)\} \) is not a function from \( X \) to \( Y \) because it specifies two different values, 1 and 2, associated with the same element, \( a \), of \( X \).

Even though the technical definition of a function is a set of ordered pairs, it’s usually better to think of a function from \( A \) to \( B \) as something that associates some element of \( B \) to every element of \( A \). The set of ordered pairs is one way of expressing this association. If the association is expressed in some other way, it’s easy to write down the set of ordered pairs. For example, the function \( s : \mathbb{N} \rightarrow \mathbb{N} \) which is specified by the formula \( s(n) = n^2 \) can be written as the set of ordered pairs \( \{(n, n^2) \mid n \in \mathbb{N}\} \).

Suppose that \( f : A \rightarrow B \) is a function from the set \( A \) to the set \( B \). We say that \( A \) is the domain of the function \( f \) and that \( B \) is the range of the function. We define the image of the function \( f \) to be the set \( \{ b \in B \mid \exists a \in A (b = f(a)) \} \). Put more simply, the image of \( f \) is the set \( \{ f(a) \mid a \in A \} \). That is, the image is the set of all values, \( f(a) \), of the function, for all \( a \in A \). (You should note that in some cases—particularly in calculus courses—the term “range” is used to refer to what I am calling the image.) For example, for the function \( s : \mathbb{N} \rightarrow \mathbb{N} \) that is specified by \( s(n) = n^2 \), both the domain and the range are \( \mathbb{N} \), and the image is the set \( \{n^2 \mid n \in \mathbb{N}\} \), or \( \{0, 1, 4, 9, 16, \ldots \} \).

Note that the image of a function is a subset of its range. It can be a proper subset, as in the above example, but it is also possible for the image of a function to be equal to the range. In that case, the function is said to be onto. Sometimes, the fancier term surjective is used instead. Formally, a function \( f : A \rightarrow B \) is said to be onto (or surjective) if every element of \( B \) is equal to \( f(a) \) for some element of \( A \). In terms of logic, \( f \) is onto if and only if

\[
\forall b \in B \left( \exists a \in A (b = f(a)) \right).
\]

For example, let \( X = \{a, b\} \) and \( Y = \{1, 2, 3\} \), and consider the function from \( Y \) to \( X \) specified by the set of ordered pairs \( \{(1, a), (2, a), (3, b)\} \). This function is onto because its image, \( \{a, b\} \), is equal to the range, \( X \). However, the function from \( X \) to \( Y \) given by \( \{(a, 1), (b, 3)\} \) is not onto, because its image, \( \{1, 3\} \), is a proper subset of its range, \( Y \). As a further example, consider the function \( f \) from \( \mathbb{Z} \) to \( \mathbb{Z} \) given by \( f(n) = n - 52 \). To show that \( f \) is onto, we need to pick an arbitrary \( b \) in the range \( \mathbb{Z} \) and show that there is some number \( a \) in the domain \( \mathbb{Z} \) such that \( f(a) = b \). So let \( b \) be an arbitrary integer; we want to find an \( a \) such that \( a - 52 = b \). Clearly this
equation will be true when \( a = b + 52 \). So every element \( b \) is the image of the number \( a = b + 52 \), and \( f \) is therefore onto. Note that if \( f \) had been specified to have domain \( \mathbb{N} \), then \( f \) would not be onto, as for some \( b \in \mathbb{Z} \) the number \( a = b + 52 \) is not in the domain \( \mathbb{N} \) (for example, the integer \(-73\) is not in the image of \( f \), since \(-21\) is not in \( \mathbb{N} \)).

If \( f : A \to B \) and if \( a \in A \), then \( a \) is associated to only one element of \( B \). This is part of the definition of a function. However, no such restriction holds for elements of \( B \). If \( b \in B \), it is possible for \( b \) to be associated to zero, one, two, three, \ldots, or even to an infinite number of elements of \( A \). In the case where each element of the range is associated to at most one element of the domain, the function is said to be one-to-one. Sometimes, the term \textit{injective} is used instead. The function \( f \) is one-to-one (or injective) if for any two distinct elements \( x \) and \( y \) in the domain of \( f \), \( f(x) \) and \( f(y) \) are also distinct. In terms of logic, \( f : A \to B \) is one-to-one if and only if

\[
\forall x \in A \forall y \in A (x \neq y \to f(x) \neq f(y)).
\]

Since a proposition is equivalent to its contrapositive, we can write this condition equivalently as

\[
\forall x \in A \forall y \in A (f(x) = f(y) \to x = y).
\]

Sometimes, it is easier to work with the definition of one-to-one when it is expressed in this form. The function that associates every person to his or her mother is not one-to-one because it is possible for two different people to have the same mother. The function \( s : \mathbb{N} \to \mathbb{N} \) specified by \( s(n) = n^2 \) is one-to-one. However, we can define a function \( r : \mathbb{Z} \to \mathbb{Z} \) by the same formula: \( r(n) = n^2 \), for \( n \in \mathbb{Z} \). The function \( r \) is not one-to-one since two different integers can have the same square. For example, \( r(-2) = r(2) \).

A function that is both one-to-one and onto is said to be \textit{bijective}. The function that associates each point in a map of New York State to a point in the state itself is presumably bijective. For each point on the map, there is a corresponding point in the state, and \textit{vice versa}. If we specify the function \( f \) from the set \( \{1, 2, 3\} \) to the set \( \{a, b, c\} \) as the set of ordered pairs \( \{(1, b), (2, a), (3, c)\} \), then \( f \) is a bijective function. Or consider the function from \( \mathbb{Z} \) to \( \mathbb{Z} \) given by \( f(n) = n - 52 \). We have already shown that \( f \) is onto. We can show that it is also one-to-one: pick an arbitrary \( x \) and \( y \) in \( \mathbb{Z} \) and assume that \( f(x) = f(y) \). This means that \( x - 52 = y - 52 \), and adding 52 to both sides of the equation gives \( x = y \). Since \( x \) and \( y \) were arbitrary, we have proved \( \forall x \in \mathbb{Z} \forall y \in \mathbb{Z} (f(x) = f(y) \to x = y) \), that is, that \( f \) is one-to-one. Altogether, then, \( f \) is a bijection.

One difficulty that people sometimes have with mathematics is its generality. A set is a collection of entities, but an “entity” can be anything at
all, including other sets. Once we have defined ordered pairs, we can use ordered pairs as elements of sets. We could also make ordered pairs of sets. Now that we have defined functions, every function is itself a

equality. This means that we can have sets that contain functions. We can even have a function whose domain and range are sets of functions. Similarly, the domain or range of a function might be a set of sets, or a set of ordered pairs. Computer scientists have a good name for this. They would say that sets, ordered pairs, and functions are first-class objects. Once a set, ordered pair, or function has been defined, it can be used just like any other entity. If they were not first-class objects, there could be restrictions on the way they can be used. For example, it might not be possible to use functions as members of sets. (This would make them “second class.”)

For example, suppose that $A$, $B$, and $C$ are sets. Then since $A \times B$ is a set, we might have a function $f: A \times B \to C$. If $(a, b) \in A \times B$, then the value of $f$ at $(a, b)$ would be denoted $f((a, b))$. In practice, though, one set of parentheses is usually dropped, and the value of $f$ at $(a, b)$ is denoted $f(a, b)$. As a particular example, we might define a function $p: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with the formula $p(n, m) = nm + 1$. Similarly, we might define a function $q: \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ by $q(n, m, k) = (nm - k, nk - n)$.

Suppose that $A$ and $B$ are sets. There are, in general, many functions that map $A$ to $B$. We can gather all those functions into a set. This set, whose elements are all the functions from $A$ to $B$, is denoted $B^A$. (We’ll see later why this notation is reasonable.) Using this notation, saying $f: A \to B$ is exactly the same as saying $f \in B^A$. Both of these notations assert that $f$ is a function from $A$ to $B$. Of course, we can also form an unlimited number of other sets, such as the power set $\mathcal{P}(B^A)$, the cross product $B^A \times A$, or the set $A^A \times A$, which contains all the functions from the set $A \times A$ to the set $A$. And of course, any of these sets can be the domain or range of a function. An example of this is the function $E: B^A \times A \to B$ defined by the formula $E(f, a) = f(a)$. Let’s see if we can make sense of this notation. Since the domain of $E$ is $B^A \times A$, an element in the domain is an ordered pair in which the first coordinate is a function from $A$ to $B$ and the second coordinate is an element of $A$. Thus, $E(f, a)$ is defined for a function $f: A \to B$ and an element $a \in A$. Given such an $f$ and $a$, the notation $f(a)$ specifies an element of $B$, so the definition of $E(f, a)$ as $f(a)$ makes sense. The function $E$ is called the “evaluation function” since it captures the idea of evaluating a function at an element of its domain.

Exercises

1. Let $A = \{1, 2, 3, 4\}$ and let $B = \{a, b, c\}$. Find the sets $A \times B$ and $B \times A$. 
2. Let \( A \) be the set \( \{a, b, c, d\} \). Let \( f \) be the function from \( A \) to \( A \) given by the set of ordered pairs \( \{(a, b), (b, b), (c, a), (d, d)\} \), and let \( g \) be the function given by the set of ordered pairs \( \{(a, b), (b, c), (c, d), (d, d)\} \). Find the set of ordered pairs for the composition \( g \circ f \).

3. Let \( A = \{a, b, c\} \) and let \( B = \{0, 1\} \). Find all possible functions from \( A \) to \( B \). Give each function as a set of ordered pairs. (Hint: Every such function corresponds to one of the subsets of \( A \).)

4. Consider the functions from \( \mathbb{Z} \) to \( \mathbb{Z} \) which are defined by the following formulas. Decide whether each function is onto and whether it is one-to-one; prove your answers.
   
   a) \( f(n) = 2n \)   
   b) \( g(n) = n + 1 \)   
   c) \( h(n) = n^2 + n + 1 \)   
   d) \( s(n) = \begin{cases} 
   n/2, & \text{if } n \text{ is even} \\
   (n+1)/2, & \text{if } n \text{ is odd} 
   \end{cases} \)

5. Prove that composition of functions is an associative operation. That is, prove that for functions \( f: A \to B \), \( g: B \to C \), and \( h: C \to D \), the compositions \( (h \circ g) \circ f \) and \( h \circ (g \circ f) \) are equal.

6. Suppose that \( f: A \to B \) and \( g: B \to C \) are functions and that \( g \circ f \) is one-to-one.
   
   a) Prove that \( f \) is one-to-one. (Hint: use a proof by contradiction.)
   
   b) Find a specific example that shows that \( g \) is not necessarily one-to-one.

7. Suppose that \( f: A \to B \) and \( g: B \to C \), and suppose that the composition \( g \circ f \) is an onto function.
   
   a) Prove that \( g \) is an onto function.
   
   b) Find a specific example that shows that \( f \) is not necessarily onto.

2.5 Application: Programming with Functions

Functions are fundamental in computer programming, although not everything in programming that goes by the name of “function” is a function according to the mathematical definition.

In computer programming, a function is a routine that is given some data as input and that will calculate and return an answer based on that data. For example, in the C++ programming language, a function that calculates the square of an integer could be written

```cpp
int square(int n) {
    return n*n;
}
```

In C++, \( int \) is a data type. From the mathematical point of view, a data type is a set. The data type \( int \) is the set of all integers that can be represented as 32-bit binary numbers. Mathematically, then, \( int \subseteq \mathbb{Z} \).
(You should get used to the fact that sets and functions can have names that consist of more than one character, since it’s done all the time in computer programming.) The first line of the above function definition, “int square(int n)”, says that we are defining a function named square whose range is int and whose domain is int. In the usual notation for functions, we would express this as square: int → int, or possibly as square ∈ int^{int}, where int^{int} is the set of all functions that map the set int to the set int.

The first line of the function, int square(int n), is called the prototype of the function. The prototype specifies the name, the domain, and the range of the function and so carries exactly the same information as the notation “f: A → B”. The “n” in “int square(int n)” is a name for an arbitrary element of the data type int. In computer jargon, n is called a parameter of the function. The rest of the definition of square tells the computer to calculate the value of square(n) for any n ∈ int by multiplying n times n. The statement “return n*n” says that n * n is the value that is computed, or “returned,” by the function. (The * stands for multiplication.)

C++ has many data types in addition to int. There is a boolean data type named bool. The values of type bool are true and false. Mathematically, bool is a name for the set \{true, false\}. The type float consists of real numbers, which can include a decimal point. Of course, on a computer, it’s not possible to represent the entire infinite set of real numbers, so float represents some subset of the mathematical set of real numbers. There is also a data type whose values are strings of characters, such as “Hello world” or “xyz152QQZ”. The name for this data type in C++ is string. All these types, and many others, can be used in functions. For example, in C++, m % n is the remainder when the integer m is divided by the integer n. We can define a function to test whether an integer is even as follows:

```c++
bool even(int k) {
    if ( k % 2 == 1 )
        return false;
    else
        return true;
}
```

You don’t need to worry about all the details here, but you should understand that the prototype, bool even(int k), says that even is a function from the set int to the set bool. That is, even: int → bool. Given an integer N, even(N) has the value true if N is an even integer, and it has the value false if N is an odd integer.
A function can have more than one parameter. For example, we might define a function with prototype `int index(string str, string sub)`. If `s` and `t` are strings, then `index(s, t)` would be the `int` that is the value of the function at the ordered pair `(s, t)`. We see that the domain of `index` is the cross product `string × string`, and we can write `index: string × string → int` or, equivalently, `index ∈ int^{string×string}`.

Not every C++ function is actually a function in the mathematical sense. In mathematics, a function must associate a single value in its range to each value in its domain. There are two things that can go wrong: The value of the function might not be defined for every element of the domain, and the function might associate several different values to the same element of the domain. Both of these things can happen with C++ functions.

In computer programming, it is very common for a “function” to be undefined for some values of its parameter. In mathematics, a partial function from a set `A` to a set `B` is defined to be a function from a subset of `A` to `B`. A partial function from `A` to `B` can be undefined for some elements of `A`, but when it is defined for some `a ∈ A`, it associates just one element of `B` to `a`. Many functions in computer programs are actually partial functions. (When dealing with partial functions, an ordinary function, which is defined for every element of its domain, is sometimes referred to as a total function. Note that—with the mind-boggling logic that is typical of mathematicians—a total function is a type of partial function, because a set is a subset of itself.)

It’s also very common for a “function” in a computer program to produce a variety of values for the same value of its parameter. A common example is a function with prototype `int random(int N)`, which returns a random integer between 1 and `N`. The value of `random(5)` could be 1, 2, 3, 4, or 5. This is not the behavior of a mathematical function!

Even though many functions in computer programs are not really mathematical functions, I will continue to refer to them as functions in this section. Mathematicians will just have to stretch their definitions a bit to accommodate the realities of computer programming.

In most programming languages, functions are not first-class objects. That is, a function cannot be treated as a data value in the same way as a `string` or an `int`. However, C++ does take a step in this direction. It is possible for a function to be a parameter to another function. For example, consider the function prototype

```
float sumten( float f(int) )
```

This is a prototype for a function named `sumten` whose parameter is a function. The parameter is specified by the prototype “`float f(int)`”.

This means that the parameter must be a function from \( \text{int} \) to \( \text{float} \). The parameter name, \( f \), stands for an arbitrary such function. Mathematically, \( f \in \text{float}^{\text{int}} \), and so \( \text{sumten} : \text{float}^{\text{int}} \rightarrow \text{float} \).

My idea is that \( \text{sumten}(f) \) would compute \( f(1) + f(2) + \cdots + f(10) \). A more useful function would be able to compute \( f(a) + f(a+1) + \cdots + f(b) \) for any integers \( a \) and \( b \). This just means that \( a \) and \( b \) should be parameters to the function. The prototype for the improved function would look like

\[
\text{float sum( float f(int), int a, int b )}
\]

The parameters to \( \text{sum} \) form an ordered triple in which the first coordinate is a function and the second and third coordinates are integers. So, we could write

\[
\text{sum} : \text{float}^{\text{int}} \times \text{int} \times \text{int} \rightarrow \text{float}
\]

It’s interesting that computer programmers deal routinely with such complex objects.

One thing you can’t do in C++ is write a function that creates new functions from scratch. The only functions that exist are those that are coded into the source code of the program. There are programming languages that do allow new functions to be created from scratch while a program is running. In such languages, functions are first-class objects. These languages support what is called \textit{functional programming}.

One of the most accessible languages that supports functional programming is JavaScript, a language that is used on Web pages. (Although the names are similar, JavaScript and Java are only distantly related.) In JavaScript, the function that computes the square of its parameter could be defined as

\[
\text{function square(n) { return n*n; }}
\]

This is similar to the C++ definition of the same function, but you’ll notice that no type is specified for the parameter \( n \) or for the value computed by the function. Given this definition of square, \( \text{square}(x) \) would be legal for any \( x \) of any type. (Of course, the value of \( \text{square}(x) \) would be undefined for most types, so \( \text{square} \) is a \textit{very} partial function, like most functions in JavaScript.) In effect, all possible data values in JavaScript are bundled together into one set, which I will call \( \text{data} \). We then have \( \text{square} : \text{data} \rightarrow \text{data} \).\footnote{Not all functional programming languages lump data types together in this way. There is a functional programming language named Haskell, for example, that is as strict about types as C++. For information about Haskell, see http://www.Haskell.org/\( .\)}
In JavaScript, a function really is a first-class object. We can begin to see this by looking at an alternative definition of the function *square*:

```javascript
square = function(n) { return n*n; }
```

Here, the notation “`function(n) { return n*n; }`” creates a function that computes the square of its parameter, but it doesn’t give any name to this function. This function object is then assigned to a variable named *square*. The value of *square* can be changed later, with another assignment statement, to a different function or even to a different type of value. This notation for creating function objects can be used in other places besides assignment statements. Suppose, for example, that a function with prototype

```javascript
function sum(f,a,b)
```

has been defined in a JavaScript program to compute \( f(a) + f(a+1) + \cdots + f(b) \). Then we could compute \( 1^2 + 2^2 + \cdots + 100^2 \) by saying

```javascript
sum( function(n) { return n*n; }, 1, 100 )
```

Here, the first parameter is the function that computes squares. We have created and used this function without ever giving it a name.

It is even possible in JavaScript for a function to return another function as its value. For example,

```javascript
function monomial(a, n) {
    return ( function(x) { a*Math.pow(x,n); } );
}
```

Here, `Math.pow(x,n)` computes \( x^n \), so for any numbers \( a \) and \( n \), the value of `monomial(a,n)` is a function that computes \( ax^n \). Thus,

```javascript
f = monomial(2,3);
```

would define \( f \) to be the function that satisfies \( f(x) = 2x^3 \), and if *sum* is the function described above, then

```javascript
sum( monomial(8,4), 3, 6 )
```

would compute \( 8 \times 3^4 + 8 \times 4^4 + 8 \times 5^4 + 8 \times 6^4 \). In fact, `monomial` can be used to create an unlimited number of new functions from scratch. It is even possible to write `monomial(2,3)(5)` to indicate the result of applying the function `monomial(2,3)` to the value 5. The value represented by `monomial(2,3)(5)` is \( 2 \times 5^3 \), or 250. This is real functional programming and might give you some idea of its power.
Exercises

1. For each of the following C++ function prototypes, translate the prototype into a standard mathematical function specification, such as \( \text{func: float} \rightarrow \text{int} \).
   
   a) int strlen(string s)
   
   b) float pythag(float x, float y)
   
   c) int round(float x)
   
   d) string sub(string s, int n, int m)
   
   e) string unlikely( int f(string) )
   
   f) int h( int f(int), int g(int) )

2. Write a C++ function prototype for a function that belongs to each of the following sets.
   
   a) string
   
   b) bool
   
   c) float

3. It is possible to define new types in C++. For example, the definition

   ```
   struct point {
      float x;
      float y;
   }
   ```

   defines a new type named `point`. A value of type `point` contains two values of type `float`. What mathematical operation corresponds to the construction of this data type? Why?

4. Let `square`, `sum` and `monomial` be the JavaScript functions described in this section. What is the value of each of the following?
   
   a) `sum(square, 2, 4)`
   
   b) `sum(monomial(5,2), 1, 3)`
   
   c) `monomial(square(2), 7)`
   
   d) `sum(function(n) { return 2 * n; }, 1, 5)`
   
   e) `square(sum(monomial(2,3), 1, 2))`

5. Write a JavaScript function named `compose` that computes the composition of two functions. That is, \( \text{compose}(f,g) \) is \( f \circ g \), where \( f \) and \( g \) are functions of one parameter. Recall that \( f \circ g \) is the function defined by \( (f \circ g)(x) = f(g(x)) \).

2.6 Counting Past Infinity

As children, we all learned to answer the question “How many?” by counting with numbers: 1, 2, 3, 4, …. But the question of “How many?” was asked and answered long before the abstract concept of number was invented. The answer can be given in terms of “as many as.” How many
cousins do you have? As many cousins as I have fingers on both hands. How many sheep do you own? As many sheep as there are notches on this stick. How many baskets of wheat must I pay in taxes? As many baskets as there are stones in this box. The question of how many things are in one collection of objects is answered by exhibiting another, more convenient, collection of objects that has just as many members.

In set theory, the idea of one set having just as many members as another set is expressed in terms of one-to-one correspondence. A one-to-one correspondence between two sets $A$ and $B$ pairs each element of $A$ with an element of $B$ in such a way that every element of $B$ is paired with one and only one element of $A$. The process of counting, as it is learned by children, establishes a one-to-one correspondence between a set of $n$ objects and the set of numbers from 1 to $n$. The rules of counting are the rules of one-to-one correspondence: Make sure you count every object, make sure you don’t count the same object more than once. That is, make sure that each object corresponds to one and only one number. Earlier in this chapter, we used the fancy name “bijective function” to refer to this idea, but we can now see it as an old, intuitive way of answering the question “How many?”

In counting, as it is learned in childhood, the set $\{1, 2, 3, \ldots, n\}$ is used as a typical set that contains $n$ elements. In mathematics and computer science, it has become more common to start counting with zero instead of with one, so we define the following sets to use as our basis for counting:

- $N_0 = \emptyset$, a set with 0 elements
- $N_1 = \{0\}$, a set with 1 element
- $N_2 = \{0, 1\}$, a set with 2 elements
- $N_3 = \{0, 1, 2\}$, a set with 3 elements
- $N_4 = \{0, 1, 2, 3\}$, a set with 4 elements

and so on. In general, $N_n = \{0, 1, 2, \ldots, n - 1\}$ for each $n \in \mathbb{N}$. For each natural number $n$, $N_n$ is a set with $n$ elements. Note that if $n \neq m$, then there is no one-to-one correspondence between $N_n$ and $N_m$. This is obvious, but like many obvious things is not all that easy to prove rigorously, and we omit the argument here.

**Theorem 2.6.** For each $n \in \mathbb{N}$, let $N_n$ be the set $N_n = \{0, 1, \ldots, n - 1\}$. If $n \neq m$, then there is no bijective function from $N_m$ to $N_n$.

We can now make the following definitions:

**Definition 2.3.** A set $A$ is said to be finite if there is a one-to-one correspondence between $A$ and $N_n$ for some natural number $n$. We then say that $n$ is the cardinality of $A$. The notation $|A|$ is used to indicate the
cardinality of $A$. That is, if $A$ is a finite set, then $|A|$ is the natural number $n$ such that there is a one-to-one correspondence between $A$ and $\mathbb{N}_n$. A set that is not finite is said to be **infinite**. That is, a set $B$ is infinite if for every $n \in \mathbb{N}$, there is no one-to-one correspondence between $B$ and $\mathbb{N}_n$.

Fortunately, we don’t always have to count every element in a set individually to determine its cardinality. Consider, for example, the set $A \times B$, where $A$ and $B$ are finite sets. If we already know $|A|$ and $|B|$, then we can determine $|A \times B|$ by computation, without explicit counting of elements. In fact, $|A \times B| = |A| \cdot |B|$. The cardinality of the cross product $A \times B$ can be computed by multiplying the cardinality of $A$ by the cardinality of $B$. To see why this is true, think of how you might count the elements of $A \times B$. You could put the elements into piles, where all the ordered pairs in a pile have the same first coordinate. There are as many piles as there are elements of $A$, and each pile contains as many ordered pairs as there are elements of $B$. That is, there are $|A|$ piles, with $|B|$ items in each. By the definition of multiplication, the total number of items in all the piles is $|A| \cdot |B|$. A similar result holds for the cross product of more than two finite sets. For example, $|A \times B \times C| = |A| \cdot |B| \cdot |C|$.

It’s also easy to compute $|A \cup B|$ in the case where $A$ and $B$ are disjoint finite sets. (Recall that two sets $A$ and $B$ are said to be disjoint if they have no members in common, that is, if $A \cap B = \emptyset$.) Suppose $|A| = n$ and $|B| = m$. If we wanted to count the elements of $A \cup B$, we could use the $n$ numbers from 0 to $n - 1$ to count the elements of $A$ and then use the $m$ numbers from $n$ to $n + m - 1$ to count the elements of $B$. This amounts to a one-to-one correspondence between $A \cup B$ and the set $\mathbb{N}_{n+m}$. We see that $|A \cup B| = n + m$. That is, for disjoint finite sets $A$ and $B$, $|A \cup B| = |A| + |B|$.

What about $A \cup B$, where $A$ and $B$ are not disjoint? We have to be careful not to count the elements of $A \cap B$ twice. After counting the elements of $A$, there are only $|B| - |A \cap B|$ new elements in $B$ that still need to be counted. So we see that for any two finite sets $A$ and $B$, $|A \cup B| = |A| + |B| - |A \cap B|$.

What about the number of subsets of a finite set $A$? What is the relationship between $|A|$ and $|\mathcal{P}(A)|$? The answer is provided by the following theorem.

**Theorem 2.7.** A finite set with cardinality $n$ has $2^n$ subsets.

**Proof.** Let $P(n)$ be the statement “Any set with cardinality $n$ has $2^n$ subsets.” We will use induction to show that $P(n)$ is true for all $n \in \mathbb{N}$.

Base case: For $n = 0$, $P(n)$ is the statement that a set with cardinality
0 has $2^0$ subsets. The only set with 0 elements is the empty set. The empty set has exactly 1 subset, namely itself. Since $2^0 = 1$, $P(0)$ is true.

Inductive case: Let $k$ be an arbitrary element of $\mathbb{N}$, and assume that $P(k)$ is true. That is, assume that any set with cardinality $k$ has $2^k$ elements. (This is the induction hypothesis.) We must show that $P(k + 1)$ follows from this assumption. That is, using the assumption that any set with cardinality $k$ has $2^k$ subsets, we must show that any set with cardinality $k + 1$ has $2^{k+1}$ subsets.

Let $A$ be an arbitrary set with cardinality $k + 1$. We must show that $|\mathcal{P}(A)| = 2^{k+1}$. Since $|A| > 0$, $A$ contains at least one element. Let $x$ be some element of $A$, and let $B = A \setminus \{x\}$. The cardinality of $B$ is $k$, so we have by the induction hypothesis that $|\mathcal{P}(B)| = 2^k$. Now, we can divide the subsets of $A$ into two classes: subsets of $A$ that do not contain $x$ and subsets of $A$ that do contain $x$. Let $Y$ be the collection of subsets of $A$ that do not contain $x$, and let $X$ be the collection of subsets of $A$ that do contain $x$. $X$ and $Y$ are disjoint, since it is impossible for a given subset of $A$ both to contain and to not contain $x$. It follows that $|\mathcal{P}(A)| = |X \cup Y| = |X| + |Y|$.  

Now, a member of $Y$ is a subset of $A$ that does not contain $x$. But that is exactly the same as saying that a member of $Y$ is a subset of $B$. So $Y = \mathcal{P}(B)$, which we know contains $2^k$ members. As for $X$, there is a one-to-one correspondence between $\mathcal{P}(B)$ and $X$. Namely, the function $f: \mathcal{P}(B) \rightarrow X$ defined by $f(C) = C \cup \{x\}$ is a bijective function. (The proof of this is left as an exercise.) From this, it follows that $|X| = |\mathcal{P}(B)| = 2^k$. Putting these facts together, we see that $|\mathcal{P}(A)| = |X| + |Y| = 2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$. This completes the proof that $P(k) \rightarrow P(k + 1)$.

We have seen that the notation $A^B$ represents the set of all functions from $B$ to $A$. Suppose $A$ and $B$ are finite, and that $|A| = n$ and $|B| = m$. Then $|A^B| = n^m = |A|^{|B|}$. (This fact is one of the reasons why the notation $A^B$ is reasonable.) One way to see this is to note that there is a one-to-one correspondence between $A^B$ and a cross product $A \times A \times \cdots A$, where the number of terms in the cross product is $m$. (This will be shown in one of the exercises at the end of this section.) It follows that $|A^B| = |A| \cdot |A| \cdots |A| = n \cdot n \cdots n$, where the factor $n$ occurs $m$ times in the product. This product is, by definition, $n^m$.

This discussion about computing cardinalities is summarized in the following theorem:

**Theorem 2.8.** Let $A$ and $B$ be finite sets. Then

- $|A \times B| = |A| \cdot |B|$.
- $|A \cup B| = |A| + |B| - |A \cap B|$.
118  

CHAPTER 2. SETS, FUNCTIONS, AND RELATIONS

• If $A$ and $B$ are disjoint then $|A \cup B| = |A| + |B|$.

• $|A^B| = |A|^{|B|}$.

• $|\mathcal{P}(A)| = 2^{|A|}$.

When it comes to counting and computing cardinalities, this theorem is only the beginning of the story. There is an entire large and deep branch of mathematics known as **combinatorics** that is devoted mostly to the problem of counting. But the theorem is already enough to answer many questions about cardinalities.

For example, suppose that $|A| = n$ and $|B| = m$. We can form the set $\mathcal{P}(A \times B)$, which consists of all subsets of $A \times B$. Using the theorem, we can compute that $|\mathcal{P}(A \times B)| = 2^{|A \times B|} = 2^{|A| \cdot |B|} = 2^{nm}$. If we assume that $A$ and $B$ are disjoint, then we can compute that $|A^{A \cup B}| = |A|^{|A \cup B|} = n^{n+m}$.

To be more concrete, let $X = \{a, b, c, d, e\}$ and let $Y = \{c, d, e, f\}$ where $a$, $b$, $c$, $d$, $e$, and $f$ are distinct. Then $|X \times Y| = 5 \cdot 4 = 20$ while $|X \cup Y| = 5 + 4 - |\{c, d, e\}| = 6$ and $|X^Y| = 5^4 = 625$.

We can also answer some simple practical questions. Suppose that in a restaurant you can choose one appetizer and one main course. What is the number of possible meals? If $A$ is the set of possible appetizers and $C$ is the set of possible main courses, then your meal is an ordered pair belonging to the set $A \times C$. The number of possible meals is $|A \times C|$, which is the product of the number of appetizers and the number of main courses.

Or suppose that four different prizes are to be awarded, and that the set of people who are eligible for the prizes is $A$. Suppose that $|A| = n$. How many different ways are there to award the prizes? One way to answer this question is to view a way of awarding the prizes as a function from the set of prizes to the set of people. Then, if $P$ is the set of prizes, the number of different ways of awarding the prizes is $|A^P|$. Since $|P| = 4$ and $|A| = n$, this is $n^4$. Another way to look at it is to note that the people who win the prizes form an ordered tuple $(a, b, c, d)$, which is an element of $A \times A \times A \times A$. So the number of different ways of awarding the prizes is $|A \times A \times A \times A|$, which is $|A| \cdot |A| \cdot |A| \cdot |A|$. This is $|A|^4$, or $n^4$, the same answer we got before.\(^5\)

\(^5\)This discussion assumes that one person can receive any number of prizes. What if the prizes have to go to four different people? This question takes us a little farther into combinatorics than I would like to go, but the answer is not hard. The first award can be given to any of $n$ people. The second prize goes to one of the remaining $n - 1$ people. There are $n - 2$ choices for the third prize and $n - 3$ for the fourth. The number of different ways of awarding the prizes to four different people is the product $n(n-1)(n-2)(n-3)$. 

So far, we have only discussed finite sets. \( \mathbb{N} \), the set of natural numbers \( \{0, 1, 2, 3, \ldots \} \), is an example of an infinite set. There is no one-to-one correspondence between \( \mathbb{N} \) and any of the finite sets \( \mathbb{N}_n \). Another example of an infinite set is the set of even natural numbers, \( E = \{0, 2, 4, 6, 8, \ldots \} \). There is a natural sense in which the sets \( \mathbb{N} \) and \( E \) have the same number of elements. That is, there is a one-to-one correspondence between them. The function \( f: \mathbb{N} \rightarrow E \) defined by \( f(n) = 2n \) is bijective. We will say that \( \mathbb{N} \) and \( E \) have the same cardinality, even though that cardinality is not a finite number. Note that \( E \) is a proper subset of \( \mathbb{N} \). That is, \( \mathbb{N} \) has a proper subset that has the same cardinality as \( \mathbb{N} \).

We will see that not all infinite sets have the same cardinality. When it comes to infinite sets, intuition is not always a good guide. Most people seem to be torn between two conflicting ideas. On the one hand, they think, it seems that a proper subset of a set should have fewer elements than the set itself. On the other hand, it seems that any two infinite sets should have the same number of elements. Neither of these is true, at least if we define having the same number of elements in terms of one-to-one correspondence.

A set \( A \) is said to be **countably infinite** if there is a one-to-one correspondence between \( \mathbb{N} \) and \( A \). A set is said to be **countable** if it is either finite or countably infinite. An infinite set that is not countably infinite is said to be **uncountable**. If \( X \) is an uncountable set, then there is no one-to-one correspondence between \( \mathbb{N} \) and \( X \).

The idea of “countable infinity” is that even though a countably infinite set cannot be counted in a finite time, we can imagine counting all the elements of \( A \), one-by-one, in an infinite process. A bijective function \( f: \mathbb{N} \rightarrow A \) provides such an infinite listing: \((f(0), f(1), f(2), f(3), \ldots)\). Since \( f \) is onto, this infinite list includes all the elements of \( A \). In fact, making such a list effectively shows that \( A \) is countably infinite, since the list amounts to a bijective function from \( \mathbb{N} \) to \( A \). For an uncountable set, it is impossible to make a list, even an infinite list, that contains all the elements of the set.

Before you start believing in uncountable sets, you should ask for an example. In Chapter 1, we worked with the infinite sets \( \mathbb{Z} \) (the integers), \( \mathbb{Q} \) (the rationals), \( \mathbb{R} \) (the reals), and \( \mathbb{R} \setminus \mathbb{Q} \) (the irrationals). Intuitively, these are all “bigger” than \( \mathbb{N} \), but as we have already mentioned, intuition is a poor guide when it comes to infinite sets. Are any of \( \mathbb{Z} \), \( \mathbb{Q} \), \( \mathbb{R} \), and \( \mathbb{R} \setminus \mathbb{Q} \) in fact uncountable?

It turns out that both \( \mathbb{Z} \) and \( \mathbb{Q} \) are only countably infinite. The proof that \( \mathbb{Z} \) is countable is left as an exercise; we will show here that the set of non-negative rational numbers is countable. (The fact that \( \mathbb{Q} \) itself is countable follows easily from this.) The reason is that it’s possible to make
an infinite list containing all the non-negative rational numbers. Start the
list with all the non-negative rational numbers \( n/m \) such that \( n + m = 1 \).
There is only one such number, namely 0/1. Next come numbers with
\( n + m = 2 \). They are 0/2 and 1/1, but we leave out 0/2 since it’s just
another way of writing 0/1, which is already in the list. Now, we add
the numbers with \( n + m = 3 \), namely 0/3, 1/2, and 2/1. Again, we leave out
0/3, since it’s equal to a number already in the list. Next come numbers
with \( n + m = 4 \). Leaving out 0/4 and 2/2 since they are already in the list,
we add 1/3 and 3/1 to the list. We continue in this way, adding numbers
with \( n + m = 5 \), then numbers with \( n + m = 6 \), and so on. The list looks like
\[
\left( \frac{0}{1}, \frac{1}{1}, \frac{2}{1}, \frac{1}{3}, \frac{3}{1}, \frac{1}{4}, \frac{4}{1}, \frac{1}{5}, \frac{5}{1}, \frac{1}{6}, \frac{6}{1}, \ldots \right)
\]
This process can be continued indefinitely, and every non-negative rational
number will eventually show up in the list. So we get a complete, infinite list
of non-negative rational numbers. This shows that the set of non-negative
rational numbers is in fact countable.

On the other hand, \( \mathbb{R} \) is uncountable. It is not possible to make an
infinite list that contains every real number. It is not even possible to make
a list that contains every real number between zero and one. Another way
of saying this is that every infinite list of real numbers between zero and one,
no matter how it is constructed, leaves something out. To see why this
is true, imagine such a list, displayed in an infinitely long column. Each
row contains one number, which has an infinite number of digits after the
decimal point. Since it is a number between zero and one, the only digit
before the decimal point is zero. For example, the list might look like this:

\begin{verbatim}
0.903989372498795612972654857945...
0.1234934204059875980239230834549...
0.22400043298436234709323279989579...
0.5000000000000000000000000000...
0.77743449234234876990120909480009...
0.777555558888888949888898000111...
0.123456788888888888888888000000...
0.34835440009848712712123940320577...
0.93473244447900498340999990948900...
\end{verbatim}

This is only (a small part of) one possible list. How can we be certain that
every such list leaves out some real number between zero and one? The
trick is to look at the digits shown in bold face. We can use these digits to
2.6. COUNTING PAST INFINITY

build a number that is not in the list. Since the first number in the list has a 9 in the first position after the decimal point, we know that this number cannot equal any number of, for example, the form 0.4\ldots. Since the second number has a 2 in the second position after the decimal point, neither of the first two numbers in the list is equal to any number that begins with 0.44\ldots. Since the third number has a 4 in the third position after the decimal point, none of the first three numbers in the list is equal to any number that begins 0.445\ldots. We can continue to construct a number in this way, and we end up with a number that is different from every number in the list. The $n^{th}$ digit of the number we are building must differ from the $n^{th}$ digit of the $n^{th}$ number in the list. These are the digits shown in bold face in the above list. To be definite, I use a 5 when the corresponding boldface number is 4, and otherwise I use a 4. For the list shown above, this gives a number that begins 0.44544445\ldots. The number constructed in this way is not in the given list, so the list is incomplete. The same construction clearly works for any list of real numbers between zero and one. No such list can be a complete listing of the real numbers between zero and one, and so there can be no complete listing of all real numbers. We conclude that the set $\mathbb{R}$ is uncountable.

The technique used in this argument is called diagonalization. It is named after the fact that the bold face digits in the above list lie along a diagonal line. This proof was discovered by a mathematician named Georg Cantor, who caused quite a fuss in the nineteenth century when he came up with the idea that there are different kinds of infinity. Since then, his notion of using one-to-one correspondence to define the cardinalities of infinite sets has been accepted. Mathematicians now consider it almost intuitive that $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{Q}$ have the same cardinality while $\mathbb{R}$ has a strictly larger cardinality.

**Theorem 2.9.** Suppose that $X$ is an uncountable set, and that $K$ is a countable subset of $X$. Then the set $X \setminus K$ is uncountable.

**Proof.** Let $X$ be an uncountable set. Let $K \subseteq X$, and suppose that $K$ is countable. Let $L = X \setminus K$. We want to show that $L$ is uncountable. Suppose that $L$ is countable. We will show that this assumption leads to a contradiction.

Note that $X = K \cup (X \setminus K) = K \cup L$. You will show in Exercise 11 of this section that the union of two countable sets is countable. Since $X$ is the union of the countable sets $K$ and $L$, it follows that $X$ is countable. But this contradicts the fact that $X$ is uncountable. This contradiction proves the theorem. 

In the proof, both $q$ and $\neg q$ are shown to follow from the assumptions,
where \( q \) is the statement “\( X \) is countable.” The statement \( q \) is shown to follow from the assumption that \( X \setminus K \) is countable. The statement \( \neg q \) is true by assumption. Since \( q \) and \( \neg q \) cannot both be true, at least one of the assumptions must be false. The only assumption that can be false is the assumption that \( X \setminus K \) is countable.

This theorem, by the way, has the following easy corollary. (A corollary is a theorem that follows easily from another, previously proved theorem.)

**Corollary 2.10.** The set of irrational real numbers is uncountable.

**Proof.** Let \( I \) be the set of irrational real numbers. By definition, \( I = \mathbb{R} \setminus \mathbb{Q} \). We have already shown that \( \mathbb{R} \) is uncountable and that \( \mathbb{Q} \) is countable, so the result follows immediately from the previous theorem. \( \square \)

You might still think that \( \mathbb{R} \) is as big as things get, that is, that any infinite set is in one-to-one correspondence with \( \mathbb{R} \) or with some subset of \( \mathbb{R} \). In fact, though, if \( X \) is any set then it’s possible to find a set that has strictly larger cardinality than \( X \). In fact, \( \mathcal{P}(X) \) is such a set. A variation of the diagonalization technique can be used to show that there is no one-to-one correspondence between \( X \) and \( \mathcal{P}(X) \). Note that this is obvious for finite sets, since for a finite set \( X \), \( |\mathcal{P}(X)| = 2^{|X|} \), which is larger than \( |X| \). The point of the theorem is that it is true even for infinite sets.

**Theorem 2.11.** Let \( X \) be any set. Then there is no one-to-one correspondence between \( X \) and \( \mathcal{P}(X) \).

**Proof.** Given an arbitrary function \( f : X \rightarrow \mathcal{P}(X) \), we can show that \( f \) is not onto. Since a one-to-one correspondence is both one-to-one and onto, this shows that \( f \) is not a one-to-one correspondence.

Recall that \( \mathcal{P}(X) \) is the set of subsets of \( X \). So, for each \( x \in X \), \( f(x) \) is a subset of \( X \). We have to show that no matter how \( f \) is defined, there is some subset of \( X \) that is not in the image of \( f \).

Given \( f \), we define \( A \) to be the set \( A = \{ x \in X \mid x \notin f(x) \} \). The test “\( x \notin f(x) \)” makes sense because \( f(x) \) is a set. Since \( A \subseteq X \), we have that \( A \in \mathcal{P}(X) \). However, \( A \) is not in the image of \( f \). That is, for every \( y \in X \), \( A \neq f(y) \).\(^6\) To see why this is true, let \( y \) be any element of \( X \). There are two cases to consider. Either \( y \in f(y) \) or \( y \notin f(y) \). We show that whichever case holds, \( A \neq f(y) \). If it is true that \( y \in f(y) \), then by the definition of \( A \), \( y \notin A \). Since \( y \in f(y) \) but \( y \notin A \), \( f(y) \) and \( A \) do not have the same elements and therefore are not equal. On the other hand, suppose that \( y \notin f(y) \).

\(^6\)In fact, we have constructed \( A \) so that the sets \( A \) and \( f(y) \) differ in at least one element, namely \( y \) itself. This is where the “diagonalization” comes in.
Again, by the definition of $A$, this implies that $y \in A$. Since $y \notin f(y)$ but $y \in A$, $f(y)$ and $A$ do not have the same elements and therefore are not equal. In either case, $A \neq f(y)$. Since this is true for any $y \in X$, we conclude that $A$ is not in the image of $f$ and therefore $f$ is not a one-to-one correspondence.

From this theorem, it follows that there is no one-to-one correspondence between $\mathbb{R}$ and $\mathcal{P}(\mathbb{R})$. The cardinality of $\mathcal{P}(\mathbb{R})$ is strictly bigger than the cardinality of $\mathbb{R}$. But it doesn’t stop there. $\mathcal{P}(\mathcal{P}(\mathbb{R}))$ has an even bigger cardinality, and the cardinality of $\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{R})))$ is bigger still. We could go on like this forever, and we still won’t have exhausted all the possible cardinalities. If we let $X$ be the infinite union $\mathbb{R} \cup \mathcal{P}(\mathbb{R}) \cup \mathcal{P}(\mathcal{P}(\mathbb{R})) \cup \cdots$, then $X$ has larger cardinality than any of the sets in the union. And then there’s $\mathcal{P}(X)$, $\mathcal{P}(\mathcal{P}(X))$, $X \cup \mathcal{P}(X) \cup \mathcal{P}(\mathcal{P}(X)) \cup \cdots$. There is no end to this. There is no upper limit on possible cardinalities, not even an infinite one! We have counted past infinity.

We have seen that $|\mathbb{R}|$ is strictly larger than $|\mathbb{N}|$. We end this section with what might look like a simple question: Is there a subset of $\mathbb{R}$ that is neither in one-to-one correspondence with $\mathbb{N}$ nor with $\mathbb{R}$? That is, is the cardinality of $\mathbb{R}$ the next largest cardinality after the cardinality of $\mathbb{N}$, or are there other cardinalities intermediate between them? This problem was unsolved for quite a while, and the solution, when it was found, proved to be completely unexpected. It was shown that both “yes” and “no” are consistent answers to this question! That is, the logical structure built on the system of axioms that had been accepted as the basis of set theory was not extensive enough to answer the question. It is possible to extend the system in various ways. In some extensions, the answer is yes. In others, the answer is no. You might object, “Yes, but which answer is true for the real real numbers?” Unfortunately, it’s not even clear whether this question makes sense, since in the world of mathematics, the real numbers are just part of a structure built from a system of axioms. And it’s not at all clear whether the “real numbers” exist in some sense in the real world. If all this sounds like it’s a bit of a philosophical muddle, it is. That’s the state of things today at the foundation of mathematics.

Exercises

1. Suppose that $A$, $B$, and $C$ are finite sets which are pairwise disjoint. (That is, $A \cap B = A \cap C = B \cap C = \emptyset$.) Express the cardinality of each of the following sets in terms of $|A|$, $|B|$, and $|C|$. Which of your answers depend on the fact that the sets are pairwise disjoint?
   a) $\mathcal{P}(A \cup B)$
   b) $A \times (B^C)$
   c) $\mathcal{P}(A) \times \mathcal{P}(C)$
2. Suppose that $A$ and $B$ are finite sets which are not necessarily disjoint. What are all the possible values for $|A \cup B|$?

3. Let's say that an “identifier” consists of one or two characters. The fist character is one of the twenty-six letters (A, B, . . . , C). The second character, if there is one, is either a letter or one of the ten digits (0, 1, . . . , 9). How many different identifiers are there? Explain your answer in terms of unions and cross products.

4. Suppose that there are five books that you might bring along to read on your vacation. In how many different ways can you decide which book to bring, assuming that you want to bring at least one? Why?

5. Show that the cardinality of a finite set is well-defined. That is, show that if $f$ is a bijective function from a set $A$ to $\mathbb{N}_n$, and if $g$ is a bijective function from $A$ to $\mathbb{N}_m$, then $n = m$.

6. Finish the proof of Theorem 2.7 by proving the following statement: Let $A$ be a non-empty set, and let $x \in A$. Let $B = A \setminus \{x\}$. Let $X = \{C \subseteq A \mid x \in C\}$. Define $f: \mathcal{P}(B) \to X$ by the formula $f(C) = C \cup \{x\}$. Show that $f$ is a bijective function.

7. Use induction on the cardinality of $B$ to show that for any finite sets $A$ and $B$, $|A^B| = |A|^{|B|}$. (Hint: For the case where $B \neq \emptyset$, choose $x \in B$, and divide $A^B$ into classes according to the value of $f(x)$.)

8. Let $A$ and $B$ be finite sets with $|A| = n$ and $|B| = m$. List the elements of $B$ as $B = \{b_0, b_1, \ldots, b_{m-1}\}$. Define the function $\mathcal{F}: A^B \to A \times A \times \cdots \times A$, where $A$ occurs $m$ times in the cross product, by $\mathcal{F}(f) = (f(b_0), f(b_1), \ldots, f(b_{m-1}))$. Show that $\mathcal{F}$ is a one-to-one correspondence.

9. Show that $\mathbb{Z}$, the set of integers, is countable by finding a one-to-one correspondence between $\mathbb{N}$ and $\mathbb{Z}$.

10. Show that the set $\mathbb{N} \times \mathbb{N}$ is countable.

11. Complete the proof of Theorem 2.9 as follows:
   a) Suppose that $A$ and $B$ are countably infinite sets. Show that $A \cup B$ is countably infinite.
   b) Suppose that $A$ and $B$ are countable sets. Show that $A \cup B$ is countable.

12. Prove that each of the following statements is true. In each case, use a proof by contradiction.
   a) Let $X$ be a countably infinite set, and let $N$ be a finite subset of $X$. Then $X \setminus N$ is countably infinite.
   b) Let $A$ be an infinite set, and let $X$ be a subset of $A$. Then at least one of the sets $X$ and $A \setminus X$ is infinite.
   c) Every subset of a finite set is finite.

13. Let $A$ and $B$ be sets and let $\bot$ be an entity that is not a member of $B$. Show that there is a one-to-one correspondence between the set of functions from
2.7 Relations

In Section 2.4, we saw that “mother of” is a functional relationship because every person has one and only one mother, but that “child of” is not a functional relationship, because a person can have no children or more than one child. However, the relationship expressed by “child of” is certainly one that we have a right to be interested in and one that we should be able to deal with mathematically.

There are many examples of relationships that are not functional relationships. The relationship that holds between two natural numbers \( n \) and \( m \) when \( n \leq m \) is an example in mathematics. The relationship between a person and a book that that person has on loan from the library is another. Some relationships involve more than two entities, such as the relationship that associates a name, an address, and a phone number in an address book or the relationship that holds among three real numbers \( x, y, \) and \( z \) if \( x^2 + y^2 + z^2 = 1 \). Each of these relationships can be represented mathematically by what is called a “relation.”

A relation on two sets, \( A \) and \( B \), is defined to be a subset of \( A \times B \). Since a function from \( A \) to \( B \) is defined, formally, as a subset of \( A \times B \) that satisfies certain properties, a function is a relation. However, relations are more general than functions, since any subset of \( A \times B \) is a relation. We also define a relation among three or more sets to be a subset of the cross product of those sets. In particular, a relation on \( A, B, \) and \( C \) is a subset of \( A \times B \times C \).

For example, if \( P \) is the set of people and \( B \) is the set of books owned by a library, then we can define a relation \( \mathcal{R} \) on the sets \( P \) and \( B \) to be the set \( \mathcal{R} = \{ (p, b) \in P \times B \mid p \text{ has } b \text{ out on loan} \} \). The fact that a particular \( (p, b) \in \mathcal{R} \) is a fact about the world that the library will certainly want to keep track of. When a collection of facts about the world is stored on a computer, it is called a database. We’ll see in the next section that relations are the most common means of representing data in databases.

If \( A \) is a set and \( \mathcal{R} \) is a relation on the sets \( A \) and \( A \) (that is, on two copies of \( A \)), then \( \mathcal{R} \) is said to be a binary relation on \( A \). That is, a binary relation on the set \( A \) is a subset of \( A \times A \). The relation consisting of all ordered pairs \( (c, p) \) of people such that \( c \) is a child of \( p \) is a binary relation on the set of people. The set \( \{ (n, m) \in \mathbb{N} \times \mathbb{N} \mid n \leq m \} \) is a binary relation on \( \mathbb{N} \). Similarly, we define a ternary relation on a set \( A \) to be a
subset of $A \times A \times A$. The set $\{(x, y, z) \in \mathbb{R} \times \mathbb{R} \times \mathbb{R} \mid x^2 + y^2 + z^2 = 1\}$ is a ternary relation on $\mathbb{R}$. For complete generality, we can define an $n$-ary relation on $A$, for any positive integer $n$, to be a subset of $A \times A \times \cdots \times A$, where $A$ occurs $n$ times in the cross product.

For the rest of this section, we will be working exclusively with binary relations. Suppose that $R \subseteq A \times A$. That is, suppose that $R$ is a binary relation on a set $A$. If $(a, b) \in R$, then we say that $a$ is related to $b$ by $R$. Instead of writing “$(a, b) \in R$”, we will often write “$a R b$”. This notation is used in analogy to the notation $n \leq m$ to express the relation that $n$ is less than or equal to $m$. Remember that $a R b$ is just an alternative way of writing $(a, b) \in R$. In fact, we could consider the relation $\leq$ to be a set of ordered pairs and write $(n, m) \in \leq$ in place of the notation $n \leq m$.

In many applications, attention is restricted to relations that satisfy some property or set of properties. (This is, of course, just what we do when we study functions.) We begin our discussion of binary relations by considering several important properties. In this discussion, let $A$ be a set and let $R$ be a binary relation on $A$, that is, a subset of $A \times A$.

$R$ is said to be reflexive if $\forall a \in A \ (a \ R a)$. That is, a binary relation on a set is reflexive if every element of the set is related to itself. This is true, for example, for the relation $\leq$ on the set $\mathbb{N}$, since $n \leq n$ for every $n \in \mathbb{N}$. On the other hand, it is not true for the relation $<$ on $\mathbb{N}$, since, for example, the statement $17 < 17$ is false.\(^7\)

$R$ is called transitive if $\forall a \in A, \forall b \in A, \forall c \in A \ ((a \ R b \land b \ R c) \rightarrow (a \ R c))$. Transitivity allows us to “chain together” two true statements $a R b$ and $b R c$, which are “linked” by the $b$ that occurs in each statement, to deduce that $a R c$. For example, suppose $P$ is the set of people, and define the relation $C$ on $P$ such that $x P y$ if and only if $x$ is a child of $y$. The relation $P$ is not transitive because the child of a child is not a child of that person. Suppose, on the other hand, that we define a relation $D$ on $P$ such that $x D y$ if and only if $x$ is a descendent of $y$. Then $D$ is a transitive relation on the set of people, since a descendent of a descendent of a person is a descendent of that person. That is, from the facts that Elizabeth is a descendent of Victoria and Victoria is a descendent of James, we can deduce that Elizabeth is a descendent of James. In the mathematical world, the relations $\leq$ and $<$ on the set $\mathbb{N}$ are both transitive.

$R$ is said to be symmetric if $\forall a \in A, \forall b \in B \ (a \ R b \rightarrow b \ R a)$. That is, whenever $a$ is related to $b$, it follows that $b$ is related to $a$. The relation “is

\(^7\)Note that to show that the relation $R$ is not reflexive, you only need to find one $a$ such that $a R a$ is false. This follows from the fact that $\neg(\forall a \in A \ (a \ R a)) \equiv \exists a \in A \ (\neg(a \ R a))$. A similar remark holds for each of the properties of relations that are discussed here.
2.7. RELATIONS

a first cousin of” on the set of people is symmetric, since whenever \( x \) is a first cousin of \( y \), we have automatically that \( y \) is a first cousin of \( x \). On the other hand, the “child of” relation is certainly not symmetric. The relation \( \leq \) on \( \mathbb{N} \) is not symmetric. From the fact that \( n \leq m \), we cannot conclude that \( m \leq n \). It is true for some \( n \) and \( m \) in \( \mathbb{N} \) that \( n \leq m \rightarrow m \leq n \), but it is not true for all \( n \) and \( m \) in \( \mathbb{N} \).

Finally, \( \mathcal{R} \) is antisymmetric if \( \forall a \in A, \forall b \in B ((a \mathcal{R} b \land b \mathcal{R} a) \rightarrow a = b) \). The relation \( \mathcal{R} \) is antisymmetric if for any two distinct elements \( x \) and \( y \) of \( A \), we can’t have both \( x \mathcal{R} y \) and \( y \mathcal{R} x \). The relation \( \leq \) on \( \mathbb{N} \) is antisymmetric because from the facts that \( n \leq m \) and \( m \leq n \), we can deduce that \( n = m \). The relation “child of” on the set of people is antisymmetric since it’s impossible to have both that \( x \) is a child of \( y \) and \( y \) is a child of \( x \).

There are a few combinations of properties that define particularly useful types of binary relations. The relation \( \leq \) on the set \( \mathbb{N} \) is reflexive, antisymmetric, and transitive. These properties define what is called a partial order: A partial order on a set \( A \) is a binary relation on \( A \) that is reflexive, antisymmetric, and transitive.

Another example of a partial order is the subset relation, \( \subseteq \), on the power set of any set. If \( X \) is a set, then of course \( \mathcal{P}(X) \) is a set in its own right, and \( \subseteq \) can be considered to be a binary relation on this set. Two elements \( A \) and \( B \) of \( \mathcal{P}(X) \) are related by \( \subseteq \) if and only if \( A \subseteq B \). This relation is reflexive since every set is a subset of itself. The fact that it is antisymmetric follows from Theorem 2.1. The fact that it is transitive was Exercise 11 in Section 2.1.

The ordering imposed on \( \mathbb{N} \) by \( \leq \) has one important property that the ordering of subsets by \( \subseteq \) does not share. If \( n \) and \( m \) are natural numbers, then at least one of the statements \( n \leq m \) and \( m \leq n \) must be true. However, if \( A \) and \( B \) are subsets of a set \( X \), it is certainly possible that both \( A \subseteq B \) and \( B \subseteq A \) are false. A binary relation \( \mathcal{R} \) on a set \( A \) is said to be a total order if it is a partial order and furthermore for any two elements \( a \) and \( b \) of \( A \), either \( a \mathcal{R} b \) or \( b \mathcal{R} a \). The relation \( \leq \) on the set \( \mathbb{N} \) is a total order. The relation \( \subseteq \) on \( \mathcal{P}(X) \) is not. (Note once again the slightly odd mathematical language: A total order is a kind of partial order—not, as you might expect, the opposite of a partial order.)

For another example of ordering, let \( L \) be the set of strings that can be made from lowercase letters. \( L \) contains both English words and nonsense strings such as “sxjja”. There is a commonly used total order on the set \( L \), namely alphabetical order.

We’ll approach another important kind of binary relation indirectly, through what might at first appear to be an unrelated idea. Let \( A \) be a
A partition of a set \( A \) is defined to be a collection of non-empty subsets of \( A \) such that each pair of distinct subsets in the collection is disjoint and the union of all the subsets in the collection is \( A \). A partition of \( A \) is just an division of all the elements of \( A \) into non-overlapping subsets. For example, the sets \( \{1, 2, 6\} \), \( \{3, 7\} \), \( \{4, 5, 8, 10\} \), and \( \{9\} \) form a partition of the set \( \{1, 2, \ldots, 10\} \). Each element of \( \{1, 2, \ldots, 10\} \) occurs in exactly one of the sets that make up the partition. As another example, we can partition the set of all people into two sets, the set of males and the set of females. Biologists try to partition the set of all organisms into different species. Librarians try to partition books into various categories such as fiction, biography, and poetry. In the real world, classifying things into categories is an essential activity, although the boundaries between categories are not always well-defined. The abstract mathematical notion of a partition of a set models the real-world notion of classification. In the mathematical world, though, the categories are sets and the boundary between two categories is sharp.

In the real world, items are classified in the same category because they are related in some way. This leads us from partitions back to relations. Suppose that we have a partition of a set \( A \). We can define a relation \( \mathcal{R} \) on \( A \) by declaring that for any \( a \) and \( b \) in \( A \), \( a \mathcal{R} b \) if and only if \( a \) and \( b \) are members of the same subset in the partition. That is, two elements of \( A \) are related if they are in the same category. It is clear that the relation defined in this way is reflexive, symmetric, and transitive.

An equivalence relation is defined to be a binary relation that is reflexive, symmetric, and transitive. Any relation defined, as above, from a partition is an equivalence relation. Conversely, we can show that any equivalence relation defines a partition. Suppose that \( \mathcal{R} \) is an equivalence relation on a set \( A \). Let \( a \in A \). We define the equivalence class of \( a \) under the equivalence relation \( \mathcal{R} \) to be the subset \( [a]_{\mathcal{R}} \) defined as \( [a]_{\mathcal{R}} = \{ b \in A \mid b \mathcal{R} a \} \). That is, the equivalence class of \( a \) is the set of all elements of \( A \) that are related to \( a \). In most cases, we’ll assume that the relation in question is understood, and we’ll write \( [a] \) instead of \( [a]_{\mathcal{R}} \). Note that each equivalence class is a subset of \( A \). The following theorem shows that the collection of equivalence classes form a partition of \( A \).

**Theorem 2.12.** Let \( A \) be a set and let \( \mathcal{R} \) be an equivalence relation on \( A \). Then the collection of all equivalence classes under \( \mathcal{R} \) is a partition of \( A \).

**Proof.** To show that a collection of subsets of \( A \) is a partition, we must show that each subset is non-empty, that the intersection of two distinct subsets is empty, and that the union of all the subsets is \( A \).

If \( [a] \) is one of the equivalence classes, it is certainly non-empty, since
2.7. RELATIONS

129

\( a \in [a] \). (This follows from the fact that \( R \) is reflexive, and hence \( a \mathbin{R} a \).) To show that \( A \) is the union of all the equivalence classes, we just have to show that each element of \( A \) is a member of one of the equivalence classes. Again, the fact that \( a \in [a] \) for each \( a \in A \) shows that this is true.

Finally, we have to show that the intersection of two distinct equivalence classes is empty. Suppose that \( a \) and \( b \) are elements of \( A \) and consider the equivalence classes \([a]\) and \([b]\). We have to show that if \([a] \neq [b] \), then \([a] \cap [b] = \emptyset \). Equivalently, we can show the converse: If \([a] \cap [b] \neq \emptyset \) then \([a] = [b] \). So, assume that \([a] \cap [b] \neq \emptyset \). Saying that a set is not empty just means that the set contains some element, so there must be an \( x \in A \) such that \( x \in [a] \cap [b] \). Since \( x \in [a] \), \( x \mathbin{R} a \). Since \( R \) is symmetric, we also have \( a \mathbin{R} x \). Since \( x \in [b] \), \( x \mathbin{R} b \). Since \( R \) is transitive and since \((a \mathbin{R} x) \land (x \mathbin{R} b)\), it follows that \( a \mathbin{R} b \). Our object is to deduce that \([a] = [b] \). Since \([a]\) and \([b]\) are sets, they are equal if and only if \([a] \subseteq [b] \) and \([b] \subseteq [a] \). To show that \([a] \subseteq [b] \), let \( c \) be an arbitrary element of \([a] \). We must show that \( c \in [b] \). Since \( c \in [a] \), we have that \( c \mathbin{R} a \). And we have already shown that \( a \mathbin{R} b \). From these two facts and the transitivity of \( R \), it follows that \( c \mathbin{R} b \). By definition, this means that \( c \in [b] \). We have shown that any member of \([a] \) is a member of \([b] \) and therefore that \([a] \subseteq [b] \). The fact that \([b] \subseteq [a] \) can be shown in the same way. We deduce that \([a] = [b] \), which proves the theorem.

The point of this theorem is that if we can find a binary relation that satisfies certain properties, namely the properties of an equivalence relation, then we can classify things into categories, where the categories are the equivalence classes.

For example, suppose that \( U \) is a possibly infinite set. Define a binary relation \( \sim \) on \( \mathcal{P}(U) \) as follows: For \( X \) and \( Y \) in \( \mathcal{P}(U) \), \( X \sim Y \) if and only if there is a bijective function from the set \( X \) to the set \( Y \). In other words, \( X \sim Y \) means that \( X \) and \( Y \) have the same cardinality. Then \( \sim \) is an equivalence relation on \( \mathcal{P}(U) \). (The symbol \( \sim \) is often used to denote equivalence relations. It is usually read “is equivalent to.”) If \( X \in \mathcal{P}(U) \), then the equivalence class \([X]_\sim \) consists of all the subsets of \( U \) that have the same cardinality as \( X \). We have classified all the subsets of \( U \) according to their cardinality—even though we have never said what an infinite cardinality is. (We have only said what it means to have the same cardinality.)

You might remember a popular puzzle called Rubic’s Cube, a cube made of smaller cubes with colored sides that could be manipulated by twisting layers of little cubes. The object was to manipulate the cube so that the colors of the little cubes formed a certain configuration. Define two con-
A few more examples of transitive closures in the exercises.

As a sequence of one or more steps. This relationship defines a binary relation on \( A \) that is called the transitive closure of \( \mathcal{R} \). The transitive closure of \( \mathcal{R} \) is denoted \( \mathcal{R}^* \). Formally, \( \mathcal{R}^* \) is defined as follows: For \( a \) and \( b \) in \( A \), \( a \mathcal{R}^* b \) if there is a sequence \( x_0, x_1, \ldots, x_n \) of elements of \( A \), where \( n > 0 \) and \( x_0 = a \) and \( x_n = b \), such that \( x_0 \mathcal{R} x_1, x_1 \mathcal{R} x_2, \ldots, \) and \( x_{n-1} \mathcal{R} x_n \).

For example, if \( a \mathcal{R} c, c \mathcal{R} d, \) and \( d \mathcal{R} b \), then we would have that \( a \mathcal{R}^* b \).

Of course, we would also have that \( a \mathcal{R}^* c \), and \( a \mathcal{R}^* d \).

For a practical example, suppose that \( C \) is the set of all cities and let \( \mathcal{A} \) be the binary relation on \( C \) such that for \( x \) and \( y \) in \( C \), \( x \mathcal{A} y \) if there is a regularly scheduled airline flight from \( x \) to \( y \). Then the transitive closure \( \mathcal{A}^* \) has a natural interpretation: \( x \mathcal{A}^* y \) if it’s possible to get from \( x \) to \( y \) by a sequence of one or more regularly scheduled airline flights. You’ll find a few more examples of transitive closures in the exercises.

Exercises

1. For a finite set, it is possible to define a binary relation on the set by listing the elements of the relation, considered as a set of ordered pairs. Let \( A \) be the set \( \{a, b, c, d\} \), where \( a, b, c, \) and \( d \) are distinct. Consider each of the following binary relations on \( A \). Is the relation reflexive? Symmetric? Antisymmetric? Transitive? Is it a partial order? An equivalence relation?
   
   a) \( \mathcal{R} = \{(a, b), (a, c), (a, d)\} \).
   
   b) \( \mathcal{S} = \{(a, a), (b, b), (c, c), (d, d), (a, b), (b, a)\} \).
   
   c) \( \mathcal{I} = \{(b, b), (c, c), (d, d)\} \).
   
   d) \( \mathcal{C} = \{(a, b), (b, c), (a, c), (d, d)\} \).
   
   e) \( \mathcal{D} = \{(a, b), (b, a), (c, d), (d, c)\} \).

2. Let \( A \) be the set \( \{1, 2, 3, 4, 5, 6\} \). Consider the partition of \( A \) into the subsets \( \{1, 4, 5\} \), \( \{3\} \), and \( \{2, 6\} \). Write out the associated equivalence relation on \( A \) as a set of ordered pairs.
3. Consider each of the following relations on the set of people. Is the relation reflexive? Symmetric? Transitive? Is it an equivalence relation?
   a) $x$ is related to $y$ if $x$ and $y$ have the same biological parents.
   b) $x$ is related to $y$ if $x$ and $y$ have at least one biological parent in common.
   c) $x$ is related to $y$ if $x$ and $y$ were born in the same year.
   d) $x$ is related to $y$ if $x$ is taller than $y$.
   e) $x$ is related to $y$ if $x$ and $y$ have both visited Honolulu.

4. It is possible for a relation to be both symmetric and antisymmetric. For example, the equality relation, $=$, is a relation on any set which is both symmetric and antisymmetric. Suppose that $A$ is a set and $\mathcal{R}$ is a relation on $A$ that is both symmetric and antisymmetric. Show that $\mathcal{R}$ is a subset of $=$ (when both relations are considered as sets of ordered pairs). That is, show that for any $a$ and $b$ in $A$, $(a \mathcal{R} b) \rightarrow (a = b)$.

5. Let $\sim$ be the relation on $\mathbb{R}$, the set of real numbers, such that for $x$ and $y$ in $\mathbb{R}$, $x \sim y$ if and only if $x - y \in \mathbb{Z}$. For example, $\sqrt{2} - 1 \sim \sqrt{2} + 17$ because the difference, $(\sqrt{2} - 1) - (\sqrt{2} + 17)$, is $-18$, which is an integer. Show that $\sim$ is an equivalence relation. Show that each equivalence class $[x]_\sim$ contains exactly one number $a$ which satisfies $0 \leq a < 1$. (Thus, the set of equivalence classes under $\sim$ is in one-to-one correspondence with the half-open interval $[0, 1)$.)

6. Let $A$ and $B$ be any sets, and suppose $f: A \rightarrow B$. Define a relation $\sim$ on $B$ such that for any $x$ and $y$ in $A$, $x \sim y$ if and only if $f(x) = f(y)$. Show that $\sim$ is an equivalence relation on $A$.

7. Let $\mathbb{Z}^+$ be the set of positive integers $\{1, 2, 3, \ldots\}$. Define a binary relation $\mathcal{D}$ on $\mathbb{Z}^+$ such that for $n$ and $m$ in $\mathbb{Z}^+$, $n \mathcal{D} m$ if $n$ divides evenly into $m$, with no remainder. Equivalently, $n \mathcal{D} m$ if $n$ is a factor of $m$, that is, if there is a $k$ in $\mathbb{Z}^+$ such that $m = nk$. Show that $\mathcal{D}$ is a partial order.

8. Consider the set $\mathbb{N} \times \mathbb{N}$, which consists of all ordered pairs of natural numbers. Since $\mathbb{N} \times \mathbb{N}$ is a set, it is possible to have binary relations on $\mathbb{N} \times \mathbb{N}$. Such a relation would be a subset of $(\mathbb{N} \times \mathbb{N}) \times (\mathbb{N} \times \mathbb{N})$. Define a binary relation $\preceq$ on $\mathbb{N} \times \mathbb{N}$ such that for $(m, n)$ and $(k, \ell)$ in $\mathbb{N} \times \mathbb{N}$, $(m, n) \preceq (k, \ell)$ if and only if either $m < k$ or $((m = k) \land (n \leq \ell))$. Which of the following are true?
   a) $(2, 7) \preceq (5, 1)$
   b) $(8, 5) \preceq (8, 0)$
   c) $(0, 1) \preceq (0, 2)$
   d) $(17, 17) \preceq (17, 17)$

Show that $\preceq$ is a total order on $\mathbb{N} \times \mathbb{N}$.

9. Let $\sim$ be the relation defined on $\mathbb{N} \times \mathbb{N}$ such that $(n, m) \sim (k, \ell)$ if and only if $n + \ell = m + k$. Show that $\sim$ is an equivalence relation.

10. Let $P$ be the set of people and let $\mathcal{C}$ be the “child of” relation. That is $x \mathcal{C} y$ means that $x$ is a child of $y$. What is the meaning of the transitive closure $\mathcal{C}^*$? Explain your answer.

11. Let $\mathcal{R}$ be the binary relation on $\mathbb{N}$ such that $x \mathcal{R} y$ if and only if $y = x + 1$. Identify the transitive closure $\mathcal{R}^*$. (It is a well-known relation.) Explain your answer.
12. Suppose that $R$ is a reflexive, symmetric binary relation on a set $A$. Show that the transitive closure $R^*$ is an equivalence relation.

2.8 Application: Relational Databases

One of the major uses of computer systems is to store and manipulate collections of data. A database is a collection of data that has been organized so that it is possible to add and delete information, to update the data that it contains, and to retrieve specified parts of the data. A Database Management System, or DBMS, is a computer program that makes it possible to create and manipulate databases. A DBMS must be able to accept and process commands that manipulate the data in the databases that it manages. These commands are called queries, and the languages in which they are written are called query languages. A query language is a kind of specialized programming language.

There are many different ways that the data in a database could be represented. Different DBMS’s use various data representations and various query languages. However, data is most commonly stored in relations. A relation in a database is a relation in the mathematical sense. That is, it is a subset of a cross product of sets. A database that stores its data in relations is called a relational database. The query language for most relational database management systems is some form of the language known as Structured Query Language, or SQL. In this section, we’ll take a very brief look at SQL, relational databases, and how they use relations.

A relation is just a subset of a cross product of sets. Since we are discussing computer representation of data, the sets are data types. As in Section 2.5, we’ll use data type names such as int and string to refer to these sets. A relation that is a subset of the cross product $\text{int} \times \text{int} \times \text{string}$ would consist of ordered 3-tuples such as $(17, 42, \text{“hike”})$. In a relational database, the data is stored in the form of one or more such relations. The relations are called tables, and the tuples that they contain are called rows or records.

As an example, consider a lending library that wants to store data about its members, the books that it owns, and which books the members have out on loan. This data could be represented in three tables, as illustrated in Figure 2.4. The relations are shown as tables rather than as sets of ordered tuples, but each table is, in fact, a relation. The rows of the table are the tuples. The Members table, for example, is a subset of $\text{int} \times \text{string} \times \text{string} \times \text{string}$, and one of the tuples is $(1782, \text{“Smith, John”}, \text{“107 Main St”}, \text{“New York, NY”})$. A table does have one thing that ordinary relations
in mathematics do not have. Each column in the table has a name. These names are used in the query language to manipulate the data in the tables.

The data in the Members table is the basic information that the library needs in order to keep track of its members, namely the name and address of each member. A member also has a MemberID number, which is presumably assigned by the library. Two different members can't have the same MemberID, even though they might have the same name or the same address. The MemberID acts as a **primary key** for the Members table. A given value of the primary key uniquely identifies one of the rows of the table. Similarly, the BookID in the Books table is a primary key for that table. In the Loans table, which holds information about which books are out on loan to which members, a MemberID unambiguously identifies the member who has a given book on loan, and the BookID says unambiguously which book that is. Every table has a primary key, but the key can consist of more than one column. The DBMS enforces the uniqueness of primary keys. That is, it won't let users make a modification to the table if it would result in two rows having the same primary key.

The fact that a relation is a set—a set of tuples—means that it can't contain the same tuple more than once. In terms of tables, this means that a table shouldn't contain two identical rows. But since no two rows can contain the same primary key, it's impossible for two rows to be identical. So tables are in fact relations in the mathematical sense.

The library must have a way to add and delete members and books and to make a record when a book is borrowed or returned. It should also have a way to change the address of a member or the due date of a borrowed book. Operations such as these are performed using the DBMS's query language. SQL has commands named **INSERT, DELETE, and UPDATE** for performing these operations. The command for adding Barack Obama as a member of the library with MemberID 999 would be

```
INSERT INTO Members
VALUES (999, "Barack Obama",
       "1600 Pennsylvania Ave", "Washington, DC")
```

When it comes to deleting and modifying rows, things become more interesting because it's necessary to specify which row or rows will be affected. This is done by specifying a condition that the rows must fulfill. For example, this command will delete the member with ID 4277:

```
DELETE FROM Members
WHERE MemberID = 4277
```

It's possible for a command to affect multiple rows. For example,
DELETE FROM Members
WHERE Name = "Smith, John"

would delete every row in which the name is “Smith, John.” The update command also specifies what changes are to be made to the row:

UPDATE Members
SET Address="19 South St", City="Hartford, CT"
WHERE MemberID = 4277

Of course, the library also needs a way of retrieving information from the database. SQL provides the `SELECT` command for this purpose. For example, the query

```
SELECT Name, Address
FROM Members
WHERE City = "New York, NY"
```

asks for the name and address of every member who lives in New York City. The last line of the query is a condition that picks out certain rows of the “Members” relation, namely all the rows in which the `City` is “New York, NY”. The first line specifies which data from those rows should be retrieved. The data is actually returned in the form of a table. For example, given the data in Figure 2.4, the query would return this table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith, John</td>
<td>107 Main St</td>
</tr>
<tr>
<td>Jones, Mary</td>
<td>1515 Center Ave</td>
</tr>
<tr>
<td>Lee, Joseph</td>
<td>90 Park Ave</td>
</tr>
<tr>
<td>O'Neil, Sally</td>
<td>89 Main St</td>
</tr>
</tbody>
</table>

The table returned by a `SELECT` query can even be used to construct more complex queries. For example, if the table returned by `SELECT` has only one column, then it can be used with the `IN` operator to specify any value listed in that column. The following query will find the `BookID` of every book that is out on loan to a member who lives in New York City:

```
SELECT BookID
FROM Loans
WHERE MemberID IN (SELECT MemberID
                    FROM Members
                    WHERE City = "New York, NY")
```

More than one table can be listed in the `FROM` part of a query. The tables that are listed are joined into one large table, which is then used for the
query. The large table is essentially the cross product of the joined tables, when the tables are understood as sets of tuples. For example, suppose that we want the titles of all the books that are out on loan to members who live in New York City. The titles are in the Books table, while information about loans is in the Loans table. To get the desired data, we can join the tables and extract the answer from the joined table:

```
SELECT Title
FROM Books, Loans
WHERE MemberID IN (SELECT MemberID
                     FROM Members
                     WHERE City = "New York, NY")
```

In fact, we can do the same query without using the nested SELECT. We need one more bit of notation: If two tables have columns that have the same name, the columns can be named unambiguously by combining the table name with the column name. For example, if the Members table and Loans table are both under discussion, then the MemberID columns in the two tables can be referred to as Members.MemberID and Loans.MemberID. So, we can say:

```
SELECT Title
FROM Books, Loans
WHERE City ="New York, NY"
    AND Members.MemberID = Loans.MemberID
```

This is just a sample of what can be done with SQL and relational databases. The conditions in WHERE clauses can get very complicated, and there are other operations besides the cross product for combining tables. The database operations that are needed to complete a given query can be complex and time-consuming. Before carrying out a query, the DBMS tries to optimize it. That is, it manipulates the query into a form that can be carried out most efficiently. The rules for manipulating and simplifying queries form an algebra of relations, and the theoretical study of relational databases is in large part the study of the algebra of relations.

**Exercises**

1. Using the library database from Figure 2.4, what is the result of each of the following SQL commands?

   a) SELECT Name, Address
      FROM Members
WHERE Name = "Smith, John"

b) DELETE FROM Books
WHERE Author = "Isaac Asimov"

c) UPDATE Loans
SET DueDate = "November 20"
WHERE BookID = 221

d) SELECT Title
FROM Books, Loans

e) DELETE FROM Loans
WHERE MemberID IN (SELECT MemberID
FROM Members
WHERE Name = "Lee, Joseph")

2. Using the library database from Figure 2.4, write an SQL command to do each of the following database manipulations:
   a) Find the BookID of every book that is due on November 1, 2010.
   b) Change the DueDate of the book with BookID 221 to November 15, 2010.
   c) Change the DueDate of the book with title “Summer Lightning” to November 14, 2010. Use a nested SELECT.
   d) Find the name of every member who has a book out on loan. Use joined tables in the FROM clause of a SELECT command.

3. Suppose that a college wants to use a database to store information about its students, the courses that are offered in a given term, and which students are taking which courses. Design tables that could be used in a relational database for representing this data. Then write SQL commands to do each of the following database manipulations. (You should design your tables so that they can support all these commands.)
   a) Enroll the student with ID number 1928882900 in “English 260”.
   b) Remove “John Smith” from “Biology 110”.
   c) Remove the student with ID number 2099299001 from every course in which that student is enrolled.
   d) Find the names and addresses of the students who are taking “Computer Science 229”.
   e) Cancel the course “History 101”.
2.8. APPLICATION: RELATIONAL DATABASES

<table>
<thead>
<tr>
<th>MemberID</th>
<th>Name</th>
<th>Address</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>1782</td>
<td>Smith, John</td>
<td>107 Main St</td>
<td>New York, NY</td>
</tr>
<tr>
<td>2889</td>
<td>Jones, Mary</td>
<td>1515 Center Ave</td>
<td>New York, NY</td>
</tr>
<tr>
<td>378</td>
<td>Lee, Joseph</td>
<td>90 Park Ave</td>
<td>New York, NY</td>
</tr>
<tr>
<td>4277</td>
<td>Smith, John</td>
<td>2390 River St</td>
<td>Newark, NJ</td>
</tr>
<tr>
<td>5704</td>
<td>O’Neil, Sally</td>
<td>89 Main St</td>
<td>New York, NY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BookID</th>
<th>Title</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>182</td>
<td>I, Robot</td>
<td>Isaac Asimov</td>
</tr>
<tr>
<td>221</td>
<td>The Sound and the Fury</td>
<td>William Faulkner</td>
</tr>
<tr>
<td>38</td>
<td>Summer Lightning</td>
<td>P.G. Wodehouse</td>
</tr>
<tr>
<td>437</td>
<td>Pride and Prejudice</td>
<td>Jane Austen</td>
</tr>
<tr>
<td>598</td>
<td>Left Hand of Darkness</td>
<td>Ursula LeGuin</td>
</tr>
<tr>
<td>629</td>
<td>Foundation Trilogy</td>
<td>Isaac Asimov</td>
</tr>
<tr>
<td>720</td>
<td>Mirror Dance</td>
<td>Lois McMaster Bujold</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MemberID</th>
<th>BookID</th>
<th>DueDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>378</td>
<td>221</td>
<td>October 8, 2010</td>
</tr>
<tr>
<td>2889</td>
<td>182</td>
<td>November 1, 2010</td>
</tr>
<tr>
<td>4277</td>
<td>221</td>
<td>November 1, 2010</td>
</tr>
<tr>
<td>1782</td>
<td>38</td>
<td>October 30, 2010</td>
</tr>
</tbody>
</table>

*Figure 2.4: Tables that could be part of a relational database. Each table has a name, shown above the table. Each column in the table also has a name, shown in the top row of the table. The remaining rows hold the data.*