Ministry of Education and Science of Ukraine
Sumy State University

Dvornichenko A. V.,
Lysenko O. V.

DISCRETE MATHEMATICS AND THEORY OF ALGORITHMS

Lecture notes

In four parts
Part I

Sumy
Sumy State University
2022

Ministry of Education and Science of Ukraine
Sumy State University

DISCRETE MATHEMATICS AND THEORY OF ALGORITHMS

Lecture notes

for students of speciality *113"Applied Mathematics"*
of full-time course of studies

In four parts
Part I

Approved at the meeting
of the department of Applied Mathematics and Complex Systems Modeling as a lecture notes on the discipline "Discrete Mathematics and Theory of Algorithms".
Minutes №5 of 29.06.2022.

Sumy
Sumy State University
2022

Discrete Mathematics and Theory of Algorithms : lecture notes : in four parts / compilers: A. V. Dvornichenko, O. V. Lysenko. — Sumy : Sumy State University, 2022. — Part I. — 275 p.

Department of Applied Mathematics and Modeling of Complex Systems

Contents

Introduction

A discrete mathematics and theory of algorithms course has more than one purpose. Students should learn a particular set of mathematical facts and how to apply them; more importantly, such a course should teach students how to think logically and mathematically. To achieve these goals, this text stresses mathematical reasoning and the different ways problems are solved. Five important themes are interwoven in this text: mathematical reasoning, combinatorial analysis, discrete structures, algorithmic thinking, and applications and modeling. A successful discrete mathematics and theory of algorithms course should carefully blend and balance all five themes.

1. **Mathematical Reasoning:** Students must understand mathematical reasoning in order to read, comprehend, and construct mathematical arguments. This text starts with a discussion of mathematical logic, which serves as the foundation for the subsequent discussions of methods of proof. Both the science and the art of constructing proofs are addressed. The technique of mathematical induction is stressed through many different types of examples of such proofs and a careful explanation of why mathematical induction is a valid proof technique.
2. **Combinatorial Analysis:** An important problem-solving skill is the ability to count or enumerate objects. The discussion of enumeration in this book begins with the basic techniques of counting. The stress is on performing combinatorial analysis to solve counting problems and analyze algorithms, not on applying formulae.
3. **Discrete Structures:** A course in discrete mathematics should

teach students how to work with discrete structures, which are the abstract mathematical structures used to represent discrete objects and relationships between these objects. These discrete structures include sets, permutations, relations, graphs, trees, and finite-state machines.

4. **Algorithmic Thinking:** Certain classes of problems are solved by the specification of an algorithm. After an algorithm has been described, a computer program can be constructed implementing it. The mathematical portions of this activity, which include the specification of the algorithm, the verification that it works properly, and the analysis of the computer memory and time required to performit, are all covered in this text. Algorithms are described using both English and an easily understood form of pseudocode.
5. **Applications and Modeling:** Discrete mathematics has applications to almost every conceivable area of study. There are many applications to computer science and data networking in this text, as well as applications to such diverse areas as chemistry, biology, linguistics, geography, business, and the Internet. These applications are natural and important uses of discrete mathematics and are not contrived. Modeling with discrete mathematics is an extremely important problem-solving skill, which students have the opportunity to develop by constructing their own models in some of the exercises. That is why there are many different examples in this text.

In the lecture materials, we used a number of books, among which we will single out the wonderful book by [1]. In the first part of the lecture notes, we consider the following sections: “Basics: logic and proofs”, “Basic structures: sets, functions, sequences and sums” and “Algorithms”.

Chapter 1

FUNDAMENTALS OF MATHEMATICAL LOGIC

1.1 Propositional Logic

1.1.1 Introduction

The rules of logic give precise meaning to mathematical statements. These rules are used to distinguish between valid and invalid mathematical arguments. Because a major goal of this book is to teach the reader how to understand and how to construct correct mathematical arguments, we begin our study of discrete mathematics with an introduction to logic.

Besides the importance of logic in understanding mathematical reasoning, logic has numerous applications to computer science. These rules are used in the design of computer circuits, the construction of computer programs, the verification of the correctness of programs, and in many other ways. Furthermore, software systems have been developed for constructing some, but not all, types of proofs automatically.We will discuss these applications of logic in this and later chapters.

1.1.2 Propositions

Our discussion begins with an introduction to the basic building blocks of logic propositions. A **proposition** is a declarative sentence (that is, a sentence that declares a fact) that is either true or false, but not both.

 EXAMPLE. 1

All the following declarative sentences are propositions.

1. Washington, D.C., is the capital of the United States of America.
2. Toronto is the capital of Canada.
3. $1 + 1 = 2$.
4. $2 + 2 = 3$.

Propositions 1 and 3 are true, whereas 2 and 4 are false. ∎

Some sentences that are not propositions are given in this Example 2.

 EXAMPLE. 2

Consider the following sentences.

1. What time is it?
2. Read this carefully.
3. $x + 1 = 2$.
4. $x + y = z$. ∎

Sentences 1 and 2 are not propositions because they are not declarative sentences. Sentences 3 and 4 are not propositions because they are neither true nor false. Note that each of sentences 3 and 4 can be turned into a proposition if we assign values to the variables. We will also discuss other ways to turn sentences such as these into propositions in Section 1.4.

We use letters to denote **propositional variables (or sentential variables)**, that is, variables that represent propositions, just as letters are used to denote numerical variables. The conventional letters used

for propositional variables are p, q, r, s, ... The **truth value** of a proposition is true, denoted by T, if it is a true proposition, and the truth value of a proposition is false, denoted by F, if it is a false proposition. Propositions that cannot be expressed in terms of simpler propositions are called **atomic propositions**.

The area of logic that deals with propositions is called the **propositional calculus or propositional logic**. It was first developed systematically by the Greek philosopher Aristotle more than 2300 years ago.

We now turn our attention to methods for producing new propositions from those that we already have. These methods were discussed by the English mathematician George Boole in 1854 in his book *The Laws of Thought*. Many mathematical statements are constructed by combining one or more propositions. New propositions, called **compound propositions**, are formed from existing propositions using **logical operators**.

Definition 1.1.1 *Let p be a proposition. The negation of p, denoted by $\neg p$ (also denoted by $\bar{p}$), is the statement*

"It is not the case that p."

The proposition $\neg p$ is read "not p". The truth value of the negation of p, $\neg p$, is the opposite of the truth value of p.

Remark! The notation for the negation operator is not standardized. Although $\neg p$ and $\bar{p}$ are the most common notations used in mathematics to express the negation of p, other notations you might see are $\sim p$, $-p$, p', Np, and $!p$.

EXAMPLE. 3

Find the negation of the proposition "Michael's PC runs Linux" and express this in simple English.

Solution: The negation is "It is not the case that Michael's PC runs Linux." This negation can be more simply expressed as "Michael's PC does not run Linux." ∎

Table 1.1: The Truth Table for the Negation of a Proposition.

p	$\neg p$
T	F
F	T

Table 1.1 displays the **truth table** for the negation of a proposition p. This table has a row for each of the two possible truth values of p. Each row shows the truth value of $\neg p$ corresponding to the truth value of p for this row.

The negation of a proposition can also be considered the result of the operation of the **negation operator** on a proposition. The negation operator constructs a new proposition from a single existing proposition. We will now introduce the logical operators that are used to form new propositions from two or more existing propositions. These logical operators are also called **connectives**.

Definition 1.1.2 *Let p and q be propositions. The conjunction of p and q, denoted by $p \wedge q$, is the proposition "p and q." The conjunction $p \wedge q$ is true when both p and q are true and is false otherwise.*

Table 1.2: The Truth Table for the Conjunction of Two Propositions.

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Table 1.2 displays the truth table of $p \wedge q$. This table has a row for each of the four possible combinations of truth values of p and q. The four rows correspond to the pairs of truth values TT, TF, FT, and FF, where the first truth value in the pair is the truth value of p and the second truth value is the truth value of q.

Note that in logic the word "but" sometimes is used instead of "and" in a conjunction. For example, the statement "The sun is shining, but it is raining" is another way of saying "The sun is shining and it is raining."

EXAMPLE. 4

Find the conjunction of the propositions p and q where p is the proposition "Rebecca's PC has more than 16 GB free hard disk space" and q is the proposition "The processor in Rebecca's PC runs faster than 1 GHz."

Solution: The conjunction of these propositions, $p \wedge q$, is the proposition "Rebecca's PC has more than 16 GB free hard disk space, and the processor in Rebecc's PC runs faster than 1 GHz." This conjunction can be expressed more simply as "Rebecca's PC has more than 16 GB free hard disk space, and its processor runs faster than 1 GHz." For this conjunction to be true, both conditions given must be true. It is false when one or both of these conditions are false. ∎

Definition 1.1.3 *Let p and q be propositions. The disjunction of p and q, denoted by $p \vee q$, is the proposition "p or q." The disjunction $p \vee q$ is false when both p and q are false and is true otherwise.*

Table 1.3 displays the truth table for $p \vee q$.

Table 1.3: The Truth Table for the Disjunction of Two Propositions.

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

The use of the connective *or* in a disjunction corresponds to one of the two ways the word *or* is used in English, namely, as an **inclusive or**. A disjunction is true when at least one of the two propositions is true. That is, $p \vee q$ is true when both p and q are true or when exactly one of p and q is true.

EXAMPLE. 5

Translate the statement "Students who have taken calculus or introductory computer science can take this class" in a statement in propositional logic using the propositions p: "A student who has taken calculus can take this class" and q: "A student who has taken introductory computer science can take this class."

Solution: We assume that this statement means that students who have taken both calculus and introductory computer science can take the class, as well as the students who have taken only one of the two subjects. Hence, this statement can be expressed as $p \vee q$, the inclusive or, or disjunction, of p and q. ∎

EXAMPLE. 6

What is the disjunction of the propositions p and q, where p and q are the same propositions as in Example 4?

Solution: The disjunction of p and q, $p \vee q$, is the proposition "Rebecca's PC has at least 16 GB free hard disk space, or the processor in Rebecca's PC runs faster than 1 GHz." ■

This proposition is true when Rebecca's PC has at least 16 GB free hard disk space, when the PC's processor runs faster than 1 GHz, and when both conditions are true. It is false when both of these conditions are false, that is, when Rebecca's PC has less than 16 GB free hard disk space and the processor in her PC runs at 1 GHz or slower.

Besides its use in disjunctions, the connective *or* is also used to express an *exclusive or*. Unlike the disjunction of two propositions p and q, the exclusive or of these two propositions is true when exactly one of p and q is true; it is false when both p and q are true (and when both are false).

Definition 1.1.4 *Let p and q be propositions. The exclusive or of p and q, denoted by (or $p \oplus q$), is the proposition that is true when exactly one of p and q is true and is false otherwise.*

The truth table for the exclusive or of two propositions is displayed in Table 1.4.

EXAMPLE. 7

Let p and q be the propositions that state "A student can have a salad with dinner" and "A student can have soup with dinner," respectively. What is $p \oplus q$, the exclusive or of p and q?

Solution: The exclusive or of p and q is the statement that is true when exactly one of p and q is true. That is, $p \oplus q$ is the statement "A student can have soup or salad, but not both, with dinner." Note that this is often stated as "A student can have soup or a salad with dinner," without explicitly

stating that taking both is not permitted. ∎

1.1.3 Conditional Statements

We will discuss several other important ways in which propositions can be combined.

Definition 1.1.5 *Let p and q be propositions. The conditional statement $p \to q$ is the proposition "if p, then q." The conditional statement $p \to q$ is false when p is true and q is false, and true otherwise. In the conditional statement $p \to q$, p is called the hypothesis (or antecedentor premise) and q is called the conclusion(or consequence).*

The statement $p \to q$ is called a conditional statement because $p \to q$ asserts that q is true on the condition that p holds. A conditional statement is also called an **implication**.

The truth table for the conditional statement $p \to q$ is shown in Table 1.5. Note that the statement $p \to q$ is true when both p and q are true and when p is false (no matter what truth value q has).

Because conditional statements play such an essential role in mathematical reasoning, a variety of terminology is used to express $p \to q$. You will encounter most if not all of the following ways to express this conditional statement:

"if p, then q"	"p implies q"
"if p, q"	"p only if q"
"p is sufficient for q"	"a sufficient condition for q is p"
"q if p"	"q whenever p"
"q when p"	"q is necessary for p"
"a necessary condition for p is q"	"q follows from p"
"q unless p"	"q provided that p"

A useful way to understand the truth value of a conditional statement is to think of an obligation or a contract. For example, the pledge many politicians make when running for office is

"If I am elected, then I will lower taxes."

Table 1.4: The Truth Table for the Exclusive Or of Two Propositions.

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Table 1.5: The Truth Table for the Conditional Statement $p \to q$.

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	F

If the politician is elected, voters would expect this politician to lower taxes. Furthermore, if the politician is not elected, then voters will not have any expectation that this person will lower taxes, although the person may have sufficient influence to cause those in power to lower taxes. It is only when the politician is elected but does not lower taxes that voters can say that the politician has broken the campaign pledge. This last scenario corresponds to the case when p is true but q is false in $p \to q$.

Similarly, consider a statement that a professor might make: "If you get 100% on the final, then you will get an A."

If you manage to get 100% on the final, then you would expect to receive an A. If you do not get 100%, you may or may not receive an A depending on other factors. However, if you do get 100%, but the professor does not give you an A, you will feel cheated.

Remark! Because some of the different ways to express the implication p implies q can be confusing, we will provide some extra guidance. To remember that "p only if q" expresses the same thing as "if p, then q," note that "p only if q" says that p cannot be true when q is not true. That is, the statement is false if p is true, but q is false. When p is false, q may be either true or false, because the statement says nothing about the truth value of q.

For example, suppose your professor tells you:

"You can receive an A in the course only if your score on the final is at least 90%."

Then, if you receive an A in the course, then you know that your score on the final is at least 90%. If you do not receive an A, you may or may not have scored at least 90% on the final. Be careful not to use "q only if p" to express $p \to q$ because this is incorrect. The word "only" plays an essential role here. To see this, note that the truth values of "q only if p" $p \to q$ are different when p and q have different truth values. To see why "q is necessary for p" is equivalent to "if p, then q," observe that "q is necessary for p" means that p cannot be true unless q is true, or that if q is false, then p is false. This is the same as saying that if p is true, then q is true. To see why "p is sufficient for q" is equivalent to "if p, then q," note that "p is sufficient for q" means if p is true, it must be the case that q is also true. This is the same as saying that if p is true, then q is also true.

To remember that "q unless $\neg p$" expresses the same conditional statement as "if p, then q" note that "q unless $\neg p$" means that if $\neg p$ is false, then q must be true. That is, the statement "q unless $\neg p$" is false when p is true but q is false, but it is true otherwise. Consequently, "q unless p" and $p \to q$ always have the same truth value.

We illustrate the translation between conditional statements and English statements in Example 8.

 EXAMPLE. 8

Let p be the statement "Maria learns discrete mathematics" and q the statement "Maria will find a good job." Express the statement $p \to q$ as a statement in English.

Solution: From the definition of conditional statements, we see that when p is the statement "Maria learns discrete mathematics" and q is the statement "Maria will find a good job," $p \to q$ represents the statement

"If Maria learns discrete mathematics, then she will find a good job."

There are many other ways to express this conditional statement in English. Among the most natural of these are

"Maria will find a good job when she learns discrete mathematics."// "For Maria to get a good job, it is sufficient for her to learn discrete mathematics."

and

"Maria will find a good job unless she does not learn discrete mathematics."

Note that the way we have defined conditional statements is more general than the meaning attached to such statements in the English language. For instance, the conditional statement in Example 8 and the statement

"If it is sunny, then we will go to the beach"

are statements used in normal language where there is a relationship between the hypothesis and the conclusion. Further, the first of these statements is true unless Maria learns discrete mathematics, but she does not get a good job, and the second is true unless it is indeed sunny, but we do not go to the beach. On the other hand, the statement

"If Juan has a smartphone, then $2 + 3 = 5$" are statements used in normal language where there is a relationship between the hypothesis and the conclusion. Further, the first of these statements is true unless Maria learns discrete mathematics, but she does not get a good job, and the second is true unless it is indeed sunny, but we do not go to the beach. On the other hand, the statement

"If Juan has a smartphone, then $2 + 3 = 5$"

is true from the definition of a conditional statement, because its conclusion is true. (The truth value of the hypothesis does not matter then.) The conditional statement

"If Juan has a smartphone, then $2 + 3 = 6$"

is true if Juan does not have a smartphone, even though $2 + 3 = 6$ is false.

We would not use these last two conditional statements in natural language (except perhaps in sarcasm), because there is no relationship between the hypothesis and the conclusion in either statement. In mathematical reasoning, we consider conditional statements of a more general sort than we use in English. The mathematical concept of a conditional statement is independent of a cause- and - efect relationship between hypothesis and conclusion. Our defnition of a conditional statement specifies its truth values; it is not based on English usage. Propositional language is an artificial language; we only parallel English usage to make it easy to use and remember. ∎

The if-then construction used in many programming languages is diferent from that used in logic. Most programming languages contain statements such as **if** p **then** S, where p is a proposition and S is a program segment (one or more statements to be executed). (Although this looks as if it might be a conditional statement, S is not a proposition, but rather is a set of executable instructions.) When execution of

a program encounters such a statement, S is executed if p is true, but S is not executed if p is false, as illustrated in Example 9.

EXAMPLE. 9

What is the value of the variable x after the statement

if $2 + 2 = 4$ **then** $x = x + 1$

if $x = 0$ before this statement is encountered? (The symbol = stands for assignment. The statement $x = x + 1$ means the assignment of the value of $x + 1$ to x.)

Solution: Because $2 + 2 = 4$ is true, the assignment statement $x = x + 1$ is executed. x has the value $0+1 = 1$ after this statement is encountered. ∎

CONVERSE, CONTRAPOSITIVE, AND INVERSE

We can form some new conditional statements starting with a conditional statement $p \to q$. In particular, there are three related conditional statements that occur so often that they have special names. The proposition $q \to p$ is called the **converse** of $p \to q$. The **contrapositive** of $p \to q$ is the proposition $\neg q \to \neg p$. The proposition $\neg p \to \neg q$ is called the **inverse** of $p \to q$. We will see that of these three conditional statements formed from $p \to q$, only the contrapositive always has the same truth value as $p \to q$.

We first show that the contrapositive, $\neg q \to \neg p$, of a conditional statement $p \to q$ always has the same truth value as $p \to q$. To see this, note that the contrapositive is false only when $\neg p$ is false and $\neg q$ is true, that is, only when p is true and q is false. We now show that neither the converse, $q \to p$, nor the inverse, $\neg p \to \neg q$, has the same truth value as $p \to q$ for all possible truth values of p and q. Note that when p is true and q is false, the original conditional statement is false, but the converse and the inverse are both true.

When two compound propositions always have the same truth values, regardless of the truth values of its propositional variables, we call them **equivalent**. Hence, a conditional statement and its contrapositive are equivalent. The converse and the inverse of a conditional statement are also equivalent, as the reader can verify, but neither is equivalent to the original conditional statement. (We will study equiv-

alent propositions in Section 1.3.) Take note that one of the most common logical errors is to assume that the converse or the inverse of a conditional statement is equivalent to this conditional statement.

We illustrate the use of conditional statements in Example 10.

EXAMPLE. 10

Find the contrapositive, the converse, and the inverse of the conditional statement

"The home team wins whenever it is raining."

Solution: Because "q whenever p" is one of the ways to express the conditional statement $p \to q$, the original statement can be rewritten as

"If it is raining, then the home team wins."

Consequently, the contrapositive of this conditional statement is

"If the home team does not win, then it is not raining."

The converse is

"If the home team wins, then it is raining."

The inverse is

"If it is not raining, then the home team does not win."

Only the contrapositive is equivalent to the original statement. ∎

BICONDITIONALS We now introduce another way to combine propositions that expresses that two propositions have the same truth value.

Definition 1.1.6 *Let p and q be propositions. The biconditional statement $p \leftrightarrow q$ is the proposition "p if and only if q." The biconditional statement $p \leftrightarrow q$ is true when p and q have the same truth values, and is false otherwise. Biconditional statements are also called bi-implications.*

The truth table $p \leftrightarrow q$ is shown in Table 6. Note that the statement$p \leftrightarrow q$ is true when both the conditional statements $p \to q$ and $q \to p$ are true and is false otherwise. That is why we use the words "if and only if" to express this logical connective and why it is symbolically written by combining the symbols $\to$ and $\leftarrow$. There are some other common ways to express $p \leftrightarrow q$:

"p is necessary and sufficient for q"
"if p then q, and conversely"
"p iff q." "p exactly when q."

The last way of expressing the biconditional statement $p \leftrightarrow q$ uses the abbreviation "iff" for "if and only if." Note that $p \leftrightarrow q$ has exactly the same truth value as $(p \leftrightarrow q) \wedge (q \leftrightarrow p)$.

EXAMPLE. 11

Let p be the statement "You can take the flight," and let q be the statement "You buy a ticket." Then $p \leftrightarrow q$ is the statement

"You can take the flight if and only if you buy a ticket."

This statement is true if p and q are either both true or both false, that is, if you buy a ticket and can take the flight or if you do not buy a ticket and you cannot take the flight. It is false when p and q have opposite truth values, that is, when you do not buy a ticket, but you can take the flight (such as when you get a free trip) and when you buy a ticket but you cannot take the flight (such as when the airline bumps you). ∎

Table 1.6: The Truth Table for the Biconditional $p \leftrightarrow q$.

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

IMPLICIT USE OF BICONDITIONALS

You should be aware that biconditionals are not always explicit in natural language. In particular, the "if and only if" construction used in biconditionals is rarely used in common language. Instead, biconditionals are often expressed using an "if, then" or an "only if" construction. The other part of the "if and only if" is implicit. That is, the converse is implied, but not stated. For example, consider the statement in English "If you finish your meal, then you can have dessert." What is really meant is "You can have dessert if and only if you finish your meal." This last statement is logically equivalent to the two statements "If you finish your meal, then you can have dessert" and "You can have dessert only if you

Table 1.7: The Truth Table of $(p \vee \neg q) \rightarrow (p \wedge q)$.

p	q	$\neg q$	$p \vee \neg q$	$p \wedge q$	$(p \vee \neg q) \rightarrow (p \wedge q)$
T	T	F	T	T	T
T	F	T	T	F	F
F	T	F	F	F	T
F	F	T	T	F	F

finish your meal." Because of this imprecision in natural language, we need to make an assumption whether a conditional statement in natural language implicitly includes its converse. Because precision is essential in mathematics and in logic, we will always distinguish between the conditional statement $p \rightarrow q$ and the biconditional statement $p \leftrightarrow q$.

1.1.4 Truth Tables of Compound Propositions

We have now introduced five important logical connectives—conjunction, disjunction, exclusive or, implication, and the biconditional operator as well as negation.We can use these connectives to build up complicated compound propositions involving any number of propositional variables.We can use truth tables to determine the truth values of these compound propositions, as Example 12 illustrates. We use a separate column to find the truth value of each compound expression that occurs in the compound proposition as it is built up. The truth values of the compound proposition for each combination of truth values of the propositional variables in it is found in the final column of the table.

EXAMPLE. 12

Construct the truth table of the compound proposition

$$(p \vee \neg q) \rightarrow (p \wedge q)$$

Solution: Because this truth table involves two propositional variables p and q, there are four rows in this truth table, one for each of the pairs of truth values TT, TF, FT, and FF. The first two columns are used for the truth values of p and q, respectively. In the third column we find the truth value of $\neg q$, needed to find the truth value of $p \vee \neg q$, found in the fourth column. The fifth column gives the truth value of $p \wedge q$. Finally, the truth value of $(p \vee \neg q) \rightarrow (p \wedge q)$ is found in the last column. The resulting truth table is shown in Table 1.7. ∎

1.1.5 Precedence of Logical Operators

We can construct compound propositions using the negation operator and the logical operators defined so far.We will generally use parentheses to specify the order in which logical operators in a compound proposition are to be applied. For instance, $(p \vee q) \wedge (\neg r)$ is the conjunction of $p \vee q$ and $\neg r$. However, to reduce the number of parentheses, we specify that the negation operator is applied before all other logical operators. This means that $\neg p \wedge q$ is the conjunction of $\neg p$ and q, namely, $(\neg p) \wedge q$, not the negation of the conjunction of p and q, namely $\neg(p \wedge q)$. (It is generally the case that unary operators that involve only one object precede binary operators.)

Table 1.8: Precedence of Logical Operators.

Operat	Preceden
$\neg$	1
$\wedge$	2
$\vee$	3
$\rightarrow$	4
$\leftrightarrow$	5

Another general rule of precedence is that the conjunction operator takes precedence over the disjunction operator, so that $p \wedge q \vee r$ means $(p \wedge q) \vee r$ rather than $p \wedge (q \vee r)$. Because this rule may be difficult to remember, we will continue to use parentheses so that the order of the disjunction and conjunction operators is clear.

Finally, it is an accepted rule that the conditional and biconditional operators, $\rightarrow$ and $\leftrightarrow$, have lower precedence than the conjunction and disjunction operators, $\wedge$ and $\vee$. Consequently, $p \vee q \rightarrow r$ is the

same as $(p \vee q) \rightarrow r$. We will use parentheses when the order of the conditional operator and biconditional operator is at issue, although the conditional operator has precedence over the biconditional operator. Table 1.8 displays the precedence levels of the logical operators,$\neg$, $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$.

1.1.6 Logic and Bit Operations

Computers represent information using bits. A bit is a symbol with two possible values, namely, 0 (zero) and 1 (one). This meaning of the word bit comes from binary digit, because zeros and ones are the digits used in binary representations of numbers. The well-known statistician John Tukey introduced this terminology in 1946. A bit can be used to represent a truth value, because there are two truth values, namely, *true* and *false*. As is customarily done, we will use a 1 bit to represent true and a 0 bit to represent false. That is, 1 represents T (true), 0 represents F (false). A variable is called a Boolean variable if its value is either true or false. Consequently, a **Boolean variable** can be represented using a bit.

Truth Value	Bit
T	1
F	0

Table 1.9: Table for the Bit Operators *OR, AND*, and *XOR*.

x	y	$x \vee y$	$x \wedge y$	$x \oplus y$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Computer **bit operations** correspond to the logical connectives. By replacing true by a one and false by a zero in the truth tables for the operators $\wedge$, $\vee$, $\oplus$ and , the columns in Table 1.9 for the corresponding bit operations are obtained.We will also use the notation *OR, AND*, and *XOR* for the operators $\vee$, $\wedge$, and $\oplus$ as is done in various programming languages.

Information is often represented using bit strings, which are lists of zeros and ones. When this is done, operations on the bit strings can be used to manipulate this information.

Definition 1.1.7 *A bit string is a sequence of zero or more bits. The length of this string is the number of bits in the string.*

EXAMPLE. 13

101010011 is a bit string of length nine.

We can extend bit operations to bit strings. We define the **bitwise OR, bitwise AND, and bitwise XOR** of two strings of the same length to be the strings that have as their bits the OR, AND, and XOR of the corresponding bits in the two strings, respectively. We use the symbols $\vee$, $\wedge$, and to represent the bitwise *OR*, bitwise *AND*, and bitwise *XOR* operations, respectively. We illustrate bitwise operations on bit strings with Example 14.

EXAMPLE. 14

Find the bitwise *OR*, bitwise *AND*, and bitwise *XOR* of the bit strings 01 1011 0110 and 11 0001 1101. (Here, and throughout this book, bit strings will be split into blocks of four bits to make them easier to read.)

Solution: The bitwise *OR*, bitwise *AND*, and bitwise XOR of these strings are obtained by taking the *OR, AND,* and *XOR* of the corresponding bits, respectively. This gives us

01 1011 0110
11 0001 1101
11 1011 1111 *bitwise OR*
01 0001 0100 *bitwise AND*
10 1010 1011 *bitwise XOR* ∎

1.2 Applications of Propositional Logic

1.2.1 Introduction

Logic has many important applications to mathematics, computer science, and numerous other disciplines. Statements in mathematics and the sciences and in natural language often are imprecise or ambiguous. To make such statements precise, they can be translated into the language of logic. For example, logic is used in the specification of software and hardware, because these specifications need to be precise before development begins. Furthermore, propositional logic and its rules can be used to design computer circuits, to construct computer programs, to verify the correctness of programs, and to build expert systems. Logic can be used to analyze and solve many familiar puzzles. Software systems based on the rules of logic have been developed for constructing some, but not all, types of proofs automatically.We will discuss some of these applications of propositional logic in this section and in later chapters.

1.2.2 Translating English Sentences

There are many reasons to translate English sentences into expressions involving propositional variables and logical connectives. In particular, English (and every other human language) is often ambiguous. Translating sentences into compound statements (and other types of logical expressions, which we will introduce later in this chapter) removes the ambiguity. Note that this may involve making a set of reasonable assumptions based on the intended meaning of the sentence. Moreover, once we have translated sentences from English into logical expressions we can analyze these logical expressions to determine their truth values, we can manipulate them, and we can use rules of inference to reason about them.

To illustrate the process of translating an English sentence into a logical expression, consider Examples 1 and 2.

EXAMPLE. 1

How can this English sentence be translated into a logical expression?

"You can access the Internet from campus only if you are a computer science major or you are not a freshman."

Solution: There are many ways to translate this sentence into a logical expression. Although it is possible to represent the sentence by a single propositional variable, such as p, this would not be useful when analyzing its meaning or reasoning with it. Instead, we will use propositional variables to represent each sentence part and determine the appropriate logical connectives between them. In particular, we let a, c, and $f$represent "You can access the Internet from campus," "You are a computer science major", and "You are a freshman", respectively. Noting that "only if" is one way a conditional statement can be expressed, this sentence can be represented as

$a \rightarrow (c \vee \neg f)$ ∎

EXAMPLE. 2

How can this English sentence be translated into a logical expression? "You cannot ride the roller coaster if you are under 4 feet tall unless you are older than 16 years old."

Solution: Let q, r, and s represent "You can ride the roller coaster", "You are under 4 feet tall", and "You are older than 16 years old", respectively. Then the sentence can be translated to

$(r \wedge \neg s) \rightarrow \neg q$.

Of course, there are other ways to represent the original sentence as a logical expression, but the one we have used should meet our needs. ∎

1.2.3 System Specifications

Translating sentences in natural language (such as English) into logical expressions is an essential part of specifying both hardware and software systems. System and software engineers take requirements in natural language and produce precise and unambiguous specifications

that can be used as the basis for system development. Example 3 shows how compound propositions can be used in this process.

EXAMPLE. 3

Express the specification "The automated reply cannot be sent when the file system is full" using logical connectives.

Solution: One way to translate this is to let p denote "The automated reply can be sent" and q denote "The file system is full". Then $\neg p$ represents "It is not the case that the automated reply can be sent", which can also be expressed as "The automated reply cannot be sent". Consequently, our specification can be represented by the conditional statement $q \rightarrow \neg p$. ∎

System specifications should be **consistent**, that is, they should not contain conflicting requirements that could be used to derive a contradiction.When specifications are not consistent, there would be no way to develop a system that satisfies all specifications.

EXAMPLE. 4

Determine whether these system specifications are consistent:

> "The diagnostic message is stored in the buffer or it is retransmitted."
> "The diagnostic message is not stored in the buffer."
> "If the diagnostic message is stored in the buffer, then it is retransmitted."

Solution: To determine whether these specifications are consistent, we first express them using logical expressions. Let p denote "The diagnostic message is stored in the buffer" and let q denote "The diagnostic message is retransmitted". The specifications can then be written as $p \vee q$, $\neg p$ and $p \rightarrow q$. An assignment of truth values that makes all three specifications true must have p false to make $\neg p$ true. Because we want $p \vee q$ to be true but p must be false, q must be true. Because $p \rightarrow q$ is true when p is false and q is true, we conclude that these specifications are consistent, because they are all true when p is false and q is true. We could come to the same conclusion by use of a truth table to examine the four possible assignments of truth values to p and q. ∎

EXAMPLE. 5

Do the system specifications in Example 4 remain consistent if the specification "The diagnostic message is not retransmitted" is added?

Solution: By the reasoning in Example 4, the three specifications from that example are true only in the case when p is false and q is true. However, this new specification is $\neg q$, which is false when q is true. Consequently, these four specifications are inconsistent. ∎

1.2.4 Boolean Searches

Logical connectives are used extensively in searches of large collections of information, such as indexes of Web pages. Because these searches employ techniques from propositional logic, they are called **Boolean searches**.

In Boolean searches, the connective *AND* is used to match records that contain both of two search terms, the connective *OR* is used to match one or both of two search terms, and the connective *NOT* (sometimes written as *AND NOT*) is used to exclude a particular search term. Careful planning of how logical connectives are used is often required when Boolean searches are used to locate information of potential interest. Example 6 illustrates how Boolean searches are carried out.

EXAMPLE. 6

Web Page Searching MostWeb search engines support Boolean searching techniques, which usually can help findWeb pages about particular subjects. For instance, using Boolean searching to find Web pages about universities in New Mexico, we can look for pages matching NEW *AND* MEXICO *AND* UNIVERSITIES. The results of this search will include those pages that contain the three words NEW, MEXICO, and UNIVERSITIES. This will include all of the pages of interest, together with others such as a page about new universities in Mexico. (Note that in Google, and many other search engines, the word "AND" is not needed, although it is understood, because all search terms are included by default. These search engines also support the use of

quotation marks to search for specific phrases. So, it may be more effective to search for pages matching "New Mexico" *AND* UNIVERSITIES.)

Next, to find pages that deal with universities in New Mexico or Arizona, we can search for pages matching (NEW *AND* MEXICO *OR* ARIZONA) *AND* UNIVERSITIES. (Note: Here the *AND* operator takes precedence over the *OR* operator. Also, in Google, the terms used for this search would be NEW MEXICO *OR* ARIZONA.) The results of this search will include all pages that contain the word UNIVERSITIES and either both the words NEW and MEXICO or the word ARIZONA. Again, pages besides those of interest will be listed. Finally, to find Web pages that deal with universities in Mexico (and not New Mexico), we might first look for pages matching MEXICO *AND* UNIVERSITIES, but because the results of this search will include pages about universities in New Mexico, as well as universities in Mexico, it might be better to search for pages matching (MEXICO *AND* UNIVERSITIES) *NOT* NEW. The results of this search include pages that contain both the words MEXICO and UNIVERSITIES but do not contain the word NEW. (In Google, and many other search engines, the word "NOT" is replaced by the symbol "-". In Google, the terms used for this last search would be MEXICO UNIVERSITIES - NEW.) ■

1.2.5 Logic Puzzles

Puzzles that can be solved using logical reasoning are known as **logic puzzles**. Solving logic puzzles is an excellent way to practice working with the rules of logic. Also, computer programs designed to carry out logical reasoning often use well-known logic puzzles to illustrate their capabilities. Many people enjoy solving logic puzzles, published in periodicals, books, and on theWeb, as a recreational activity.

We will discuss two logic puzzles here.We begin with a puzzle originally posed by Raymond Smullyan, a master of logic puzzles, who has published more than a dozen books containing challenging puzzles that involve logical reasoning.

 EXAMPLE. 7

In [2] Smullyan posed many puzzles about an island that has two kinds of

inhabitants, knights, who always tell the truth, and their opposites, knaves, who always lie. You encounter two people A and B. What are A and B if A says "B is a knight" and B says "The two of us are opposite types?"

Solution: Let p and q be the statements that A is a knight and B is a knight, respectively, so that $\neg p$ and $\neg q$ are the statements that A is a knave and B is a knave, respectively. We first consider the possibility that A is a knight; this is the statement that p is true. If A is a knight, then he is telling the truth when he says that B is a knight, so that q is true, and A and B are the same type. However, if B is a knight, then B's statement that A and B are of opposite types, the statement $(p \wedge \neg q) \vee (\neg p \wedge q)$, would have to be true, which it is not, because A and B are both knights. Consequently, we can conclude that A is not a knight, that is, that p is false.

If A is a knave, then because everything a knave says is false, A's statement that B is a knight, that is, that q is true, is a lie. This means that q is false and B is also a knave. Furthermore, if B is a knave, then B's statement that A and B are opposite types is a lie, which is consistent with both A and B being knaves. We can conclude that both A and B are knaves. ∎

Next, we pose a puzzle known as the **muddy children puzzle** for the case of two children.

EXAMPLE. 8

A father tells his two children, a boy and a girl, to play in their backyard without getting dirty. However, while playing, both children get mud on their foreheads. When the children stop playing, the father says "At least one of you has a muddy forehead," and then asks the children to answer "Yes" or "No" to the question: "Do you know whether you have a muddy forehead?" The father asks this question twice. What will the children answer each time this question is asked, assuming that a child can see whether his or her sibling has a muddy forehead, but cannot see his or her own forehead? Assume that both children are honest and that the children answer each question simultaneously.

Solution: Let s be the statement that the son has a muddy forehead and let d be the statement that the daughter has a muddy forehead. When the father says that at least one of the two children has a muddy forehead, he is stating that the disjunction $s \vee d$ is true. Both children will answer "No" the first time the question is asked because each sees mud on the other child's forehead. That is, the son knows that d is true, but does not know whether s is true, and the daughter knows that s is true, but does not know whether

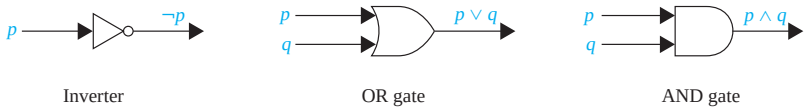


Figure 1.1: Basic logic gates.

d is true.

After the son has answered "No" to the first question, the daughter can determine that d must be true. This follows because when the first question is asked, the son knows that $s \vee d$ is true, but cannot determine whether s is true. Using this information, the daughter can conclude that d must be true, for if d were false, the son could have reasoned that because $s \vee d$ is true, then s must be true, and he would have answered "Yes" to the first question. The son can reason in a similar way to determine that s must be true. It follows that both children answer "Yes" the second time the question is asked. ∎

1.2.6 Logic Circuits

Propositional logic can be applied to the design of computer hardware. This was first observed in 1938 by Claude Shannon in his MIT master's thesis.

A **logic circuit** (or **digital circuit**) receives input signals $p_1, p_2, \ldots, p_n$, each a bit [either 0 (off) or 1 (on)], and produces output signals $s_1, s_2, \ldots, s_n$, each a bit. In this section we will restrict our attention to logic circuits with a single output signal; in general, digital circuits may have multiple outputs.

Complicated digital circuits can be constructed from three basic circuits, called gates, shown in Figure 1.1. The **inverter**, or **NOT** gate, takes an input bit p, and produces as output $\neg p$. The **OR gate** takes two input signals p and q, each a bit, and produces as output the signal $p \vee q$. Finally, the **AND gate** takes two input signals p and q, each a bit, and produces as output the signal $p \wedge q$.

Given a circuit built from the basic logic gates and the inputs to the circuit, we determine the output by tracing through the circuit, as Example 9 shows.

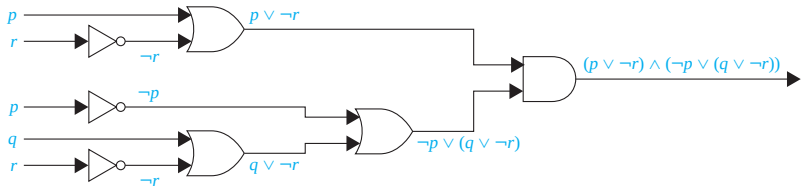


Figure 1.2: The circuit for $(p \vee \neg r) \wedge (\neg p \vee (q \vee \neg r))$.

EXAMPLE. 9

Build a digital circuit that produces the output $(p \vee \neq r) \wedge (\neq p \vee (q \vee \neq r))$ when given input bits p, q, and r.

Solution: To construct the desired circuit, we build separate circuits for $p \vee \neg r$ and for $\neg p \vee (q \vee \neg r)$ and combine them using an AND gate. To construct a circuit for $p \vee \neg r$, we use an inverter to produce $\neg r$ from the input r. Then, we use an OR gate to combine p and $\neg r$. To build a circuit for $\neg p \vee (q \vee \neg r)$, we first use an inverter to obtain $\neg r$. Then we use an OR gate with inputs q and $\neg r$ to obtain $q \vee \neg r$. Finally, we use another inverter and an OR gate to get $\neg p \vee (q \vee \neg r)$ from the inputs p and $q \vee \neg r$.

To complete the construction, we employ a final AND gate, with inputs $p \vee \neg r$ and $\neg p \vee (q \vee \neg r)$. The resulting circuit is displayed in Figure 1.2. ∎

1.3 Propositional Equivalences

1.3.1 Introduction

An important type of step used in a mathematical argument is the replacement of a statement with another statement with the same truth value. Because of this, methods that produce propositions with the same truth value as a given compound proposition are used extensively in the construction of mathematical arguments. Note that we will use the term "compound proposition" to refer to an expression formed from propositional variables using logical operators, such as $p \wedge q$.

Table 1.10: Examples of a Tautology and a Contradiction.

p	$\neg p$	$p \vee \neg q$	$p \wedge \neg p$
T	F	T	F
F	T	T	F

We begin our discussion with a classification of compound propositions according to their possible truth values.

Definition 1.3.1 *A compound proposition that is always true, no matter what the truth values of the propositional variables that occur in it, is called a* ***tautology****.A compound proposition that is always false is called a* ***contradiction****. A compound proposition that is neither a tautology nor a contradiction is called a* ***contingency****.*

Tautologies and contradictions are often important in mathematical reasoning. Example 1 illustrates these types of compound propositions.

EXAMPLE. 1

We can construct examples of tautologies and contradictions using just one propositional variable. Consider the truth tables of $p \vee \neg p$ and $p \wedge \neg p$, shown in Table 1.10. Because $p \vee \neg p$ is always true, it is a tautology. Because $p \wedge \neg p$ is always false, it is a contradiction. ∎

1.3.2 Logical Equivalences

Compound propositions that have the same truth values in all possible cases are called **logically equivalent**. We can also define this notion as follows.

Definition 1.3.2 *The compound propositions p and q are called* ***logically equivalent*** *if $p \leftrightarrow q$ is a tautology. The notation $p \equiv q$ denotes that p and q are logically equivalent.*

Remark! The symbol $\equiv$ is not a logical connective, and $p \equiv q$ is not a compound proposition but rather is the statement that $p \leftrightarrow q$ is a tautology. The symbol$\Leftrightarrow$ is sometimes used instead of $\equiv$ to denote logical equivalence.

Table 1.11: De Morgan's Laws.

$\neg(p \wedge q) \equiv \neg p \vee \neg q$
$\neg(p \vee q) \equiv \neg p \wedge \neg q$

One way to determine whether two compound propositions are equivalent is to use a truth table. In particular, the compound propositions p and q are equivalent if and only if the columns giving their truth values agree.

Example 2 illustrates this method to establish an extremely important and useful logical equivalence, namely, that of $\neg(p \vee q)$ with $\neg p \wedge \neg q$. This equivalence is one of the two **De Morgan laws**, shown in Table 1.11, named after the English mathematician Augustus De Morgan, of the mid-nineteenth century.

Table 1.12: Truth Tables for $\neg(p \vee q)$ and $\neg p \wedge \neg q$

p	q	$p \vee q$	$\neg(p \vee q)$	$\neg p$	$\neg q$	$\neg p \wedge q$
T	T	T	F	F	F	F
T	F	T	F	F	T	F
F	T	T	F	T	F	F
F	F	F	T	T	T	T

EXAMPLE. 2

Show that $\neg(p \vee q)$ and $\neg p \wedge \neg q$ are logically equivalent.

Table 1.13: Truth Tables for $\neg p \vee q$ and $p \rightarrow q$

p	q	$\neg q$	$p \vee \neg q$	$p \wedge q$	$(p \vee \neg q) \rightarrow (p \wedge q)$
T	T	F	T	T	T
T	F	T	T	F	F
F	T	F	F	F	T
F	F	T	T	F	F

Solution: The truth tables for these compound propositions are displayed in Table 1.12. Because the truth values of the compound propositions $\neg(p \vee q)$ and $\neg p \wedge \neg q$ agree for all possible combinations of the truth values of p and q, it follows that $\neg(p \vee q) \leftrightarrow (\neg p \wedge \neg q)$ is a tautology and that these compound propositions are logically equivalent. ∎

EXAMPLE. 3

Show that $p \rightarrow q$ and $\neg p \vee q$ are logically equivalent.

Solution: We construct the truth table for these compound propositions in Table 1.13. Because the truth values of $\neg p \vee q$ and $p \rightarrow q$ agree, they are logically equivalent. ∎

We will now establish a logical equivalence of two compound propositions involving three different propositional variables p, q, and r. To use a truth table to establish such a logical equivalence, we need eight rows, one for each possible combination of truth values of these three variables. We symbolically represent these combinations by listing the truth values of p, q, and r, respectively. These eight combinations of truth values are TTT, TTF, TFT, TFF, FTT, FTF, FFT, and FFF; we use this order when we display the rows of the truth table. Note that we need to double the number of rows in the truth tables we use to showthat compound propositions are equivalent for each additional propositional variable, so that 16 rows are needed to establish the logical

Table 1.14: A Demonstration That $p \vee (q \wedge r)$ and $(p \vee q) \wedge (p \vee r)$ Are Logically Equivalent.

p	q	r	$p \wedge q$	$p \vee (q \wedge r)$	$p \vee q$	$p \vee r)$	$(p \vee q) \wedge (p \vee r)$
T	T	T	T	T	T	T	T
T	T	F	F	T	T	T	T
T	F	T	F	T	T	T	T
T	F	F	F	T	T	T	T
F	T	T	T	T	T	T	T
F	T	F	F	F	T	F	F
F	F	T	F	F	F	T	F
F	F	F	F	F	F	F	F

equivalence of two compound propositions involving four propositional variables, and so on. In general, 2^n rows are required if a compound proposition involves n propositional variables.

EXAMPLE. 4

Show that $p \vee (q \wedge r)$ and $(p \vee q) \wedge (p \vee r)$ are logically equivalent. This is the distributive law of disjunction over conjunction.

Solution: We construct the truth table for these compound propositions in Table 1.14. Because the truth values of $p \vee (q \wedge r)$ and $(p \vee q) \wedge (p \vee r)$ agree, these compound propositions are logically equivalent. ∎

Table 1.15 contains some important equivalences. In these equivalences, **T** denotes the compound proposition that is always true and **F** denotes the compound proposition that is always false. We also display some useful equivalences for compound propositions involving conditional statements and biconditional statements in Table 1.16 and 1.17, respectively. The reader is asked to verify the equivalences in Tables 1.15-1.17 in the exercises.

The associative law for disjunction shows that the expression $p \vee q \vee r$

Table 1.15: Logical Equivalences.

Equivalence	**Name**
$p \wedge \mathbf{T} \equiv p$ $p \vee \mathbf{F} \equiv p$	Identity laws
$p \vee \mathbf{T} \equiv \mathbf{T}$ $p \wedge \mathbf{F} \equiv \mathbf{F}$	Domination laws
$p \vee p \equiv p$ $p \wedge p \equiv p$	Idempotent laws
$\neg(\neg p) \equiv p$	Double negation law
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	Commutative laws
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associative laws
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Distributive laws
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	De Morgan's laws
$p \vee (p \wedge q) \equiv p$ $p \wedge (q \vee q) \equiv p$	Absorption laws
$p \vee \neg p \equiv \mathbf{T}$ $p \wedge \neg p \equiv \mathbf{F}$	Negation laws

Table 1.16: Logical Equivalences Involving Conditional Statements.

$p \to q \equiv \neg p \vee q$
$p \to q \equiv \neg q \to \neg p$
$p \vee q \equiv \neg p \to q$
$p \wedge q \equiv \neg(p \to \neg q)$
$\neg(p \to q) \equiv p \wedge \neg q$
$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$
$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$
$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$
$(p \to r) \vee (q \to r) \equiv (p \wedge q) \to r$

Table 1.17: Logical Equivalences Involving Biconditional Statements.

$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$
$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$
$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
$\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$

is well defined, in the sense that it does not matter whether we first take the disjunction of p with q and then the disjunction of $p \vee q$ with r, or if we first take the disjunction of q and r and then take the disjunction ofp with $q \vee r$. Similarly, the expression $p \wedge q \wedge r$ is well defined. By extending this reasoning, it follows that $p_1 \vee p_2 \vee \ldots \vee p_n$ and $p_1 \wedge p_2 \wedge \ldots \wedge p_n$ are well defined whenever $p_1, p_2, \ldots, p_n$ are propositions.

Furthermore, note that De Morgan's laws extend to

$$\neg(p_1 \vee p_2 \vee \ldots \vee p_n) \equiv (\neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n)$$

and

$$\neg\big(p_1 \wedge p_2 \wedge \ldots \wedge p_n\big) \equiv \big(\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n\big).$$

We will sometimes use the notation $\bigvee_{j=1}^{n} p_j$ for $p_1 \vee p_2 \vee \ldots \vee p_n$ and $\bigwedge_{j=1}^{n} p_j$ for $p_1 \wedge p_2 \wedge \ldots \wedge p_n$. Using this notation, the extended version of De Morgan's laws can be written concisely as $\neg \bigvee_{j=1}^{n} p_j \equiv \bigwedge_{j=1}^{n} \neg p_j$ and $\neg \bigwedge_{j=1}^{n} p_j \equiv \bigvee_{j=1}^{n} \neg p_j$.

1.3.3 Using De Morgan's Laws

The two logical equivalences known as De Morgan's laws are particularly important. They tell us how to negate conjunctions and how to negate disjunctions. In particular, the equivalence $\neg(p \vee q) \equiv \neg p \wedge \neg q$

tells us that the negation of a disjunction is formed by taking the conjunction of the negations of the component propositions. Similarly, the equivalence $\neg(p \wedge q) \equiv \neg p \vee \neg q$ tells us that the negation of a conjunction is formed by taking the disjunction of the negations of the component propositions. Example 5 illustrates the use of De Morgan's laws.

EXAMPLE. 5

Use De Morgan's laws to express the negations of "Miguel has a cellphone and he has a laptop computer" and "Heather will go to the concert or Steve will go to the concert."

Solution: Let p be "Miguel has a cellphone" and q be "Miguel has a laptop computer". Then "Miguel has a cellphone and he has a laptop computer" can be represented by $p \wedge q$. By the first of De Morgan's laws, $\neg(p \wedge q)$ is equivalent to $\neg p \vee \neg q$. Consequently, we can express the negation of our original statement as "Miguel does not have a cellphone or he does not have a laptop computer."

Let r be "Heather will go to the concert" and s be "Steve will go to the concert." Then "Heather will go to the concert or Steve will go to the concert" can be represented by $r \vee s$. By the second of De Morgan's laws, $\neg(r \vee s)$ is equivalent to $\neg r \wedge \neg s$. Consequently, we can express the negation of our original statement as "Heather will not go to the concert and Steve will not go to the concert."

1.3.4 Constructing New Logical Equivalences

The logical equivalences in Table 1.15, as well as any others that have been established (such as those shown in Table 1.16 and Table 1.17), can be used to construct additional logical equivalences. The reason for this is that a proposition in a compound proposition can be replaced by a compound proposition that is logically equivalent to it without changing the truth value of the original compound proposition. This technique is illustrated in Examples 6–8, where we also use the fact that if p and q are logically equivalent and q and r are logically equivalent, then p and r are logically equivalent.

EXAMPLE. 6

Show that $\neg(p \to q)$ and $p \wedge \neg q$ are logically equivalent.

Solution: We could use a truth table to show that these compound propositions are equivalent (similar to what we did in Example 4). Indeed, it would not be hard to do so. However, we want to illustrate how to use logical identities that we already know to establish new logical identities, something that is of practical importance for establishing equivalences of compound propositions with a large number of variables. So, we will establish this equivalence by developing a series of logical equivalences, using one of the equivalences in Table 1.15 at a time, starting with $\neg(p \to q)$ and ending with $p \wedge \neg q$.We have the following equivalences.

$\neg(p \to q)$	$\equiv \neg(\neg p \vee q)$	by Example 3
	$\equiv \neg(\neg p) \wedge \neg q$	by the section De Morgan law
	$\equiv p \wedge \neg q$	by the double negation law

■

EXAMPLE. 7

Show that $\neg(p \vee (\neg p \wedge q))$ and $\neg p \wedge \neg q$ are logically equivalent by developing a series of logical equivalences.

Solution: We will use one of the equivalences in Table 1.15 at a time, starting with $\neg(p \vee (\neg p \wedge q))$ and ending with $\neg p \wedge \neg q$. (*Note*: we could also easily establish this equivalence using a truth table.)We have the following equivalences.

$\neg(p \vee (\neg p \wedge q))$	$\equiv \neg p \wedge \neg(\neg p \wedge q)$	by the second De Morgan law
	$\equiv \neg p \wedge [\neg(\neg p) \vee \neg q]$	by the first De Morgan law
	$\equiv \neg p \wedge (p \vee \neg q)$	by the double negation law
	$\equiv (\neg p \wedge p) \vee (\neg p \wedge \neg q)$	by the second distributive law
	$\equiv \mathbf{F} \vee (\neg p \wedge \neg q)$	because $\neg p \wedge p \equiv \mathbf{F}$
	$\equiv (\neg p \wedge \neg q) \vee \mathbf{F}$	by the commutative law for disjunction
	$\equiv \neg p \wedge \neg q$	by the identity law for$\mathbf{F}$

Consequently $\neg(p \vee (\neg p \wedge q))$ and $\neg p \wedge \neg q$ are logically equivalent. ■

EXAMPLE. 8

Show that $(p \wedge q) \rightarrow (p \vee q)$ is a tautology.

Solution: To show that this statement is a tautology, we will use logical equivalences to demonstrate that it is logically equivalent to $\mathbf{T}$.

$(p \wedge q) \rightarrow (p \vee q)$	$\equiv \neg(p \wedge q) \vee (p \vee q)$	by Example 3
	$\equiv (\neg p \vee \neg q) \vee (p \vee q)$	by the first De Morgan law
	$\equiv (\neg p \vee p) \vee (\neg q \vee q)$	by the associative and commutative laws for disjunction
	$\equiv \mathbf{T} \vee \mathbf{T}$	by Example 1 and the commutative law for disjunction
	$\equiv \mathbf{T}$	by the domination law ∎

1.3.5 Propositional Satisfiability

Acompound proposition is **satisfiable** if there is an assignment of truth values to its variables that makes it true. When no such assignments exists, that is, when the compound proposition is false for all assignments of truth values to its variables, the compound proposition is unsatisfiable. Note that a compound proposition is unsatisfiable if and only if its negation is true for all assignments of truth values to the variables, that is, if and only if its negation is a tautology.

When we find a particular assignment of truth values that makes a compound proposition true, we have shown that it is satisfiable; such an assignment is called a solution of this particular satisfiability problem. However, to show that a compound proposition is unsatisfiable, we need to show that *every* assignment of truth values to its variables makes it false. Although we can always use a truth table to determine whether a compound proposition is satisfiable, it is often more efficient not to, as Example 9 demonstrates.

EXAMPLE. 9

Determine whether each of the compound propositions $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$, $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$, and $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ is satisfiable.

Solution: Instead of using truth table to solve this problem, we will reason about truth values. Note that $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$ is true when the three variable p, q, and r have the same truth value. Hence, it is satisfiable as there is at least one assignment of truth values for p, q, and r that makes it true. Similarly, note that $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ is true when at least one of p, q, and r is true and at least one is false. Hence, $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ is satisfiable, as there is at least one assignment of truth values for p, q, and r that makes it true.

Finally, note that for $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ to be true, $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$ and $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ must both be true. For the first to be true, the three variables must have the same truth values, and for the second to be true, at least one of three variables must be true and at least one must be false. However, these conditions are contradictory. From these observations we conclude that no assignment of truth values to p, q, and r makes $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ true. Hence, it is unsatisfiable. ∎

1.3.6 Applications of Satisfiability

Many problems, in diverse areas such as robotics, software testing, computer-aided design, machine vision, integrated circuit design, computer networking, and genetics, can be modeled in terms of propositional satisfiability. Although most of these applications are beyond the scope of this book, we will study one application here. In particular, we will show how to use propositional satisfiability to model Sudoku puzzles.

SUDOKU

A Sudoku puzzle is represented by a 9×9 grid made up of nine 3×3 subgrids, known as blocks, as shown in Figure 1.3. For each puzzle, some of the 81 cells, called **givens**, are assigned one of the numbers $1, 2, \ldots, 9$, and the other cells are blank. The puzzle is solved by assigning a number to each blank cell so that every row, every column, and every one of the nine 3×3 blocks contains each of the nine possible numbers. Note that instead of using a 9×9 grid, Sudoku puzzles can

	2	9				4		
			5			1		
	4							
				4	2			
6							7	
5								
7			3					5
	1			9				
							6	

Figure 1.3: A 9×9 Sudoku puzzle.

be based on $n^2 \times n^2$ grids, for any positive integer n, with the $n^2 \times n^2$ grid made up of n^2 $n \times n$ subgrids.

The popularity of Sudoku dates back to the 1980s when it was introduced in Japan. It took 20 years for Sudoku to spread to rest of the world, but by 2005, Sudoku puzzles were a worldwide craze. The name Sudoku is short for the Japanese *suuji wa dokushin ni kagiru*, which means "the digits must remain single". The modern game of Sudoku was apparently designed in the late 1970s by an American puzzle designer. The basic ideas of Sudoku date back even further; puzzles printed in French newspapers in the 1890s were quite similar, but not identical, to modern Sudoku.

Sudoku puzzles designed for entertainment have two additional important properties. First, they have exactly one solution. Second, they can be solved using reasoning alone, that is, without resorting to searching all possible assignments of numbers to the cells. As a Sudoku puzzle is solved, entries in blank cells are successively determined by already known values. For instance, in the grid in Figure 1.3, the number 4 must appear in exactly one cell in the second row. How can we determine which of the seven blank cells it must appear? First, we observe that 4 cannot appear in one of the first three cells or in one of the last three cells of this row, because it already appears in another cell in the block each of these cells is in.We can also see that 4 cannot appear in the fifth cell in this row, as it already appears in the fifth column in the fourth row. This means that 4 must appear in the sixth cell of the

second row.

Many strategies based on logic and mathematics have been devised for solving Sudoku puzzles. Here, we discuss one of the ways that have been developed for solving Sudoku puzzles with the aid of a computer, which depends on modeling the puzzle as a propositional satisfiability problem. Using the model we describe, particular Sudoku puzzles can be solved using software developed to solve satisfiability problems. Currently, Sudoku puzzles can be solved in less than 10 milliseconds this way. It should be noted that there are many other approaches for solving Sudoku puzzles via computers using other techniques.

To encode a Sudoku puzzle, let *p(i, j, n)* denote the proposition that is true when the number n is in the cell in the ith row and j th column. There are $9 \times 9 \times 9 = 729$ such propositions, as i, j , and n all range from 1 to 9. For example, for the puzzle in Figure 1.3, the number 6 is given as the value in the fifth row and first column. Hence, we see that $p(5, 1, 6)$ is true, but $p(5, j, 6)$ is false for $j = 2, 3, \ldots, 9$.

Given a particular Sudoku puzzle, we begin by encoding each of the given values. Then, we construct compound propositions that assert that every row contains every number, every column contains every number, every 3×3 block contains every number, and each cell contains no more than one number. It follows, as the reader should verify, that the Sudoku puzzle is solved by finding an assignment of truth values to the 729 propositions *p(i, j, n)* with i, j, and n each ranging from 1 to 9 that makes the conjunction of all these compound propositions true. After listing these assertions, we will explain how to construct the assertion that every row contains every integer from 1 to 9.We will leave the construction of the other assertions that every column contains every number and each of the nine 3×3 blocks contains every number to the exercises.

- For each cell with a given value, we assert *p(i, j, n)* when the cell in row i and column j has the given value n.

- We assert that every row contains every number:

$$\bigwedge_{i=1}^{9} \bigwedge_{n=1}^{9} \bigvee_{j=1}^{9} p(i, j, n)$$

- We assert that every column contains every number:

$$\bigwedge_{j=1}^{9}\bigwedge_{n=1}^{9}\bigvee_{i=1}^{9} p(i,j,n)$$

- We assert that each of the nine 3×3 blocks contains every number:

$$\bigwedge_{r=0}^{2}\bigwedge_{s=0}^{2}\bigwedge_{n=1}^{9}\bigvee_{i=1}^{3}\bigvee_{j=1}^{3} p(3r+i,3s+j,n)$$

- To assert that no cell contains more than one number, we take the conjunction over all values of n, n', i, and j where each variable ranges from 1 to 9 and $n \neq n'$ of $p(i,j,n) \rightarrow \neg p(i,j,n')$

We now explain how to construct the assertion that every row contains every number. First, to assert that row i contains the number n, we form $\bigvee_{j=1}^{9} p(i,j,n)$. To assert that row i contains all n numbers, we form the conjunction of these disjunctions over all nine possible values of n, giving us $\bigwedge_{n=1}^{9}\bigvee_{j=1}^{9} p(i,j,n)$ Finally, to assert that every row contains every number, we take the conjunction of $\bigwedge_{n=1}^{9}\bigvee_{j=1}^{9} p(i,j,n)$ over all nine rows. This gives us $\bigwedge_{i=1}^{9}\bigwedge_{n=1}^{9}\bigvee_{j=1}^{9} p(i,j,n)$.

Given a particular Sudoku puzzle, to solve this puzzle we can find a solution to the satisfiability problems that asks for a set of truth values for the 729 variables *p(i, j, n)* that makes the conjunction of all the listed assertions true.

1.3.7 Solving Satisfiability Problems

truth table can be used to determine whether a compound proposition is satisfiable, or equivalently, whether its negation is a tautology. This can be done by hand for a compound proposition with a small number of variables, but when the number of variables grows, this becomes impractical. For instance, there are $2^{20} = 1,048,576$ rows in the truth table for a compound proposition with 20 variables. Clearly, you need a computer to help you determine, in this way, whether a compound proposition in 20 variables is satisfiable.

When many applications are modeled, questions concerning the satisfiability of compound propositions with hundreds, thousands, or millions of variables arise. Note, for example, that when there are 1000 variables, checking every one of the 2^{1000} (a number with more than 300 decimal digits) possible combinations of truth values of the variables in a compound proposition cannot be done by a computer in even trillions of years. No procedure is known that a computer can follow to determine in a reasonable amount of time whether an arbitrary compound proposition in such a large number of variables is satisfiable. However, progress has been made developing methods for solving the satisfiability problem for the particular types of compound propositions that arise in practical applications, such as for the solution of Sudoku puzzles. Many computer programs have been developed for solving satisfiability problems which have practical use. In our discussion of the subject of algorithms, we will discuss this question further. In particular, we will explain the important role the propositional satisfiability problem plays in the study of the complexity of algorithms.

1.4 Predicates and Quantifiers

1.4.1 Introduction

Propositional logic, studied in previous sections, cannot adequately express the meaning of all statements in mathematics and in natural language. For example, suppose that we know that

"Every computer connected to the university network is functioning properly."

No rules of propositional logic allow us to conclude the truth of the statement

"MATH3 is functioning properly,"

whereMATH3 is one of the computers connected to the university network. Likewise, we cannot use the rules of propositional logic to conclude from the statement

"CS2 is under attack by an intruder,"

where CS2 is a computer on the university network, to conclude the truth of

"There is a computer on the university network that is under attack by an intruder."

In this section we will introduce a more powerful type of logic called predicate logic. We will see how predicate logic can be used to express the meaning of a wide range of statements in mathematics and computer science in ways that permit us to reason and explore relationships between objects. To understand predicate logic, we first need to introduce the concept of a predicate. Afterward, we will introduce the notion of quantifiers, which enable us to reason with statements that assert that a certain property holds for all objects of a certain type and with statements that assert the existence of an object with a particular property.

1.4.2 Predicates

Statements involving variables, such as

$$\text{“}x > 3\text{”}, \text{“}x = y + 3\text{”}, \text{“}x + y = z\text{”},$$

and

"computer x is under attack by an intruder,"

and

"computer x is functioning properly,"

are often found in mathematical assertions, in computer programs, and in system specifications. These statements are neither true nor false when the values of the variables are not specified. In this section, we will discuss the ways that propositions can be produced from such statements.

The statement "x is greater than 3" has two parts. The first part, the variable x, is the subject of the statement. The second part — the **predicate**, "is greater than 3" —refers to a property that the subject of the statement can have.We can denote the statement "x is greater than 3" by *P(x)*, where P denotes the predicate "is greater than 3" and x is the variable. The statement *P(x)* is also said to be the value of the **propositional function** P at x. Once a value has been assigned to the variable x, the statement *P(x)* becomes a proposition and has a truth value. Consider Examples 1 and 2.

EXAMPLE. 1

Let *P(x)* denote the statement "$x > 3$". What are the truth values of *P(4)* and *P(2)*?

Solution: We obtain the statement *P(4)* by setting $x = 4$ in the statement "$x > 3$". Hence, *P(4)*, which is the statement "$4 > 3$", is true. However, *P(2)*, which is the statement "$2 > 3$", is false. ∎

EXAMPLE. 2

Let *A(x)* denote the statement "Computer x is under attack by an intruder". Suppose that of the computers on campus, only CS2 and MATH1 are currently under attack by intruders. What are truth values of A(CS1), A(CS2), and A(MATH1)?

Solution: We obtain the statement A(CS1) by setting $x = CS1$ in the statement "Computer x is under attack by an intruder". Because CS1 is not on the list of computers currently under attack, we conclude that A(CS1) is false. Similarly, because CS2 and MATH1 are on the list of computers under attack, we know that A(CS2) and A(MATH1) are true. ∎

We can also have statements that involve more than one variable. For instance, consider the statement "x = y + 3". We can denote this statement by *Q(x, y)*, where x and y are variables and Q is the predicate. When values are assigned to the variables x and y, the statement *Q(x, y)* has a truth value.

EXAMPLE. 3

Let *Q(x, y)* denote the statement "x = y + 3". What are the truth values of the propositions *Q(1, 2)* and *Q(3, 0)*?

Solution: To obtain *Q(1, 2)*, set $x = 1$ and $y = 2$ in the statement *Q(x, y)*. Hence, *Q(1, 2)* is the statement "$1 = 2 + 3$", which is false. The statement *Q(3, 0)* is the proposition "$3 = 0 + 3$", which is true. ∎

EXAMPLE. 4

Let *A(c, n)* denote the statement "Computer c is connected to network n", where c is a variable representing a computer and n is a variable representing a network. Suppose that the computer MATH1 is connected to network CAMPUS2, but not to network CAMPUS1. What are the values of A(MATH1, CAMPUS1) and A(MATH1, CAMPUS2)?

Solution: Because MATH1 is not connected to the CAMPUS1 network, we see that A(MATH1, CAMPUS1) is false. However, because MATH1 is connected to the CAMPUS2 network, we see that A(MATH1, CAMPUS2) is true. ∎

Similarly, we can let *R(x, y, z)* denote the statement"x + y = z". When values are assigned to the variables *x, y,* and *z*, this statement has a truth value.

EXAMPLE. 5

What are the truth values of the propositions *R(1, 2, 3)* and *R(0, 0, 1)*?

Solution: The proposition *R(1, 2, 3)* is obtained by setting $x = 1, y = 2$, and $z = 3$ in the statement *R(x, y, z)*. We see that *R(1, 2, 3)* is the statement "$1 + 2 = 3$", which is true. Also note that *R(0, 0, 1)*, which is the statement "$0 + 0 = 1$", is false. ∎

In general, a statement involving the n variables $x_1, x_2, \ldots, x_n$ can be denoted by

$$P(x_1, x_2, \ldots, x_n)$$

. A statement of the form $P(x_1, x_2, \ldots, x_n)$ is the value of the **propositional function** P at the n-tuple $(x_1, x_2, \ldots, x_n)$, and P is also called an n-**place predicate** or a n-**ary predicate**.

Propositional functions occur in computer programs, as Example 6 demonstrates.

EXAMPLE. 6

Consider the statement

if $x > 0$ **then** $x := x + 1$.

When this statement is encountered in a program, the value of the variable x at that point in the execution of the program is inserted into *P(x)*, which is "$x > 0$". If *P(x)* is true for this value of x, the assignment statement $x := x + 1$ is executed, so the value of x is increased by 1. If *P(x)* is false for this value of x, the assignment statement is not executed, so the value of x is not changed. ∎

1.4.3 Quantifiers

When the variables in a propositional function are assigned values, the resulting statement becomes a proposition with a certain truth value. However, there is another importantway, called **quantification**, to create a proposition from a propositional function. Quantification expresses the extent to which a predicate is true over a range of elements. In English, the words *all, some, many, none*, and *few* are used in quantifications. We will focus on two types of quantification here: universal quantification, which tells us that a predicate is true for every element under consideration, and existential quantification, which tells us that there is one or more element under consideration for which the predicate is true. The area of logic that deals with predicates and quantifiers is called the **predicate calculus**.

THE UNIVERSAL QUANTIFIER

Many mathematical statements assert that a property is true for all values of a variable in a particular domain, called the **domain of discourse** (or the **universe of discourse**), often just referred to as the **domain**. Such a statement is expressed using universal quantification. The universal quantification of *P(x)* for a particular domain is the proposition that asserts that *P(x)* is true for all values of x in this domain. Note that the domain specifies the possible values of the variable x. The meaning of the universal quantification of *P(x)* changes when we change the domain. The domain must always be specified when a universal quantifier is used; without it, the universal quantification of a statement is not defined.

Table 1.18: Quantifiers.

Statement	When True?	When False?
$\forall x P(x)$	$P(x)$ is true for every x.	There is an x for which $P(x)$ is false.
$\exists x P(x)$	There is an x for which $P(x)$ is true.	$P(x)$ is false for every x.

Definition 1.4.1 *The **universal quantification** of $P(x)$ is the statement*

"$P(x)$ for all values of x in the domain."

*The notation $\forall x P(x)$ denotes the universal quantification of $P(x)$. Here $\forall$ is called the **universal quantifier**. We read $\forall x P(x)$ as "for all $x P(x)$" or "for every $x P(x)$." An element for which $P(x)$ is false is called a **counterexample** of $\forall x P(x)$.*

The meaning of the universal quantifier is summarized in the first row of Table 1.18. We illustrate the use of the universal quantifier in Examples 7–12.

EXAMPLE. 7

Let $P(x)$ be the statement "$x + 1 > x$". What is the truth value of the quantification $\forall x P(x)$, where the domain consists of all real numbers?

Solution: Because $P(x)$ is true for all real numbers x, the quantification

$$\forall x P(x)$$

is true. ∎

Remark! Generally, an implicit assumption is made that all domains of discourse for quantifiers are nonempty. Note that if the domain is empty, then $\forall x P(x)$ is true for any propositional function $P(x)$ because there are no elements x in the domain for which $P(x)$ is false.

Besides "for all" and "for every," universal quantification can be expressed in many other ways, including "all of", "for each", "given any", "for arbitrary", "for each", and "for any".

> **Remark!** It is best to avoid using "for any x" because it is often ambiguous as to whether "any" means "every" or "some". In some cases, "any" is unambiguous, such as when it is used in negatives, for example, "there is not any reason to avoid studying".

A statement $\forall x P(x)$ is false, where *P(x)* is a propositional function, if and only if *P(x)* is not always true when x is in the domain. One way to show that *P(x)* is not always true when x is in the domain is to find a counterexample to the statement $\forall x P(x)$. Note that a single counterexample is all we need to establish that $\forall x P(x)$ is false. Example 8 illustrates how counterexamples are used.

 EXAMPLE. 8

Let *Q(x)* be the statement "$x < 2$". What is the truth value of the quantification $\forall x Q(x)$, where the domain consists of all real numbers?

Solution: *Q(x)* is not true for every real number x, because, for instance, *Q(3)* is false. That is, $x = 3$ is a counterexample for the statement $\forall x Q(x)$. Thus $\forall x Q(x)$ is false. ∎

 EXAMPLE. 9

Suppose that *P(x)* is "$x^2 > 0$". To show that the statement $\forall x P(x)$ is false where the universe of discourse consists of all integers, we give a counterexample. We see that $x = 0$ is a counterexample because $x^2 = 0$ when $x = 0$, so that x^2 is not greater than 0 when $x = 0$. ∎

Looking for counterexamples to universally quantified statements is an important activity in the study of mathematics, as we will see in subsequent sections of this book.

When all the elements in the domain can be listed—say, $x_1, x_2, \ldots, x_n$ — it follows that the universal quantification $\forall x P(x)$ is the same as the

conjunction

$$P(x_1) \wedge P(x_2) \wedge \ldots \wedge P(x_n),$$

because this conjunction is true if and only if $P(x_1), P(x_2), \ldots, P(x_n)$ are all true.

EXAMPLE. 10

What is the truth value of $\forall x P(x)$, where *P(x)* is the statement "$x_2 < 10$" and the domain consists of the positive integers not exceeding 4?

Solution: The statement $\forall x P(x)$ is the same as the conjunction $P(1) \wedge P(2) \wedge P(3) \wedge P(4)$, because the domain consists of the integers 1, 2, 3, and 4. Because *P(4)*, which is the statement "$42 < 10$," is false, it follows that $\forall x P(x)$ is false. ■

EXAMPLE. 11

What does the statement $\forall x N(x)$ mean if *N(x)* is "Computer x is connected to the network" and the domain consists of all computers on campus?

Solution: The statement $\forall x N(x)$ means that for every computer x on campus, that computer x is connected to the network. This statement can be expressed in English as "Every computer on campus is connected to the network". ■

As we have pointed out, specifying the domain is mandatory when quantifiers are used. The truth value of a quantified statement often depends on which elements are in this domain, as Example 12 shows.

EXAMPLE. 12

What is the truth value of $\forall x(x^2 \geq x)$ if the domain consists of all real numbers? What is the truth value of this statement if the domain consists of all integers?

Solution: The universal quantification $\forall x(x^2 \geq x)$, where the domain consists of all real numbers, is false. For example, $(\frac{1}{2})^2 \geq \frac{1}{2}$. Note that $x^2 \geq x$ if and only if $x^2 - x = x(x-1) \geq 0$. Consequently, $x^2 \geq x$ if and only if

$x \leq 0$ or $x \geq 1$. It follows that $\forall x(x^2 \geq x)$ is false if the domain consists of all real numbers (because the inequality is false for all real numbers x with $0 < x < 1$). However, if the domain consists of the integers, $\forall x(x^2 \geq x)$ is true, because there are no integers x with $0 < x < 1$. ∎

THE EXISTENTIAL QUANTIFIER

Many mathematical statements assert that there is an element with a certain property. Such statements are expressed using existential quantification. With existential quantification, we form a proposition that is true if and only if *P(x)* is true for at least one value of x in the domain.

Definition 1.4.2 *The **existential quantification** of $P(x)$ is the proposition*

"There exists an element x in the domain such that $P(x)$."

*We use the notation $\exists x P(x)$ for the existential quantification of $P(x)$. Here $\exists$ is called the **existential quantifier**.*

A domain must always be specified when a statement $\exists x P(x)$ is used. Furthermore, the meaning of $\exists x P(x)$ changes when the domain changes. Without specifying the domain, the statement $\exists x P(x)$ has no meaning.

Besides the phrase "there exists", we can also express existential quantification in many other ways, such as by using the words "for some", "for at least one", or "there is". The existential quantification $\exists x P(x)$ is read as

"There is an x such that *P(x)*",

"There is at least one x such that *P(x)*",

or

"For some *xP(x)*".

The meaning of the existential quantifier is summarized in the second row of Table 1.18. We illustrate the use of the existential quantifier in Examples 13–15.


EXAMPLE. 13

Let *P(x)* denote the statement "$x > 3$". What is the truth value of the quantification $\exists x P(x)$, where the domain consists of all real numbers?

Solution: Because "$x > 3$" is sometimes true—for instance, when $x = 4$ — the existential quantification of *P(x)*, which is $\exists x P(x)$, is true. ∎

Observe that the statement $\exists x P(x)$ is false if and only if there is no element x in the domain for which *P(x)* is true. That is, $\exists x P(x)$ is false if and only if *P(x)* is false for every element of the domain.We illustrate this observation in Example 14.

EXAMPLE. 14

Let *Q(x)* denote the statement "$x = x + 1$". What is the truth value of the quantification $\exists x Q(x)$, where the domain consists of all real numbers?

Solution: Because *Q(x)* is false for every real number x, the existential quantification of *Q(x)*, which is $\exists x Q(x)$, is false. ∎

> **Remark!** Generally, an implicit assumption is made that all domains of discourse for quantifiers are nonempty. If the domain is empty, then $\exists x Q(x)$ is false whenever *Q(x)* is a propositional function because when the domain is empty, there can be no element x in the domain for which *Q(x)* is true.

When all elements in the domain can be listed—say, $x_1, x_2, \ldots, x_n$ — the existential quantification $\exists x P(x)$ is the same as the disjunction

$$P(x_1) \vee P(x_2) \vee \ldots \vee P(x_n),$$

because this disjunction is true if and only if at least one of $P(x_1)$, $P(x_2)$, …, $P(x_n)$ is true.

EXAMPLE. 15

What is the truth value of $\exists x P(x)$, where *P(x)* is the statement "$x_2 > 10$" and the universe of discourse consists of the positive integers not exceeding 4?

Solution: Because the domain is 1, 2, 3, 4, the proposition $\exists x P(x)$ is the same as the disjunction

$$P(1) \vee P(2) \vee P(3) \vee P(4).$$

Because *P(4)*, which is the statement "$42 > 10$", is true, it follows that $\exists x P(x)$ is true. ∎

It is sometimes helpful to think in terms of looping and searching when determining the truth value of a quantification. Suppose that there are n objects in the domain for the variable x. To determine whether $\forall x P(x)$ is true, we can loop through all n values of x to see whether *P(x)* is always true. If we encounter a value x for which *P(x)* is false, then we have shown that $\forall x P(x)$ is false. Otherwise, $\forall x P(x)$ is true. To see whether $\exists x P(x)$ is true, we loop through the n values of x searching for a value for which *P(x)* is true. If we find one, then $\exists x P(x)$ is true. If we never find such an x, then we have determined that $\exists x P(x)$ is false. (Note that this searching procedure does not apply if there are infinitely many values in the domain. However, it is still a useful way of thinking about the truth values of quantifications.)

THE UNIQUENESS QUANTIFIER

We have now introduced universal and existential quantifiers. These are the most important quantifiers in mathematics and computer science. However, there is no limitation on the number of different quantifiers we can define, such as "there are exactly two", "there are no more than three", "there are at least 100", and so on. Of these other quantifiers, the one that is most often seen is the **uniqueness quantifier**, denoted by $\exists!$ or $\exists_1$. The notation $\exists! x P(x)$ [or $\exists_1 x P(x)$] states "There exists a unique x such that *P(x)* is true". (Other phrases for uniqueness quantification include "there is exactly one" and "there is one and only one.") For instance, $\exists! x(x - 1 = 0)$, where the domain is the set of real numbers, states that there is a unique real number x such that $x - 1 = 0$. This is a true statement, as $x = 1$ is the unique real number such that $x - 1 = 0$. Observe that we can use quantifiers and propositional logic to express uniqueness, so the uniqueness quantifier can be avoided. Generally, it is best to stick with existential and universal quantifiers so that rules of inference for these quantifiers can be used.

1.4.4 Quantifiers over finite domains

When the domain of a quantifier is finite, that is, when all its elements can be listed, quantified statements can be expressed using propositional logic. In particular, when the elements of the domain are $x_1, x_2, \ldots, x_n$, where n is a positive integer, the universal quantification $\forall x P(x)$ is the same as the conjunction

$$P(x_1) \wedge P(x_2) \wedge \ldots \wedge P(x_n),$$

because this conjunction is true if and only if $P(x_1), P(x_2), \ldots, P(x_n)$ are all true.

 EXAMPLE. 15

What is the truth value of $\forall x P(x)$, where $P(x)$ is the statement "$x^2 < 10$" and the domain consists of the positive integers not exceeding 4?

Solution: The statement $\forall x P(x)$ is the same as the conjunction

$$P(1) \wedge P(2) \wedge P(3) \wedge P(4),$$

because the domain consists of the integers *1, 2, 3,* and *4*. Because $P(4)$, which is the statement "$4^2 < 10$," is false, it follows that $\forall x P(x)$ is false. ∎

Similarly, when the elements of the domain are $x_1, x_2, \ldots, x_n$, where n is a positive integer, the existential quantification $\exists x P(x)$ is the same as the disjunction

$$P(x_1) \vee P(x_2) \vee \ldots \vee P(x_n),$$

because this disjunction is true if and only if at least one of $P(x_1)$, $P(x_2), \ldots, P(x_n)$ is true.

 EXAMPLE. 16

What is the truth value of $\exists x P(x)$, where $P(x)$ is the statement "$x^2 > 10$" and the universe of discourse consists of the positive integers not exceeding 4?

Solution: Because the domain is $\{1,\ 2,\ 3,\ 4\}$, the proposition $\exists x P(x)$ is the same as the disjunction

$$P(1) \vee P(2) \vee P(3) \vee P(4).$$

Because $P(4)$, which is the statement "$4^2 > 10$," is true, it follows that $\exists x P(x)$ is true. ∎

CONNECTIONS BETWEEN QUANTIFICATION AND LOOPING

It is sometimes helpful to think in terms of looping and searching when determining the truth value of a quantification. Suppose that there are n objects in the domain for the variable x. To determine whether $\forall x P(x)$ is true, we can loop through all n values of x to see whether $P(x)$ is always true. If we encounter a value x for which $P(x)$ is false, then we have shown that $\forall x P(x)$ is false. Otherwise, $\forall x P(x)$ is true. To see whether $\exists x P(x)$ is true, we loop through the n values of x searching for a value for which $P(x)$ is true. If we find one, then $\exists x P(x)$ is true. If we never find such an x, then we have determined that $\exists x P(x)$ is false.

1.4.5 Quantifiers with Restricted Domains

An abbreviated notation is often used to restrict the domain of a quantifier. In this notation, a condition a variable must satisfy is included after the quantifier. This is illustrated in Example 17.

EXAMPLE. 17

What do the statements $\forall x < 0(x^2 > 0), \forall y \neq 0(y^3 \neq 0)$, and $\exists z > 0(z^2 = 2)$ mean, where the domain in each case consists of the real numbers?

Solution: The statement $\forall x < 0(x^2 > 0)$ states that for every real number x with $x < 0, x^2 > 0$. That is, it states "The square of a negative real number is positive". This statement is the same as $\forall x(x < 0 \rightarrow x^2 > 0)$.

The statement $\forall y = 0(y^3 = 0)$ states that for every real number y with $y = 0$, we have $y^3 = 0$. That is, it states "The cube of every nonzero real number is nonzero". Note that this statement is equivalent to $\forall y(y = 0 \rightarrow y^3 = 0)$.

Finally, the statement $\exists z > 0(z^2 = 2)$ states that there exists a real number z with $z > 0$ such that $z^2 = 2$. That is, it states "There is a positive square root of 2." This statement is equivalent to $\exists z(z > 0 \wedge z^2 = 2)$. ∎

Note that the restriction of a universal quantification is the same as the universal quantification of a conditional statement. For instance, $\forall x < 0(x^2 > 0)$ is another way of expressing $\forall x(x < 0 \rightarrow x^2 > 0)$. On the other hand, the restriction of an existential quantification is the same as the existential quantification of a conjunction. For instance, $\exists z > 0(z^2 = 2)$ is another way of expressing $\exists z(z > 0 \wedge z^2 = 2)$.

1.4.6 Precedence of Quantifiers

The quantifiers $\forall$ and $\exists$ have higher precedence than all logical operators from propositional calculus. For example, $\forall x P(x) \vee Q(x)$ is the disjunction of $\forall x P(x)$ and *Q(x).* In other words, it means $(\forall x P(x)) \vee Q(x)$ rather than $\forall x(P(x) \vee Q(x))$.

1.4.7 Binding Variables

When a quantifier is used on the variable x, we say that this occurrence of the variable is bound. An occurrence of a variable that is not bound by a quantifier or set equal to a particular value is said to be **free**. All the variables that occur in a propositional function must be bound or set equal to a particular value to turn it into a proposition. This can be done using a combination of universal quantifiers, existential quantifiers, and value assignments.

The part of a logical expression to which a quantifier is applied is called the **scope** of this quantifier. Consequently, a variable is free if it is outside the scope of all quantifiers in the formula that specify this variable.

 EXAMPLE. 18

In the statement $\exists x(x + y = 1)$, the variable x is bound by the existential quantification $\exists x$, but the variable y is free because it is not bound by a quantifier and no value is assigned to this variable. This illustrates that in the statement $\exists x(x + y = 1)$, x is bound, but y is free.

In the statement $\exists x(P(x) \wedge Q(x)) \vee \forall x R(x)$, all variables are bound. The scope of the first quantifier, $\exists x$, is the expression $P(x) \wedge Q(x)$ because $\exists x$ is applied only to $P(x) \wedge Q(x)$, and not to the rest of the statement. Similarly, the

scope of the second quantifier, $\forall x$, is the expression $R(x)$. That is, the existential quantifier binds the variable x in $P(x) \wedge Q(x)$ and the universal quantifier $\forall x$ binds the variable x in $R(x)$. Observe that we could have written our statement using two different variables x and y, as $\forall x(P(x) \wedge Q(x)) \vee \forall y R(y)$, because the scopes of the two quantifiers do not overlap. The reader should be aware that in common usage, the same letter is often used to represent variables bound by different quantifiers with scopes that do not overlap. ∎

1.4.8 Logical Equivalences Involving Quantifiers

In previous section we introduced the notion of logical equivalences of compound propositions. We can extend this notion to expressions involving predicates and quantifiers.

Definition 1.4.3 *Statements involving predicates and quantifiers are* ***logically equivalent*** *if and only if they have the same truth value no matter which predicates are substituted into these statements and which domain of discourse is used for the variables in these propositional functions. We use the notation $S \equiv T$ to indicate that two statements S and T involving predicates and quantifiers are logically equivalent.*

Example 19 illustrates how to show that two statements involving predicates and quantifiers are logically equivalent.

EXAMPLE. 19

Show that $\forall x(P(x) \wedge Q(x))$ and $\forall x P(x) \wedge \forall x Q(x)$ are logically equivalent (where the same domain is used throughout). This logical equivalence shows that we can distribute a universal quantifier over a conjunction. Furthermore, we can also distribute an existential quantifier over a disjunction. However, we cannot distribute a universal quantifier over a disjunction, nor can we distribute an existential quantifier over a conjunction.

Solution: To show that these statements are logically equivalent, we must show that they always take the same truth value, no matter what the predicates P and Q are, and no matter which domain of discourse is used. Suppose we have particular predicates P and Q, with a common domain. We can show that $\forall x(P(x) \wedge Q(x))$ and $\forall x P(x) \wedge \forall x Q(x)$ are logically equivalent

by doing two things. First, we show that if $\forall x(P(x) \wedge Q(x))$ is true, then $\forall x P(x) \wedge \forall x Q(x)$ is true. Second, we show that if $\forall x P(x) \wedge \forall x Q(x)$ is true, then $\forall x(P(x) \wedge Q(x))$ is true.

So, suppose that $\forall x(P(x) \wedge Q(x))$ is true. This means that if a is in the domain, then $P(a) \wedge Q(a)$ is true. Hence, *P(a)* is true and *Q(a)* is true. Because *P(a)* is true and *Q(a)* is true for every element in the domain, we can conclude that $\forall x P(x)$ and $\forall x Q(x)$ are both true. This means that $\forall x P(x) \wedge \forall x Q(x)$ is true.

Next, suppose that $\forall x P(x) \wedge \forall x Q(x)$ is true. It follows that $\forall x P(x)$ is true and $\forall x Q(x)$ is true. Hence, if a is in the domain, then *P(a)* is true and *Q(a)* is true [because *P(x)* and *Q(x)* are both true for all elements in the domain, there is no conflict using the same value of a here]. It follows that for all $a, P(a) \wedge Q(a)$ is true. It follows that $\forall x(P(x) \wedge Q(x))$ is true. We can now conclude that

$$\forall x(P(x) \wedge Q(x)) \equiv \forall x P(x) \wedge \forall x Q(x). \quad \blacksquare$$

1.4.9 Negating Quantified Expressions

We will often want to consider the negation of a quantified expression. For instance, consider the negation of the statement

"Every student in your class has taken a course in calculus".

This statement is a universal quantification, namely,

$$\forall x P(x),$$

where *P(x)* is the statement "x has taken a course in calculus" and the domain consists of the students in your class. The negation of this statement is "It is not the case that every student in your class has taken a course in calculus". This is equivalent to "There is a student in your class who has not taken a course in calculus". And this is simply the existential quantification of the negation of the original propositional function, namely,

$$\exists x \neg P(x).$$

This example illustrates the following logical equivalence:

$$\neg \forall x P(x) \equiv \exists x \neg P(x).$$

To show that $\neg\forall x P(x)$ and $\exists x P(x)$ are logically equivalent no matter what the propositional function *P(x)* is and what the domain is, first note that $\neg\forall x P(x)$ is true if and only if $\forall x P(x)$ is false. Next, note that $\forall x P(x)$ is false if and only if there is an element x in the domain for which *P(x)* is false. This holds if and only if there is an element x in the domain for which $\neg P(x)$ is true. Finally, note that there is an element x in the domain for which $\neg P(x)$ is true if and only if $\exists x \neg P(x)$ is true. Putting these steps together, we can conclude that $\neg\forall x P(x)$ is true if and only if $\exists x \neg P(x)$ is true. It follows that $\neg\forall x P(x)$ and $\exists x \neg P(x)$ are logically equivalent.

Suppose we wish to negate an existential quantification. For instance, consider the proposition "There is a student in this class who has taken a course in calculus". This is the existential quantification

$$\exists x Q(x)$$

, where *Q(x)* is the statement "x has taken a course in calculus". The negation of this statement is the proposition "It is not the case that there is a student in this class who has taken a course in calculus". This is equivalent to "Every student in this class has not taken calculus", which is just the universal quantification of the negation of the original propositional function, or, phrased in the language of quantifiers,

$$\forall x \neg Q(x).$$

This example illustrates the equivalence

$$\neg\exists x Q(x) \equiv \forall x \neg Q(x).$$

To show that $\neg\exists x Q(x)$ and $\forall x \neg Q(x)$ are logically equivalent no matter what *Q(x)* is and what the domain is, first note that $\neg\exists x Q(x)$ is true if and only if $\exists x Q(x)$ is false. This is true if and only if no x exists in the domain for which *Q(x)* is true. Next, note that no x exists in the domain for which *Q(x)* is true if and only if *Q(x)* is false for every x in the domain. Finally, note that *Q(x)* is false for every x in the domain if and only if $\neg Q(x)$ is true for all x in the domain, which holds if and only if $\forall x \neg Q(x)$ is true. Putting these steps together, we see that $\neg\exists x Q(x)$ is true if and only if $\forall x \neg Q(x)$ is true. We conclude that $\neg\exists x Q(x)$ and $\forall x \neg Q(x)$ are logically equivalent.

Table 1.19: De Morgan's Laws for Quantifiers.

Negation	**Equivalent Statement**	**When Is Negation True?**	**When False?**
$\neg\exists P(x)$	$\forall x \neg P(x)$	For every x, $P(x)$ is false.	There is an x for which $P(x)$ is true.
$\neg\forall P(x)$	$\exists x \neg P(x)$	There is an x for which $P(x)$ is false.	$P(x)$ is true for every x.

The rules for negations for quantifiers are called De Morgan's laws for quantifiers. These rules are summarized in Table 1.19.

Remark! When the domain of a predicate $P(x)$ consists of n elements, where n is a positive integer greater than one, the rules for negating quantified statements are exactly the same as De Morgan's laws discussed in previous section. This is why these rules are called De Morgan's laws for quantifiers. When the domain has n elements $x_1, x_2, \ldots, x_n$, it follows that $\neg\forall x P(x)$ is the same as $\neg(P(x_1) \wedge P(x_2) \wedge \ldots \wedge P(x_n))$, which is equivalent to $\neg P(x_1) \vee \neg P(x_2) \vee \ldots \vee \neg P(x_n)$ by De Morgan's laws, and this is the same as $\exists x \neg P(x)$. Similarly, $\neg\exists x P(x)$ is the same as $\neg(P(x_1) \vee P(x_2) \vee \ldots \vee P(x_n))$, which by De Morgan's laws is equivalent to $\neg P(x_1) \wedge \neg P(x_2) \wedge \ldots \wedge \neg P(x_n)$, and this is the same as $\forall x \neg P(x)$.

We illustrate the negation of quantified statements in Examples 20 and 21.

EXAMPLE. 20

What are the negations of the statements "There is an honest politician" and "All Americans eat cheeseburgers?"

Solution: Let $H(x)$ denote "x is honest". Then the statement "There is an honest politician" is represented by $\exists x H(x)$, where the domain consists of all politicians. The negation of this statement is $\neg\exists x H(x)$, which is equivalent to $\forall x \neg H(x)$. This negation can be expressed as "Every politician is dishonest". (Note: In English, the statement "All politicians are not honest" is ambiguous.

In common usage, this statement often means "Not all politicians are honest". Consequently, we do not use this statement to express this negation.)

Let *C(x)* denote "x eats cheeseburgers". Then the statement "All Americans eat cheeseburgers" is represented by $\forall x C(x)$, where the domain consists of all Americans. The negation of this statement is $\neg\forall x C(x)$, which is equivalent to $\exists x \neg C(x)$. This negation can be expressed in several different ways, including "Some American does not eat cheeseburgers" and "There is an American who does not eat cheeseburgers". ∎

EXAMPLE. 21

What are the negations of the statements $\forall x(x^2 > x)$ and $\exists x(x^2 = 2)$?

Solution: The negation of $\forall x(x^2 > x)$ is the statement $\neg\forall x(x^2 > x)$, which is equivalent to $\exists x \neg(x^2 > x)$. This can be rewritten as $\exists x(x^2 \leq x)$. The negation of $\exists x(x^2 = 2)$ is the statement $\neg\exists x(x^2 = 2)$, which is equivalent to $\forall x \neg(x^2 = 2)$. This can be rewritten as truth values of these statements depend on the domain. ∎

We use De Morgan's laws for quantifiers in Example 21.

EXAMPLE. 22

Show that $\neg\forall x(P(x) \rightarrow Q(x))$ and $\exists x(P(x) \wedge \neg Q(x))$ are logically equivalent.

Solution: By De Morgan's law for universal quantifiers, we know that $\neg\forall x(P(x) \rightarrow Q(x))$ and $\exists x(\neg(P(x) \rightarrow Q(x)))$ are logically equivalent. By the fifth logical equivalence in Table 1.16 in previous section, we know that $\neg(P(x) \rightarrow Q(x))$ and $P(x \wedge \neg Q(x)$ are logically equivalent for every x. Because we can substitute one logically equivalent expression for another in a logical equivalence, it follows that $\neg\forall x(P(x) \rightarrow Q(x))$ and $\exists x(P(x) \wedge \neg Q(x))$ are logically equivalent. ∎

1.4.10 Translating from English into Logical Expressions

Translating sentences in English (or other natural languages) into logical expressions is a crucial task in mathematics, logic programming,

artificial intelligence, software engineering, and many other disciplines. We began studying this topic first section, where we used propositions to express sentences in logical expressions. In that discussion, we purposely avoided sentences whose translations required predicates and quantifiers. Translating from English to logical expressions becomes even more complex when quantifiers are needed. Furthermore, there can be many ways to translate a particular sentence. (As a consequence, there is no "cookbook" approach that can be followed step by step.) We will use some examples to illustrate how to translate sentences from English into logical expressions. The goal in this translation is to produce simple and useful logical expressions. In this section, we restrict ourselves to sentences that can be translated into logical expressions using a single quantifier; in the next section, we will look at more complicated sentences that require multiple quantifiers.

 EXAMPLE. 23

Express the statement "Every student in this class has studied calculus" using predicates and quantifiers.

Solution: First, we rewrite the statement so that we can clearly identify the appropriate quantifiers to use. Doing so, we obtain:

"For every student in this class, that student has studied calculus."

Next, we introduce a variable x so that our statement becomes

"For every student x in this class, x has studied calculus."

Continuing, we introduce *C(x)*, which is the statement "x has studied calculus". Consequently, if the domain for x consists of the students in the class, we can translate our statement as $\forall x C(x)$.

However, there are other correct approaches; different domains of discourse and other predicates can be used. The approach we select depends on the subsequent reasoning we want to carry out. For example, we may be interested in a wider group of people than only those in this class. If we change the domain to consist of all people, we will need to express our statement as

"For every person x, if person x is a student in this class then x has studied calculus."

If *S(x)* represents the statement that person x is in this class, we see that our statement can be expressed as $\forall x(S(x) \rightarrow C(x))$.

[**Caution!** Our statement cannot be expressed as $\forall x(S(x) \wedge C(x))$ because this statement says that all people are students in this class and have studied calculus!]

Finally, when we are interested in the background of people in subjects besides calculus, we may prefer to use the two-variable quantifier *Q(x, y)* for the statement "student x has studied subject y". Then we would replace *C(x)* by *Q(x*, calculus) in both approaches to obtain $\forall x Q(x,$ calculus) or $\forall x(S(x) \rightarrow Q(x,$ calculus)). ∎

In Example 23 we displayed different approaches for expressing the same statement using predicates and quantifiers. However, we should always adopt the simplest approach that is adequate for use in subsequent reasoning.

EXAMPLE. 24

Express the statements "Some student in this class has visited Mexico" and "Every student in this class has visited either Canada or Mexico" using predicates and quantifiers.

Solution: The statement "Some student in this class has visited Mexico" means that

"There is a student in this class with the property that the student has visited Mexico."

We can introduce a variable x, so that our statement becomes

"There is a student x in this class having the property that x has visited Mexico."

We introduce *M(x)*, which is the statement "x has visited Mexico." If the domain for x consists of the students in this class, we can translate this first statement as $\exists x M(x)$.

However, if we are interested in people other than those in this class, we look at the statement a little differently. Our statement can be expressed as

"There is a person x having the properties that x is a student in this class and x has visited Mexico."

In this case, the domain for the variable x consists of all people. We introduce *S(x)* to represent "x is a student in this class". Our solution becomes $\exists x(S(x) \wedge M(x))$ because the statement is that there is a person x who is a student in this class and who has visited Mexico.

[**Caution!** Our statement cannot be expressed as $\exists x(S(x) \rightarrow M(x))$, which is true when there is someone not in the class because, in that case, for such a person x, $S(x) \rightarrow M(x)$ becomes either $F \rightarrow T$ or $F \rightarrow F$, both of which are true.]

Similarly, the second statement can be expressed as "For every x in this class, x has the property that x has visited Mexico or x has visited Canada."

(Note that we are assuming the inclusive, rather than the exclusive, or here.) We let *C(x)* be "x has visited Canada". Following our earlier reasoning, we see that if the domain for x consists of the students in this class, this second statement can be expressed as $\forall x(C(x) \vee M(x))$. However, if the domain for x consists of all people, our statement can be expressed as

"For every person x, if x is a student in this class, then x has visited Mexico or x has visited Canada."

In this case, the statement can be expressed as $\forall x(S(x) \rightarrow (C(x) \vee M(x)))$.

Instead of using *M(x)* and *C(x)* to represent that x has visited Mexico and x has visited Canada, respectively, we could use a two-place predicate *V (x, y)* to represent "x has visited country y". In this case, *V (x,* Mexico) and *V (x,* Canada)would have the same meaning as *M(x)* and *C(x)* and could replace them in our answers. If we are working with many statements that involve people visiting different countries, we might prefer to use this two-variable approach. Otherwise, for simplicity, we would stick with the one-variable predicates M(x) and C(x). ∎

1.4.11 Using Quantifiers in System Specifications

Many system specifications involve predicates and quantifications. This is illustrated in Example 25.

EXAMPLE. 25

Use predicates and quantifiers to express the system specifications "Every mail message larger than one megabyte will be compressed" and "If a user is active, at least one network link will be available."

Solution: Let *S(m, y)* be "Mail message m is larger than y megabytes", where the variable x has the domain of all mail messages and the variable y is a positive real number, and let *C(m)* denote "Mail message m will be compressed." Then the specification "Every mail message larger than one megabyte will be compressed" can be represented as $\forall m(S(m, 1) \rightarrow C(m))$.

Let *A(u)* represent "User u is active," where the variable u has the domain of all users, let *S(n, x)* denote "Network link n is in state x," where n has the domain of all network links and x has the domain of all possible states for a

network link. Then the specification "If a user is active, at least one network link will be available" can be represented by $\exists u A(u) \rightarrow \exists n S(n,\ \text{available})$. ∎

1.4.12 Examples from Lewis Carroll

Lewis Carroll (really C. L. Dodgson writing under a pseudonym), the author of *Alice in Wonderland*, is also the author of several works on symbolic logic. His books contain many examples of reasoning using quantifiers. Examples 26 and 27 come from his book *Symbolic Logic*; other examples from that book are given in the exercises at the end of this section. These examples illustrate how quantifiers are used to express various types of statements.

EXAMPLE. 26

Consider these statements. The first two are called premises and the third is called the conclusion. The entire set is called an argument.

"All lions are fierce."
"Some lions do not drink coffee."
"Some fierce creatures do not drink coffee."

Let *P(x), Q(x)*, and *R(x)* be the statements "x is a lion", "x is fierce", and "x drinks coffee", respectively.Assuming that the domain consists of all creatures, express the statements in the argument using quantifiers and *P(x), Q(x)*, and *R(x)*.

Solution: We can express these statements as:

$\forall x(P(x) \rightarrow Q(x))$.
$\exists x(P(x) \wedge negR(x))$.
$\exists x(Q(x) \wedge negR(x))$.

Notice that the second statement cannot be written as $\exists x(P(x) \rightarrow \neg R(x))$. The reason is that $P(x) \rightarrow \neg R(x)$ is true whenever x is not a lion, so that $\exists x(P(x) \rightarrow \neg R(x))$ is true as long as there is at least one creature that is not a lion, even if every lion drinks coffee. Similarly, the third statement cannot be written as

$$\exists x(Q(x) \rightarrow \neg R(x)).$$

∎

EXAMPLE. 27

Consider these statements, of which the first three are premises and the fourth is a valid conclusion.

"All hummingbirds are richly colored."
"No large birds live on honey"
"Birds that do not live on honey are dull in color."
"Hummingbirds are small."

Let *P(x), Q(x), R(x)*, and *S(x)* be the statements "x is a hummingbird", "x is large", "x lives on honey", and "x is richly colored", respectively. Assuming that the domain consists of all birds, express the statements in the argument using quantifiers and *P(x), Q(x), R(x)*, and *S(x)*.

Solution: We can express the statements in the argument as

$\forall x(P(x) \rightarrow S(x))$.
$\neg\exists x(Q(x) \wedge R(x))$.
$\forall x(\neg R(x) \rightarrow \neg S(x))$.
$\forall x(P(x) \rightarrow \neg Q(x))$.

(Note we have assumed that "small" is the same as "not large" and that "dull in color" is the same as "not richly colored". To show that the fourth statement is a valid conclusion of the first three, we need to use rules of inference that will be discussed in next section. ■

1.5 Nested Quantifiers

1.5.1 Introduction

In Section 1.4 we defined the existential and universal quantifiers and showed how they can be used to represent mathematical statements. We also explained how they can be used to translate English sentences into logical expressions. However, in Section 1.4 we avoided **nested quantifiers**, where one quantifier is within the scope of another, such as

$$\forall x\exists y(x + y = 0).$$

Note that everything within the scope of a quantifier can be thought of as a propositional function.

For example,

$$\forall x \exists y(x + y = 0)$$

is the same thing as $\forall x Q(x)$, where $Q(x)$ is $\exists y P(x, y)$, where $P(x, y)$ is $x + y = 0$.

Nested quantifiers commonly occur in mathematics and computer science. Although nested quantifiers can sometimes be difficult to understand, the rules we have already studied in Section 1.4 can help us use them. In this section we will gain experience working with nested quantifiers.We will see how to use nested quantifiers to express mathematical statements such as "The sum of two positive integers is always positive". We will show how nested quantifiers can be used to translate English sentences such as "Everyone has exactly one best friend" into logical statements. Moreover, we will gain experience working with the negations of statements involving nested quantifiers.

1.5.2 Understanding Statements Involving Nested Quantifiers

To understand statements involving nested quantifiers, we need to unravel what the quantifiers and predicates that appear mean. This is illustrated in Examples 1 and 2.

EXAMPLE. 1

Assume that the domain for the variables x and y consists of all real numbers. The statement

$$\forall x \forall y(x + y = y + x)$$

says that $x + y = y + x$ for all real numbers x and y. This is the commutative law for addition of real numbers. Likewise, the statement

$$\forall x \exists y(x + y = 0)$$

says that for every real number x there is a real number y such that $x + y = 0$. This states that every real number has an additive inverse. Similarly, the statement

$$\forall x \forall y \forall z(x + (y + z) = (x + y) + z)$$

is the associative law for addition of real numbers. ∎

EXAMPLE. 2

Translate into English the statement

$$\forall x \forall y((x > 0) \wedge (y < 0) \rightarrow (xy < 0)),$$

where the domain for both variables consists of all real numbers.

Solution: This statement says that for every real number x and for every real number y, if $x > 0$ and $y < 0$, then $xy < 0$. That is, this statement says that for real numbers x and y, if x is positive and y is negative, then xy is negative. This can be stated more succinctly as "The product of a positive real number and a negative real number is always a negative real number". ∎

THINKING OF QUANTIFICATION AS LOOPS

In working with quantifications of more than one variable, it is sometimes helpful to think in terms of nested loops. (Of course, if there are infinitely many elements in the domain of some variable, we cannot actually loop through all values. Nevertheless, this way of thinking is helpful in understanding nested quantifiers.) For example, to see whether $\forall x \forall y P(x, y)$ is true, we loop through the values for x, and for each x we loop through the values for y. If we find that $P(x, y)$ is true for all values for x and y, we have determined that $\forall x \forall y P(x, y)$ is true. If we ever hit a value x for which we hit a value y for which $P(x, y)$ is false, we have shown that $\forall x \forall y P(x, y)$ is false.

Similarly, to determine whether $\forall x \exists y P(x, y)$ is true, we loop through the values for x. For each x we loop through the values for y until we find a y for which $P(x, y)$ is true. If for every x we hit such a y, then $\forall x \exists y P(x, y)$ is true; if for some x we never hit such a y, then $\forall x \exists y P(x, y)$ is false. To see whether $\exists x \forall y P(x, y)$ is true, we loop through the values for x until we find an x for which $P(x, y)$ is always true when we loop through all values for y. Once we find such an x, we knowthat $\exists x \forall y P(x, y)$ is true. If we never hit such an x, then we knowthat $\exists x \forall y P(x, y)$ is false.

Finally, to see whether $\exists x \exists y P(x, y)$ is true, we loop through the values for x, where for each x we loop through the values for y until we

hit an x for which we hit a y for which $P(x, y)$ is true. The statement $\exists x \exists y P(x, y)$ is false only if we never hit an x for which we hit a y such that $P(x, y)$ is true.

1.5.3 The Order of Quantifiers

Many mathematical statements involve multiple quantifications of propositional functions involving more than one variable. It is important to note that the order of the quantifiers is important, unless all the quantifiers are universal quantifiers or all are existential quantifiers. These remarks are illustrated by Examples 3–5.

 EXAMPLE. 3

Let *P(x, y)* be the statement "$x + y = y + x$". What are the truth values of the quantifications $\forall x \forall y P(x, y)$ and $\forall y \forall x P(x, y)$ where the domain for all variables consists of all real numbers?

Solution: The quantification

$$\forall x \forall y P(x, y)$$

denotes the proposition

"For all real numbers x, for all real numbers y, $x + y = y + x$".

Because *P(x, y)* is true for all real numbers x and y (it is the commutative law for addition, which is an axiom for the real numbers—see Appendix 1), the proposition $\forall x \forall y P(x, y)$ is true. Note that the statement $\forall y \forall x P(x, y)$ says "For all real numbers y, for all real numbers x, $x + y = y + x$". This has the same meaning as the statement "For all real numbers x, for all real numbers y, $x + y = y + x$". That is, $\forall x \forall y P(x, y)$ and $\forall y \forall x P(x, y)$ have the same meaning, and both are true. This illustrates the principle that the order of nested universal quantifiers in a statement without other quantifiers can be changed without changing the meaning of the quantified statement. ∎

 EXAMPLE. 4

Let *Q(x, y)* denote "$x + y = 0$." What are the truth values of the quantifications $\exists y \forall x Q(x, y)$ and $\forall x \exists y Q(x, y)$, where the domain for all variables consists of all real numbers?

Solution: The quantification

$$\exists y \forall x Q(x, y)$$

denotes the proposition

"There is a real number y such that for every real number x, $Q(x, y)$."

No matter what value of y is chosen, there is only one value of x for which $x + y = 0$. Because there is no real number y such that $x + y = 0$ for all real numbers x, the statement $\exists y \forall x Q(x, y)$ is false.

The quantification

$$\forall x \exists y Q(x, y)$$

denotes the proposition

"For every real number x there is a real number y such that $Q(x, y)$."

Given a real number x, there is a real number y such that $x + y = 0$; namely, $y = -x$. Hence, the statement $\forall x \exists y Q(x, y)$ is true. ■

Example 4 illustrates that the order in which quantifiers appear makes a difference. The statements $\exists y \forall x P(x, y)$ and $\forall x \exists y P(x, y)$ are not logically equivalent. The statement $\exists y \forall x P(x, y)$ is true if and only if there is a y that makes $P(x, y)$ true for every x. So, for this statement to be true, there must be a particular value of y for which $P(x, y)$ is true regardless of the choice of x. On the other hand, $\forall x \exists y P(x, y)$ is true if and only if for every value of x there is a value of y for which $P(x, y)$ is true. So, for this statement to be true, no matter which x you choose, there must be a value of y (possibly depending on the x you choose) for which $P(x, y)$ is true. In other words, in the second case, y can depend on x, whereas in the first case, y is a constant independent of x.

From these observations, it follows that if $\exists y \forall x P(x, y)$ is true, then $\forall x \exists y P(x, y)$ must also be true. However, if $\forall x \exists y P(x, y)$ is true, it is not necessary for $\exists y \forall x P(x, y)$ to be true.

Table 1.20 summarizes the meanings of the different possible quantifications involving two variables.

Quantifications of more than two variables are also common, as Example 5 illustrates.


EXAMPLE. 5

Table 1.20: Quantifications of Two Variables.

Statement	**When True?**	**When False?**
$\forall x \forall y P(x, y)$ $\forall y \forall x P(x, y)$	*P(x, y)* is true for every pair *x, y.*	There is a pair *x, y* for which *P(x, y)* is false.
$\exists x \forall y P(x, y)$	For every x there is a y for which *P(x, y)* is true.	There is an x such that *P(x, y)* is false for every y.
$\exists x \forall y P(x, y)$	There is an x for which *P(x, y)* is true for every y.	For every x there is a y for which *P(x, y)* is false.
$\exists x \exists y P(x, y)$ $\exists y \exists x P(x, y)$	There is a pair *x, y* for which *P(x, y)* is true.	*P(x, y)* is false for every pair *x, y.*

Let *Q(x, y, z)* be the statement "$x + y = z$." What are the truth values of the statements $\forall x \forall y \exists z Q(x, y, z)$ and $\exists z\ \forall x \forall y Q(x, y, z)$, where the domain of all variables consists of all real numbers?

Solution: Suppose that x and y are assigned values. Then, there exists a real number z such that $x + y = z$. Consequently, the quantification

$$\forall x \forall y \exists z Q(x, y, z),$$

which is the statement

"For all real numbers x and for all real numbers y there is a real number z such that $x + y = z$,"

is true. The order of the quantification here is important, because the quantification

$$\exists z\ \forall x \forall y Q(x, y, z),$$

which is the statement

"There is a real number z such that for all real numbers x and for all real numbers y it is true that $x + y = z$,"

is false, because there is no value of z that satisfies the equation $x + y = z$ for all values of x and y. ∎

1.5.4 Translating Mathematical Statements into Statements Involving Nested Quantifiers

Mathematical statements expressed in English can be translated into logical expressions, as Examples 6–8 show.

 EXAMPLE. 6

Translate the statement "The sum of two positive integers is always positive" into a logical expression.

Solution: To translate this statement into a logical expression, we first rewrite it so that the implied quantifiers and a domain are shown: "For every two integers, if these integers are both positive, then the sum of these integers is positive". Next, we introduce the variables x and y to obtain "For all positive integers x and y, $x+y$ is positive". Consequently, we can express this statement as

$$\forall x \forall y((x > 0) \wedge (y > 0) \rightarrow (x + y > 0)),$$

where the domain for both variables consists of all integers. Note that we could also translate this using the positive integers as the domain. Then the statement "The sum of two positive integers is always positive" becomes "For every two positive integers, the sum of these integers is positive". We can express this as

$$\forall x \forall y(x + y > 0),$$

where the domain for both variables consists of all positive integers. ∎

 EXAMPLE. 7

Translate the statement "Every real number except zero has a multiplicative inverse." (**A multiplicative inverse** of a real number x is a real number y such that $xy = 1$.)

Solution: We first rewrite this as "For every real number x except zero, x has a multiplicative inverse". We can rewrite this as "For every real number x, if $x = 0$, then there exists a real number y such that $xy = 1$". This can be rewritten as

$$\forall x((x = 0) \rightarrow \exists y(xy = 1)).$$ ∎

One example that you may be familiar with is the concept of limit, which is important in calculus.

EXAMPLE. 8

(**Requires calculus**) Use quantifiers to express the definition of the limit of a real-valued function *f(x)* of a real variable x at a point a in its domain.

Solution: Recall that the definition of the statement

$$lim_{x \to a} f(x) = L$$

is: For every real number $\epsilon > 0$ there exists a real number $\delta > 0$ such that $|f(x) - L| < \epsilon$ whenever $0 < |x - a| < \delta$. This definition of a limit can be phrased in terms of quantifiers by

$$\forall \epsilon \exists \delta \forall x (0 < |x - a| < \delta \rightarrow |f(x) - L| < \epsilon),$$

where the domain for the variables δ and ϵ consists of all positive real numbers and for x consists of all real numbers.

This definition can also be expressed as

$$\forall \epsilon > 0 \exists \delta > 0 \forall x (0 < |x - a| < \delta \rightarrow |f(x) - L| < \epsilon),$$

when the domain for the variables ϵ and δ consists of all real numbers, rather than just the positive real numbers. [Here, restricted quantifiers have been used. Recall that $\forall x > 0$ *P(x)* means that for all x with $x > 0, P(x)$ is true.]

■

1.5.5 Translating from Nested Quantifiers into English

Expressions with nested quantifiers expressing statements in English can be quite complicated. The first step in translating such an expression is to write out what the quantifiers and predicates in the expression mean. The next step is to express this meaning in a simpler sentence. This process is illustrated in Examples 9 and 10.

EXAMPLE. 9

Translate the statement

$$\forall x(C(x) \vee \exists y(C(y) \wedge F(x, y)))$$

into English, where *C(x)* is "x has a computer", *F(x, y)* is "x and y are friends", and the domain for both x and y consists of all students in your school.

Solution: The statement says that for every student x in your school, x has a computer or there is a student y such that y has a computer and x and y are friends. In other words, every student in your school has a computer or has a friend who has a computer. ∎

EXAMPLE. 10

Translate the statement

$$\exists x \forall y \forall z((F(x, y) \wedge F(x, z) \wedge (y \neq z)) \rightarrow \neg F(y, z))$$

into English, where *F(a,b)* means a and b are friends and the domain for x, y, and z consists of all students in your school.

Solution: We first examine the expression $(F(x, y) \wedge F(x, z) \wedge (y \neq z)) \rightarrow \neg F(y, z)$. This expression says that if students x and y are friends, and students x and z are friends, and furthermore, if y and z are not the same student, then y and z are not friends. It follows that the original statement, which is triply quantified, says that there is a student x such that for all students y and all students z other than y, if x and y are friends and x and z are friends, then y and z are not friends. In other words, there is a student none of whose friends are also friends with each other. ∎

1.5.6 Translating English Sentences into Logical Expressions

In Section 1.4 we showed how quantifiers can be used to translate sentences into logical expressions. However, we avoided sentences whose translation into logical expressions required the use of nested quantifiers.We now address the translation of such sentences.

EXAMPLE. 11

Express the statement "If a person is female and is a parent, then this person is someone's mother" as a logical expression involving predicates, quantifiers with a domain consisting of all people, and logical connectives.

Solution: The statement "If a person is female and is a parent, then this person is someone's mother" can be expressed as "For every person x, if person x is female and person x is a parent, then there exists a person y such that person x is the mother of person y." We introduce the propositional functions *F(x)* to represent "x is female," *P(x)* to represent "x is a parent," and *M(x, y)* to represent "x is the mother of y." The original statement can be represented as

$$\forall x((F(x) \wedge P(x)) \rightarrow \exists y M(x, y)).$$

Using the null quantification rule, we can move $\exists y$ to the left so that it appears just after $\forall x$, because y does not appear in $F(x) \wedge P(x)$. We obtain the logically equivalent expression

$$\forall x \exists y((F(x) \wedge P(x) \rightarrow M(x, y)). \quad \blacksquare$$

EXAMPLE. 12

Express the statement "Everyone has exactly one best friend" as a logical expression involving predicates, quantifiers with a domain consisting of all people, and logical connectives.

Solution: The statement "Everyone has exactly one best friend" can be expressed as "For every person x, person x has exactly one best friend." Introducing the universal quantifier, we see that this statement is the same as "$\forall x$(person x has exactly one best friend)," where the domain consists of all people.

To say that x has exactly one best friend means that there is a person y who is the best friend of x, and furthermore, that for every person z, if person z is not person y, then z is not the best friend of x. When we introduce the predicate *B(x, y)* to be the statement "y is the best friend of x," the statement that x has exactly one best friend can be represented as

$$\exists y(B(x, y) \wedge \forall z((z = y) \rightarrow \neg B(x, z))).$$

Consequently, our original statement can be expressed as

$$\forall x \exists y(B(x, y) \wedge \forall z((z = y) \rightarrow \neg B(x, z))) \quad \blacksquare.$$

EXAMPLE. 13

Use quantifiers to express the statement "There is a woman who has taken a flight on every airline in the world."

Solution: *Let P(w, f)* be "w has taken f " and *Q(f, a)* be "f is a flight on a. "We can express the statement as

$$\exists w \forall a \exists f (P(w, f) \wedge Q(f, a)),$$

where the domains of discourse for w, f, and a consist of all thewomen in theworld, all airplane flights, and all airlines, respectively.

The statement could also be expressed as

$$\exists w \forall a \exists f R(w, f, a),$$

where *R(w, f, a)* is "w has taken f on a." Although this is more compact, it somewhat obscures the relationships among the variables. Consequently, the first solution is usually preferable. ∎

1.5.7 Negating Nested Quantifiers

Statements involving nested quantifiers can be negated by successively applying the rules for negating statements involving a single quantifier. This is illustrated in Examples 14–16.

EXAMPLE. 14

Express the negation of the statement $\forall x \exists y(xy = 1)$ so that no negation precedes a quantifier.

Solution: By successively applying De Morgan's laws for quantifiers in Table 1.19 of Section 1.4, we can move the negation in $\neg \forall x \exists y(xy = 1)$ inside all the quantifiers. We find that $\neg \forall x \exists y(xy = 1)$ is equivalent to $\exists x \neg \exists y(xy = 1)$, which is equivalent to $\exists x \forall y \neg(xy = 1)$. Because $\neg(xy = 1)$ can be expressed more simply as $xy = 1$, we conclude that our negated statement can be expressed as $\exists x \forall y(xy \neq 1)$. ∎

EXAMPLE. 15

Use quantifiers to express the statement that "There does not exist a woman who has taken a flight on every airline in the world."

Solution: This statement is the negation of the statement "There is a woman who has taken a flight on every airline in the world" from Example 13. By Example 13, our statement can be expressed as $\neg\exists w\forall a\exists f(P(w,f) \wedge Q(f,a))$, where $P(w,f)$ is "w has taken f" and *Q(f, a)* is "f is a flight on a". By successively applying De Morgan's laws for quantifiers in Table 1.19

of Section 1.4 to move the negation inside successive quantifiers and by applying De Morgan's law for negating a conjunction in the last step, we find that our statement is equivalent to each of this sequence of statements:

$$\begin{aligned}\forall w\neg\forall a\exists f(P(w,f) \wedge Q(f,a)) &\equiv \forall w\exists a\neg\exists f(P(w,f) \wedge Q(f,a))\\ &\equiv \forall w\exists a\forall f\neg(P(w,f) \wedge Q(f,a))\\ &\equiv \forall w\exists a\forall f(\neg P(w,f) \vee \neg Q(f,a))\end{aligned}$$

This last statement states "For every woman there is an airline such that for all flights, this woman has not taken that flight or that flight is not on this airline." ∎

EXAMPLE. 16

(*Requires calculus*) Use quantifiers and predicates to express the fact that $lim_{x\to a}f(x)$ does not exist where *f (x)* is a real-valued function of a real variable x and a belongs to the domain of f.

Solution: To say that $\lim_{x\to a} f(x)$ does not exist means that for all real numbers L, $\lim_{x\to a} f(x) \neq L$. By using Example 8, the statement $\lim_{x\to a} f(x) \neq L$ can be expressed as

$$\neg\forall\epsilon > 0\exists\delta > 0\forall x(0 < |x-a| < \delta \rightarrow |f(x)-L| < \epsilon).$$

Successively applying the rules for negating quantified expressions, we construct this sequence of equivalent statements

$$\begin{aligned}\neg\forall\epsilon > 0 \quad &\exists\delta > 0\forall x(0 < |x-a| < \delta \rightarrow |f(x)-L| < \epsilon)\\ &\equiv \exists\epsilon > 0\neg\exists\delta > 0\forall x(0 < |x-a| < \delta \rightarrow |f(x)-L| < \epsilon)\\ &\equiv \exists\epsilon > 0\forall\delta > 0\neg\forall x(0 < |x-a| < \delta \rightarrow |f(x)-L| < \epsilon)\\ &\equiv \exists\epsilon > 0\forall\delta > 0\exists x\neg(0 < |x-a| < \delta \rightarrow |f(x)-L| < \epsilon)\\ &\equiv \exists\epsilon > 0\forall\delta > 0\exists x(0 < |x-a| < \delta \wedge |f(x)-L| \geq \epsilon).\end{aligned}$$

In the last step we used the equivalence $\neg(p \rightarrow q) \equiv p \wedge \neg q$, which follows from the fifth equivalence in Table 1.16 of Section 1.3.

Because the statement "$\lim_{x \to a} f(x)$ does not exist" means for all real numbers L, $\lim_{x \to a} f(x) \neq L$, this can be expressed as

$$\forall L \exists \epsilon > 0 \forall \delta > 0 \exists x (0 < |x - a| < \delta \wedge |f(x) - L| \geq \epsilon).$$

This last statement says that for every real number L there is a real number $\epsilon > 0$ such that for every real number $\delta > 0$, there exists a real number x such that $0 < |x - a| < \delta$ and $|f(x) - L| \geq \epsilon$ ∎

1.6 Rules of Inference

1.6.1 Introduction

Later in this chapter we will study proofs. Proofs in mathematics are valid arguments that establish the truth of mathematical statements. By an **argument**, we mean a sequence of statements that end with a conclusion. By **valid**, we mean that the conclusion, or final statement of the argument, must follow from the truth of the preceding statements, or **premises**, of the argument. That is, an argument is valid if and only if it is impossible for all the premises to be true and the conclusion to be false. To deduce new statements from statements we already have, we use rules of inference which are templates for constructing valid arguments. Rules of inference are our basic tools for establishing the truth of statements.

Before we study mathematical proofs, we will look at arguments that involve only compound propositions.We will define what it means for an argument involving compound propositions to be valid. Then we will introduce a collection of rules of inference in propositional logic. These rules of inference are among the most important ingredients in producing valid arguments. After we illustrate how rules of inference are used to produce valid arguments, we will describe some common forms of incorrect reasoning, called **fallacies**, which lead to invalid arguments.

After studying rules of inference in propositional logic, we will introduce rules of inference for quantified statements.We will describe how these rules of inference can be used to produce valid arguments.

These rules of inference for statements involving existential and universal quantifiers play an important role in proofs in computer science and mathematics, although they are often used without being explicitly mentioned.

Finally, we will show how rules of inference for propositions and for quantified statements can be combined. These combinations of rule of inference are often used together in complicated arguments.

1.6.2 Valid Arguments in Propositional Logic

Consider the following argument involving propositions (which, by definition, is a sequence of propositions):

"If you have a current password, then you can log onto the network."

"You have a current password."

Therefore,

"You can log onto the network.

We would like to determine whether this is a valid argument. That is, we would like to determine whether the conclusion "You can log onto the network" must be true when the premises "If you have a current password, then you can log onto the network" and "You have a current password" are both true.

Before we discuss the validity of this particular argument, we will look at its form. Use p to represent "You have a current password" and q to represent "You can log onto the network." Then, the argument has the form

$$\begin{array}{l} p \rightarrow q \\ p \\ \hline \therefore\ q \end{array}$$

where $\therefore$ is the symbol that denotes "therefore".

We know that when p and q are propositional variables, the statement $((p \rightarrow q) \wedge p) \rightarrow q$ is a tautology. In particular, when both $p \rightarrow q$ and p are true, we know that q must also be true. We say this form of argument is **valid** because whenever all its premises (all statements in the argument other than the final one, the conclusion) are true, the conclusion must also be true. Now suppose that both "If you have a current password, then you can log onto the network" and "You have

a current password" are true statements. When we replace p by "You have a current password" and q by "You can log onto the network," it necessarily follows that the conclusion "You can log onto the network" is true. This argument is valid because its form is valid. Note that whenever we replace p and q by propositions where $p \rightarrow q$ and p are both true, then q must also be true.

What happens when we replace p and q in this argument form by propositions where not both p and $p \rightarrow q$ are true? For example, suppose that p represents "You have access to the network" and q represents "You can change your grade" and that p is true, but $p \rightarrow q$ is false. The argument we obtain by substituting these values of p and q into the argument form is

"If you have access to the network, then you can change your grade."
"You have access to the network."

$\therefore$ "You can change your grade."

The argument we obtained is a valid argument, but because one of the premises, namely the first premise, is false, we cannot conclude that the conclusion is true. (Most likely, this conclusion is false.)

In our discussion, to analyze an argument, we replaced propositions by propositional variables. This changed an argument to an **argument form**. We saw that the validity of an argument follows from the validity of the form of the argument. We summarize the terminology used to discuss the validity of arguments with our definition of the key notions.

Definition 1.6.1 *An **argument** in propositional logic is a sequence of propositions. All but the final proposition in the argument are called **premises** and the final proposition is called the conclusion. An argument is **valid** if the truth of all its premises implies that the conclusion is true.*

*An **argument** form in propositional logic is a sequence of compound propositions involving propositional variables. An argument form is **valid** no matter which particular propositions are substituted for the propositional variables in its premises, the conclusion is true if the premises are all true.*

From the definition of a valid argument form we see that the argument form with premises $p_1, p_2, \ldots, p_n$ and conclusion q is valid, when $(p_1 \wedge p_2 \wedge \ldots \wedge p_n) \rightarrow q$ is a tautology.

The key to showing that an argument in propositional logic is valid is to show that its argument form is valid. Consequently, we would like techniques to show that argument forms are valid.We will now develop methods for accomplishing this task.

1.6.3 Rules of Inference for Propositional Logic

We can always use a truth table to show that an argument form is valid. We do this by showing that whenever the premises are true, the conclusion must also be true. However, this can be a tedious approach. For example, when an argument form involves 10 different propositional variables, to use a truth table to show this argument form is valid requires $2^{10} = 1024$ different rows. Fortunately, we do not have to resort to truth tables. Instead, we can first establish the validity of some relatively simple argument forms, called **rules of inference**. These rules of inference can be used as building blocks to construct more complicated valid argument forms. We will now introduce the most important rules of inference in propositional logic.

The tautology $(p \wedge (p \rightarrow q)) \rightarrow q$ is the basis of the rule of inference called modus ponens, or the law of detachment. (Modus ponens is Latin for mode that affirms.) This tautology leads to the following valid argument form, which we have already seen in our initial discussion about arguments:

$$\begin{array}{l} p \\ p \rightarrow q \\ \hline \therefore\ q \end{array}$$

Using this notation, the hypotheses are written in a column, followed by a horizontal bar, followed by a line that begins with the therefore symbol and ends with the conclusion. In particular, modus ponens tells us that if a conditional statement and the hypothesis of this conditional statement are both true, then the conclusion must also be true. Example 1 illustrates the use of modus ponens.

EXAMPLE. 1

Suppose that the conditional statement "If it snows today, then we will go skiing" and its hypothesis, "It is snowing today," are true. Then, by modus ponens, it follows that the conclusion of the conditional statement, "We will go skiing," is true. ∎

As we mentioned earlier, a valid argument can lead to an incorrect conclusion if one or more of its premises is false. We illustrate this again in Example 2.

EXAMPLE. 2

Determine whether the argument given here is valid and determine whether its conclusion must be true because of the validity of the argument.

"If $\sqrt{2} > \frac{3}{2}$, then $(\sqrt{2})^2 > (\frac{3}{2})^2$. We know that $\sqrt{2} > \frac{3}{2}$. Consequently, $(\sqrt{2})^2 = 2 > (\frac{3}{2})^2 = \frac{9}{4}$."

Solution: Let p be the proposition "$\sqrt{2} > \frac{3}{2}$" and q the proposition $2 > (\frac{3}{2})^2$. The premises of the argument are $p \rightarrow q$ and p, and q is its conclusion. This argument is valid because it is constructed by using modus ponens, a valid argument form. However, one of its premises, $\sqrt{2} > \frac{3}{2}$, is false. Consequently, we cannot conclude that the conclusion is true. Furthermore, note that the conclusion of this argument is false, because $2 < \frac{9}{4}$.

There are many useful rules of inference for propositional logic. We now give examples of arguments that use these rules of inference. In each argument, we first use propositional variables to express the propositions in the argument. We then show that the resulting argument form is a rule of inference from Table 1.21.

EXAMPLE. 3

State which rule of inference is the basis of the following argument: "It is below freezing now. Therefore, it is either below freezing or raining now."

Solution: Let p be the proposition "It is below freezing now" and q the proposition "It is raining now." Then this argument is of the form

Table 1.21: Rules of Inference.

Rule of Inference	**Tautology**	**Name**
p $p \to q$ $\therefore \quad q$	$(p \wedge (p \to q)) \to q$	Modus ponens
$\neg q$ $p \to q$ $\therefore \quad \neg p$	$(\neg q \wedge (p \to q)) \to \neg p$	Modus tollens
$p \to q$ $q \to r$ $\therefore \quad p \to r$	$((p \to q) \wedge (q \to r)) \to (p \to r)$	Hypothetical syllogism
$p \vee q$ $\neg p$ $\therefore \quad q$	$((p \vee q) \wedge \neg p) \to q$	Disjunctive syllogism
p $\therefore \quad p \vee q$	$p \to (p \vee q)$	Addition
$p \wedge q$ $\therefore \quad p$	$(p \wedge q) \to p$	Simplification
p q $\therefore \quad p \wedge q$	$((p) \wedge (q)) \to (p \wedge q)$	Conjunction
$p \vee q$ $\neg p \vee r$ $\therefore \quad q \vee r$	$((p \vee q) \wedge (\neg p \vee r)) \to (q \vee r)$	Resolution

$$\begin{array}{l} p \\ \hline \therefore\ p \vee q \end{array}$$

This is an argument that uses the addition rule. ∎

EXAMPLE. 4

State which rule of inference is the basis of the following argument: "It is below freezing and raining now. Therefore, it is below freezing now."

Solution: Let p be the proposition "It is below freezing now," and let q be the proposition "It is raining now." This argument is of the form

$$\begin{array}{l} p \wedge q \\ \hline \therefore\ p \end{array}$$

This argument uses the simplification rule. ∎

EXAMPLE. 5

State which rule of inference is used in the argument:

If it rains today, then we will not have a barbecue today. If we do not have a barbecue today, then we will have a barbecue tomorrow. Therefore, if it rains today, then we will have a barbecue tomorrow.

Solution: Let p be the proposition "It is raining today," let q be the proposition "We will not have a barbecue today," and let r be the proposition "We will have a barbecue tomorrow." Then this argument is of the form

$$\begin{array}{l} p \rightarrow q \\ q \rightarrow r \\ \hline \therefore\ p \rightarrow r \end{array}$$

Hence, this argument is a hypothetical syllogism. ∎

1.6.4 Using Rules of Inference to Build Arguments

When there are many premises, several rules of inference are often needed to show that an argument is valid. This is illustrated by Examples 6 and 7, where the steps of arguments are displayed on separate

lines, with the reason for each step explicitly stated. These examples also show how arguments in English can be analyzed using rules of inference.

EXAMPLE. 6

Show that the premises "It is not sunny this afternoon and it is colder than yesterday", "We will go swimming only if it is sunny", "If we do not go swimming, then we will take a canoe trip", and "If we take a canoe trip, then we will be home by sunset" lead to the conclusion "We will be home by sunset".

Solution: Let p be the proposition "It is sunny this afternoon", q the proposition "It is colder than yesterday", r the proposition "We will go swimming", s the proposition "We will take a canoe trip", and t the proposition "We will be home by sunset". Then the premises become $\neg p \wedge q$, $r \rightarrow p$, $\neg r \rightarrow s$, and $s \rightarrow t$. The conclusion is simply t. We need to give a valid argument with premises $\neg p \wedge q$, $r \rightarrow p$, $\neg r \rightarrow s$, and $s \rightarrow t$ and conclusion t.

We construct an argument to show that our premises lead to the desired conclusion as follows.

Step	**Reason**
1. $\neg p \wedge q$	Premise
2. $\neg p$	Simplification using (1)
3. $r \rightarrow p$	Premise
4. $\neg r$	Modus tollens using (2) and (3)
5. $\neg r \rightarrow s$	Premise
6. s	Modus ponens using (4) and (5)
7. $s \rightarrow t$	Premise
8. t	Modus ponens using (6) and (7)

Note that we could have used a truth table to show that whenever each of the four hypotheses is true, the conclusion is also true. However, because we are working with five propositional variables, p, q, r, s, and t , such a truth table would have 32 rows. ∎

EXAMPLE. 7

Show that the premises "If you send me an e-mail message, then I will finish writing the program", "If you do not send me an e-mail message, then I will go to sleep early", and "If I go to sleep early, then I will wake up feeling refreshed" lead to the conclusion "If I do not finish writing the program, then I will wake

up feeling refreshed."

Solution: Let p be the proposition "You send me an e-mail message", q the proposition "I will finish writing the program", r the proposition "I will go to sleep early", and s the proposition "I willwake up feeling refreshed". Then the premises are $p \rightarrow q$, $\neg p \rightarrow r$, and $r \rightarrow s$. The desired conclusion is $\neg q \rightarrow s$. We need to give a valid argument with premises $p \rightarrow q$, $\neg p \rightarrow r$, and $r \rightarrow s$ and conclusion $\neg q \rightarrow s$.

This argument form shows that the premises lead to the desired conclusion.

Step	**Reason**
1. $p \rightarrow q$	Premise
2. $\neg q \rightarrow \neg p$	Contrapositive of (1)
3. $\neg p \rightarrow r$	Premise
4. $\neg q \rightarrow r$	Hypothetical syllogism using (2) and (3)
5. $r \rightarrow s$	Premise
6. $\neg q \rightarrow s$	Hypothetical syllogism using (4) and (5) ∎

1.6.5 Resolution

Computer programs have been developed to automate the task of reasoning and proving theorems. Many of these programs make use of a rule of inference known as **resolution**. This rule of inference is based on the tautology

$$((p \vee q) \wedge (\neg p \vee r)) \rightarrow (q \vee r).$$

The final disjunction in the resolution rule, $q \vee r$, is called the **resolvent**. When we let $q = r$ in this tautology, we obtain $(p \vee q) \wedge (\neg p \vee q) \rightarrow q$. Furthermore, when we let $r = \mathbf{F}$, we obtain $(p \vee q) \wedge (\neg p) \rightarrow q$ (because $q \vee \mathbf{F} \equiv q$), which is the tautology on which the rule of disjunctive syllogism is based.

EXAMPLE. 8

Use resolution to show that the hypotheses "Jasmine is skiing or it is not snowing" and "It is snowing or Bart is playing hockey" imply that "Jasmine is skiing or Bart is playing hockey."

Solution: Let p be the proposition "It is snowing", q the proposition "Jasmine is skiing", and r the proposition "Bart is playing hockey". We can represent the hypotheses as $\neg p \vee q$ and $p \vee r$, respectively. Using resolution, the

proposition $q \vee r$, "Jasmine is skiing or Bart is playing hockey", follows. ∎

Resolution plays an important role in programming languages based on the rules of logic, such as Prolog (where resolution rules for quantified statements are applied). Furthermore, it can be used to build automatic theorem proving systems. To construct proofs in propositional logic using resolution as the only rule of inference, the hypotheses and the conclusion must be expressed as **clauses**, where a clause is a disjunction of variables or negations of these variables. We can replace a statement in propositional logic that is not a clause by one or more equivalent statements that are clauses. For example, suppose we have a statement of the form $p \vee (q \wedge r)$. Because $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$, we can replace the single statement $p \vee (q \wedge r)$ by two statements $p \vee q$ and $p \vee r$, each of which is a clause. We can replace a statement of the form $\neg(p \vee q)$ by the two statements $\neg p$ and $\neg q$ because De Morgan's law tells us that $\neg(p \vee q) \equiv \neg p \wedge \neg q$. We can also replace a conditional statement $p \rightarrow q$ with the equivalent disjunction $\neg p \vee q$.

EXAMPLE. 9

Show that the premises $(p \wedge q) \vee r$ and $r \rightarrow s$ imply the conclusion $p \vee s$.

Solution: We can rewrite the premises $(p \wedge q) \vee r$ as two clauses, $p \vee r$ and $q \vee r$. We can also replace $r \rightarrow s$ by the equivalent clause $\neg r \vee s$. Using the two clauses $p \vee r$ and $\neg r \vee s$, we can use resolution to conclude $p \vee s$.

1.6.6 Fallacies

Several common fallacies arise in incorrect arguments. These fallacies resemble rules of inference, but are based on contingencies rather than tautologies. These are discussed here to show the distinction between correct and incorrect reasoning.

The proposition $((p \rightarrow q) \wedge q) \rightarrow p$ is not a tautology, because it is false when p is false and q is true. However, there are many incorrect arguments that treat this as a tautology. In other words, they treat the argument with premises $p \rightarrow q$ and q and conclusion p as a valid

argument form, which it is not. This type of incorrect reasoning is called the **fallacy of affirming the conclusion.**

EXAMPLE. 10

Is the following argument valid?

If you do every problem in this book, then you will learn discrete mathematics.You learned discrete mathematics.

Therefore, you did every problem in this book.

Solution: Let p be the proposition "You did every problem in this book." Let q be the proposition "You learned discrete mathematics". Then this argument is of the form: if $p \rightarrow q$ and q, then p. This is an example of an incorrect argument using the fallacy of affirming the conclusion. Indeed, it is possible for you to learn discrete mathematics in someway other than by doing every problem in this book. (You may learn discrete mathematics by reading, listening to lectures, doing some, but not all, the problems in this book, and so on.) ∎

The proposition $((p \rightarrow q) \wedge \neg p) \rightarrow \neg q$ is not a tautology, because it is false when p is false and q is true. Many incorrect arguments use this incorrectly as a rule of inference. This type of incorrect reasoning is called the **fallacy of denying the hypothesis.**

EXAMPLE. 11

Let p and q be as in Example 10. If the conditional statement $p \rightarrow q$ is true, and $\neg p$ is true, is it correct to conclude that $\neg q$ is true? In other words, is it correct to assume that you did not learn discrete mathematics if you did not do every problem in the book, assuming that if you do every problem in this book, then you will learn discrete mathematics?

Solution: It is possible that you learned discrete mathematics even if you did not do every problem in this book. This incorrect argument is of the form $p \rightarrow q$ and $\neg p$ imply $\neg q$, which is an example of the fallacy of denying the hypothesis. ∎

1.6.7 Rules of Inference for Quantified Statements

We have discussed rules of inference for propositions.We will nowdescribe some important rules of inference for statements involving quantifiers. These rules of inference are used extensively in mathematical arguments, often without being explicitly mentioned.

Universal instantiation is the rule of inference used to conclude that *P(c)* is true, where c is a particular member of the domain, given the premise $\forall x P(x)$. Universal instantiation is used when we conclude from the statement "All women are wise" that "Lisa is wise", where Lisa is a member of the domain of all women.

Universal generalization is the rule of inference that states that $\forall x P(x)$ is true, given the premise that *P(c)* is true for all elements c in the domain. Universal generalization is used when we show that $\forall x P(x)$ is true by taking an arbitrary element c from the domain and showing that *P(c)* is true. The element c that we select must be an arbitrary, and not a specific, element of the domain. That is, when we assert from $\forall x P(x)$ the existence of an element c in the domain, we have no control over c and cannot make any other assumptions about c other than it comes from the domain. Universal generalization is used implicitly in many proofs in mathematics and is seldom mentioned explicitly. However, the error of adding unwarranted assumptions about the arbitrary element c when universal generalization is used is all too common in incorrect reasoning.

Existential instantiation is the rule that allows us to conclude that there is an element c in the domain for which *P(c)* is true if we know that $\exists x P(x)$ is true. We cannot select an arbitrary value of c here, but rather it must be a c for which *P(c)* is true. Usually we have no knowledge of what c is, only that it exists. Because it exists, we may give it a name *(c)* and continue our argument.

Existential generalization is the rule of inference that is used to conclude that $\exists x P(x)$ is true when a particular element c with *P(c)* true is known. That is, if we know one element c in the domain for which *P(c)* is true, then we know that $\exists x P(x)$ is true.

We summarize these rules of inference in Table 1.22. We will illustrate how some of these rules of inference for quantified statements are used in Examples 12 and 13.

Table 1.22: Rules of Inference for Quantified Statements.

Rule of Inference	**Name**
$\forall x P(x)$ $\therefore\ P(c)$	Universal instantiation
$P(c)$ for an arbitary c $\therefore\ \forall x P(x)$	Universal generalization
$\exists x P(x)$ $\therefore\ P(c)$ for some element c	Existential instantiation
$P(c)$ for some element c $\therefore\ \exists x P(x)$	Existential generalization

EXAMPLE. 12

Show that the premises "Everyone in this discrete mathematics class has taken a course in computer science" and "Marla is a student in this class" imply the conclusion "Marla has taken a course in computer science."

Solution: Let *D(x)* denote "x is in this discrete mathematics class", and let *C(x)* denote "x has taken a course in computer science". Then the premises are $\forall x(D(x) \rightarrow C(x))$ and D(Marla). The conclusion is C(Marla).

The following steps can be used to establish the conclusion from the premises.

Step	**Reason**
1. $\forall x(D(x) \rightarrow C(x))$	Premise
2. D(Marla)$\rightarrow C$(Marla)	Universal instantiation from (1)
3. D(Marla)	Premise
4. C(Marla)	Modus ponens from (2) and (3) ∎

EXAMPLE. 13

Show that the premises "A student in this class has not read the book", and "Everyone in this class passed the first exam" imply the conclusion "Someone who passed the first exam has not read the book".

Solution: Let *C(x)* be "x is in this class", *B(x)* be "x has read the book", and *P(x)* be "x passed the first exam". The premises are $\exists x(C(x) \wedge \neg B(x))$ and $\forall x(C(x) \rightarrow P(x))$. The conclusion is $\exists x(P(x) \wedge \neg B(x))$. These steps can be used to establish the conclusion from the premises.

Step	**Reason**
1. $\exists x(C(x) \wedge \neg B(x))$	Premise
2. $C(a) \wedge \neg B(a)$	Existential instantiation from (1)
3. $C(a)$	Simplification from (2)
4. $\forall x(C(x) \rightarrow P(x))$	Premise
5. $C(a) \rightarrow P(a)$	Universal instantiation from (4)
6. $P(a)$	Modus ponens from (3) and (5)
7. $\neg B(a)$	Simplification from (2)
8. $P(a) \wedge \neg B(a)$	Conjunction from (6) and (7)
9. $\exists x(P(x) \wedge \neg B(x))$	Existential generalization from (8)

■

1.6.8 Combining Rules of Inference for Propositions and Quantified Statements

We have developed rules of inference both for propositions and for quantified statements. Note that in our arguments in Examples 12 and 13 we used both universal instantiation, a rule of inference for quantified statements, and modus ponens, a rule of inference for propositional logic. We will often need to use this combination of rules of inference. Because universal instantiation and modus ponens are used so often together, this combination of rules is sometimes called **universal modus ponens**. This rule tells us that if $\forall x(P(x) \rightarrow Q(x))$ is true, and if $P(a)$ is true for a particular element a in the domain of the universal quantifier, then $Q(a)$ must also be true. To see this, note that by universal instantiation, $P(a) \rightarrow Q(a)$ is true. Then, by modus ponens, $Q(a)$ must also be true.We can describe universal modus ponens as follows:

$\forall x(P(x) \rightarrow Q(x))$
$P(a)$ where a is a particular element in the domain
$\therefore$ Q(a)

Universal modus ponens is commonly used in mathematical arguments. This is illustrated in Example 14.

EXAMPLE. 14

Assume that "For all positive integers n, if n is greater than 4, then n^2 is less than 2^n" is true. Use universal modus ponens to show that $100^2 < 2^{100}$.

Solution: Let $P(n)$ denote "$n > 4$" and $Q(n)$ denote "$n^2 < 2^n$". The statement "For all positive integers n, if n is greater than 4, then n^2 is less than 2^n" can be represented by $\forall n(P(n) \rightarrow Q(n))$, where the domain consists of all positive integers. We are assuming that $\forall n(P(n) \rightarrow Q(n))$ is true. Note that $P(100)$ is true because $100 > 4$. It follows by universal modus ponens that $Q(100)$ is true, namely that $100^2 < 2^{100}$. ∎

Another useful combination of a rule of inference from propositional logic and a rule of inference for quantified statements is **universal modus tollens**. Universal modus tollens combines universal instantiation and modus tollens and can be expressed in the following way:

$$\begin{array}{ll} & \forall x(P(x) \rightarrow Q(x)) \\ & \neg Q(a) \text{ where } a \text{ is a particular element in the domain} \\ \hline \therefore & \neg P(a) \end{array}$$

1.7 Introduction to Proofs

1.7.1 Introduction

In this section we introduce the notion of a proof and describe methods for constructing proofs. A proof is a valid argument that establishes the truth of a mathematical statement. A proof can use the hypotheses of the theorem, if any, axioms assumed to be true, and previously proven theorems. Using these ingredients and rules of inference, the final step of the proof establishes the truth of the statement being proved.

In our discussion we move from formal proofs of theorems toward more informal proofs. The arguments we introduced in Section 1.6 to show that statements involving propositions and quantified statements

are true were formal proofs, where all steps were supplied, and the rules for each step in the argument were given. However, formal proofs of useful theorems can be extremely long and hard to follow. In practice, the proofs of theorems designed for human consumption are almost always **informal proofs,** where more than one rule of inference may be used in each step, where steps may be skipped, where the axioms being assumed and the rules of inference used are not explicitly stated. Informal proofs can often explain to humans why theorems are true, while computers are perfectly happy producing formal proofs using automated reasoning systems.

The methods of proof discussed in this chapter are important not only because they are used to prove mathematical theorems, but also for their many applications to computer science. These applications include verifying that computer programs are correct, establishing that operating systems are secure, making inferences in artificial intelligence, showing that system specifications are consistent, and so on. Consequently, understanding the techniques used in proofs is essential both in mathematics and in computer science.

1.7.2 Some Terminology

Formally, a **theorem** is a statement that can be shown to be true. In mathematical writing, the term theorem is usually reserved for a statement that is considered at least somewhat important. Less important theorems sometimes are called **propositions**. (Theorems can also be referred to as **facts** or **results**.) A theorem may be the universal quantification of a conditional statement with one or more premises and a conclusion. However, it may be some other type of logical statement, as the examples later in this chapter will show.We demonstrate that a theorem is true with a **proof**. A proof is a valid argument that establishes the truth of a theorem. The statements used in a proof can include **axioms** (or **postulates**), which are statements we assume to be true (for example, the axioms for the real numbers, given in Appendix 1, and the axioms of plane geometry), the premises, if any, of the theorem, and previously proven theorems. Axioms may be stated using primitive terms that do not require definition, but all other terms used in theorems and their proofs must be defined. Rules of inference,

together with definitions of terms, are used to draw conclusions from other assertions, tying together the steps of a proof. In practice, the final step of a proof is usually just the conclusion of the theorem. However, for clarity, we will often recap the statement of the theorem as the final step of a proof.

A less important theorem that is helpful in the proof of other results is called a **lemma** (plural *lemmas or lemmata*). Complicated proofs are usually easier to understand when they are proved using a series of lemmas, where each lemma is proved individually. A **corollary** is a theorem that can be established directly from a theorem that has been proved. A **conjecture** is a statement that is being proposed to be a true statement, usually on the basis of some partial evidence, a heuristic argument, or the intuition of an expert. When a proof of a conjecture is found, the conjecture becomes a theorem. Many times conjectures are shown to be false, so they are not theorems.

1.7.3 Understanding How Theorems Are Stated

Before we introduce methods for proving theorems, we need to understand how many mathematical theorems are stated. Many theorems assert that a property holds for all elements in a domain, such as the integers or the real numbers. Although the precise statement of such theorems needs to include a universal quantifier, the standard convention in mathematics is to omit it. For example, the statement

"If $x > y$, where x and y are positive real numbers, then $x^2 > y^2$".

really means

"For all positive real numbers x and y, if $x > y$, then $x^2 > y^2$".

Furthermore, when theorems of this type are proved, the first step of the proof usually involves selecting a general element of the domain. Subsequent steps show that this element has the property in question. Finally, universal generalization implies that the theorem holds for all members of the domain.

1.7.4 Methods of Proving Theorems

Proving mathematical theorems can be difficult. To construct proofs we need all available ammunition, including a powerful battery of dif-

ferent proof methods. These methods provide the overall approach and strategy of proofs. Understanding these methods is a key component of learning how to read and construct mathematical proofs. One we have chosen a proof method, we use axioms, definitions of terms, previously proved results, and rules of inference to complete the proof. Note that in this book we will always assume the axioms for real numbers found in Appendix 1.We will also assume the usual axioms whenever we prove a result about geometry. When you construct your own proofs, be careful not to use anything but these axioms, definitions, and previously proved results as facts!

To prove a theorem of the form $\forall x(P(x) \rightarrow Q(x))$, our goal is to show that $P(c) \rightarrow Q(c)$ is true, where c is an arbitrary element of the domain, and then apply universal generalization. In this proof, we need to show that a conditional statement is true. Because of this, we now focus on methods that show that conditional statements are true. Recall that $p \rightarrow q$ is true unless p is true but q is false. Note that to prove the statement $p \rightarrow q$, we need only show that q is true if p is true. The following discussion will give the most common techniques for proving conditional statements. Later we will discuss methods for proving other types of statements. In this section we will develop a large arsenal of proof techniques that can be used to prove a wide variety of theorems.

When you read proofs, you will often find the words "obviously" or "clearly". These words indicate that steps have been omitted that the author expects the reader to be able to fill in. Unfortunately, this assumption is often not warranted and readers are not at all sure how to fill in the gaps. We will assiduously try to avoid using these words and try not to omit too many steps. However, if we included all steps in proofs, our proofs would often be excruciatingly long.

1.7.5 Direct Proofs

A **direct proof** of a conditional statement $p \rightarrow q$ is constructed when the first step is the assumption that p is true; subsequent steps are constructed using rules of inference, with the final step showing that q must also be true. A direct proof shows that a conditional statement $p \rightarrow q$ is true by showing that if p is true, then q must also be true, so

that the combination p true and q false never occurs. In a direct proof, we assume that p is true and use axioms, definitions, and previously proven theorems, together with rules of inference, to show that q must also be true. You will find that direct proofs of many results are quite straightforward, with a fairly obvious sequence of steps leading from the hypothesis to the conclusion. However, direct proofs sometimes require particular insights and can be quite tricky. The first direct proofs we present here are quite straightforward; later in the text you will see some that are less obvious.

We will provide examples of several different direct proofs. Before we give the first example, we need to define some terminology

Definition 1.7.1 *The integer n is **even** if there exists an integer k such that $n = 2k$, and n is **odd** if there exists an integer k such that $n = 2k+1$. (Note that every integer is either even or odd, and no integer is both even and odd.) Two integers have the **same parity** when both are even or both are odd; they have **opposite** parity when one is even and the other is odd.*

EXAMPLE. 1

Give a direct proof of the theorem "If n is an odd integer, then n^2 is odd".

Solution: Note that this theorem states $\forall n P((n) \rightarrow Q(n))$, where *P(n)* is "$n$ is an odd integer" and *Q(n)* is "n^2 is odd". As we have said, we will follow the usual convention in mathematical proofs by showing that *P(n)* implies *Q(n)*, and not explicitly using universal instantiation. To begin a direct proof of this theorem, we assume that the hypothesis of this conditional statement is true, namely, we assume that n is odd. By the definition of an odd integer, it follows that $n = 2k + 1$, where k is some integer. We want to show that n^2 is also odd. We can square both sides of the equation $n = 2k + 1$ to obtain a new equation that expresses n^2. When we do this, we find that $n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$. By the definition of an odd integer, we can conclude that n^2 is an odd integer (it is one more than twice an integer). Consequently, we have proved that if n is an odd integer, then n^2 is an odd integer. ∎

EXAMPLE. 2

Give a direct proof that if m and n are both perfect squares, then nm is also a perfect square. (An integer a is a **perfect square** if there is an integer b such that $a = b^2$.)

Solution: To produce a direct proof of this theorem, we assume that the hypothesis of this conditional statement is true, namely, we assume that m and n are both perfect squares. By the definition of a perfect square, it follows that there are integers s and t such that $m = s^2$ and $n = t^2$. The goal of the proof is to show that mn must also be a perfect square when m and n are; looking ahead we see how we can show this by substituting s^2 for m and t^2 for n into mn. This tells us that $mn = s^2t^2$. Hence, $mn = s^2t^2 = (ss)(tt) = (st)(st) = (st)^2$, using commutativity and associativity of multiplication. By the definition of perfect square, it follows that mn is also a perfect square, because it is the square of st, which is an integer. We have proved that if m and n are both perfect squares, then mn is also a perfect square. ∎

1.7.6 Proof by Contraposition

Direct proofs lead from the premises of a theorem to the conclusion. They begin with the premises, continue with a sequence of deductions, and end with the conclusion. However, we will see that attempts at direct proofs often reach dead ends.We need other methods of proving theorems of the form $\forall x(P(x) \rightarrow Q(x))$. Proofs of theorems of this type that are not direct proofs, that is, that do not start with the premises and end with the conclusion, are called indirect proofs.

An extremely useful type of indirect proof is known as proof by contraposition. Proofs by contraposition make use of the fact that the conditional statement $p \rightarrow q$ is equivalent to its contrapositive, $\neg q \rightarrow \neg p$. This means that the conditional statement $p \rightarrow q$ can be proved by showing that its contrapositive, $\neg q \rightarrow \neg p$, is true. In a proof by contraposition of $p \rightarrow q$, we take $\neg q$ as a premise, and using axioms, definitions, and previously proven theorems, together with rules of inference, we show that $\neg p$ must follow. We will illustrate proof by

contraposition with two examples. These examples show that proof by contraposition can succeed when we cannot easily find a direct proof.

EXAMPLE. 3

Prove that if n is an integer and $3n + 2$ is odd, then n is odd.

Solution: We first attempt a direct proof. To construct a direct proof, we first assume that $3n + 2$ is an odd integer. This means that $3n + 2 = 2k + 1$ for some integer k. Can we use this fact to show that n is odd? We see that $3n + 1 = 2k$, but there does not seem to be any direct way to conclude that n is odd. Because our attempt at a direct proof failed, we next try a proof by contraposition.

The first step in a proof by contraposition is to assume that the conclusion of the conditional statement "If $3n + 2$ is odd, then n is odd" is false; namely, assume that n is even. Then, by the definition of an even integer, $n = 2k$ for some integer k. Substituting $2k$ for n, we find that $3n + 2 = 3(2k) + 2 = 6k + 2 = 2(3k + 1)$. This tells us that $3n + 2$ is even (because it is a multiple of 2), and therefore not odd. This is the negation of the premise of the theorem. Because the negation of the conclusion of the conditional statement implies that the hypothesis is false, the original conditional statement is true. Our proof by contraposition succeeded; we have proved the theorem "If $3n + 2$ is odd, then n is odd". ∎

EXAMPLE. 4

Prove that if $= ab$, where a and b are positive integers, then $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$.

Solution: Because there is no obvious way of showing that $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$ directly from the equation $n = ab$, where a and b are positive integers, we attempt a proof by contraposition.

The first step in a proof by contraposition is to assume that the conclusion of the conditional statement "If $n = ab$, where a and b are positive integers, then $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$" is false. That is, we assume that the statement $(a \leq \sqrt{n}) \vee (b \leq \sqrt{n})$ is false. Using the meaning of disjunction together with De Morgan's law, we see that this implies that both $a \leq \sqrt{n}$ and $b \leq \sqrt{n}$ are false. This implies that $a > \sqrt{n}$ and $b > \sqrt{n}$. We can multiply these inequalities together (using the fact that if $0 < s < t$ and $0 < u < v$, then

$su < tv$) to obtain $ab > \sqrt{n} \cdot \sqrt{n} = n$. This shows that $ab \neq n$, which contradicts the statement $n = ab$.

Because the negation of the conclusion of the conditional statement implies that the hypothesis is false, the original conditional statement is true. Our proof by contraposition succeeded; we have proved that if $n = ab$, where a and b are positive integers, then $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$. ∎

VACUOUS AND TRIVIAL PROOFS

We can quickly prove that a conditional statement $p \rightarrow q$ is true when we know that p is false, because $p \rightarrow q$ must be true when p is false. Consequently, if we can show that p is false, then we have a proof, called a vacuous proof, of the conditional statement $p \rightarrow q$. Vacuous proofs are often used to establish special cases of theorems that state that a conditional statement is true for all positive integers.

EXAMPLE. 5

Show that the proposition *P(0)* is true, where *P(n)* is "If $n > 1$, then $n^2 > n$" and the domain consists of all integers.

Solution: Note that *P(0)* is "If $0 > 1$, then $0^2 > 0$". We can show *P(0)* using a vacuous proof. Indeed, the hypothesis $0 > 1$ is false. This tells us that *P(0)* is automatically true. ∎

> **Remark!** The fact that the conclusion of this conditional statement, $0^2 > 0$, is false is irrelevant to the truth value of the conditional statement, because a conditional statement with a false hypothesis is guaranteed to be true.

We can also quickly prove a conditional statement $p \rightarrow q$ if we know that the conclusion q is true. By showing that q is true, it follows that $p \rightarrow q$ must also be true. A proof of $p \rightarrow q$ that uses the fact that q is true is called a trivial proof. Trivial proofs are often important when special cases of theorems are proved and in mathematical induction, which is a proof technique discussed in Section 5.1.

EXAMPLE. 6

Let *P(n)* be "If a and b are positive integers with $a \geq b$, then $a^n \geq b^n$", where the domain consists of all nonnegative integers. Show that *P(0)* is true.

Solution: The proposition *P(0)* is "If $a \geq b$, then $a^0 \geq b^0$". Because $a^0 = b^0 = 1$, the conclusion of the conditional statement "If $a \geq b$, then $a^0 \geq b^0$" is true. Hence, this conditional statement, which is *P(0)*, is true. This is an example of a trivial proof. Note that the hypothesis, which is the statement "$a \geq b$", was not needed in this proof. ∎

A LITTLE PROOF STRATEGY

We have described two important approaches for proving theorems of the form $\forall x(P(x) \rightarrow Q(x))$: direct proof and proof by contraposition. We have also given examples that show how each is used. However, when you are presented with a theorem of the form $\forall x(P(x) \rightarrow Q(x))$, which method should you use to attempt to prove it? We will provide a fewrules of thumb here; in Section 1.8 we will discuss proof strategy at greater length. When you want to prove a statement of the form $\forall x(P(x) \rightarrow Q(x))$, first evaluate whether a direct proof looks promising. Begin by expanding the definitions in the hypotheses. Start to reason using these hypotheses, together with axioms and available theorems. If a direct proof does not seem to go anywhere, try the same thing with a proof by contraposition. Recall that in a proof by contraposition you assume that the conclusion of the conditional statement is false and use a direct proof to show this implies that the hypothesis must be false. We illustrate this strategy in Examples 7 and 8. Before we present our next example, we need a definition.

Definition 1.7.2 *The real number r is **rational** if there exist integers p and q with $q \neq 0$ such that $r = p/q$. A real number that is not rational is called **irrational**.*

EXAMPLE. 7

Prove that the sum of two rational numbers is rational. (Note that if we include the implicit quantifiers here, the theorem we want to prove is "For every real number r and every real number s, if r and s are rational numbers, then $r + s$ is rational".)

Solution: We first attempt a direct proof.To begin, suppose that r and s are rational numbers. From the definition of a rational number, it follows that there are integers p and q, with $q \neq 0$, such that $r = p/q$, and integers t and u, with $u \neq 0$, such that $s = t/u$. Can we use this information to show that $r + s$ is rational? The obvious next step is to add $r = p/q$ and $s = t/u$, to obtain

$$r + s = \frac{p}{q} + \frac{t}{u} = \frac{pu + qt}{qu}.$$

Because $q \neq 0$ and $u \neq 0$, it follows that $qu \neq 0$. Consequently, we have expressed $r + s$ as the ratio of two integers, $pu + qt$ and qu, where $qu \neq 0$. This means that $r+s$ is rational.We have proved that the sum of two rational numbers is rational; our attempt to find a direct proof succeeded. ∎

EXAMPLE. 8

Prove that if n is an integer and n^2 is odd, then n is odd.

Solution: We first attempt a direct proof. Suppose that n is an integer and n^2 is odd. Then, there exists an integer k such that $n^2 = 2k + 1$. Can we use this information to show that n is odd? There seems to be no obvious approach to show that n is odd because solving for n produces the equation $n = \pm\sqrt{2k + 1}$, which is not terribly useful.

Because this attempt to use a direct proof did not bear fruit, we next attempt a proof by contraposition. We take as our hypothesis the statement that n is not odd. Because every integer is odd or even, this means that n is even. This implies that there exists an integer k such that $n = 2k$. To prove the theorem, we need to show that this hypothesis implies the conclusion that n^2 is not odd, that is, that n^2 is even. Can we use the equation $n = 2k$ to achieve this? By squaring both sides of this equation, we obtain $n^2 = 4k^2 = 2(2k^2)$, which implies that n^2 is also even because $n^2 = 2t$, where $t = 2k^2$. We have proved that if n is an integer and n^2 is odd, then n is odd. Our attempt to find a proof by contraposition succeeded. ∎

1.7.7 Proofs by Contradiction

Suppose we want to prove that a statement p is true. Furthermore, suppose that we can find a contradiction q such that $\neg p \rightarrow q$ is true.

Because q is false, but $\neg p \rightarrow q$ is true, we can conclude that $\neg p$ is false, which means that p is true. How can we find a contradiction q that might help us prove that p is true in this way?

Because the statement r$\wedge\neg r$ is a contradiction whenever r is a proposition, we can prove that p is true if we can show that $\neg p \rightarrow (r \wedge \neg r)$ is true for some proposition r. Proofs of this type are called **proofs by contradiction**. Because a proof by contradiction does not prove a result directly, it is another type of indirect proof. We provide three examples of proof by contradiction.

EXAMPLE. 9

Show that at least four of any 22 days must fall on the same day of the week.

Solution: Let p be the proposition "At least four of 22 chosen days fall on the same day of the week". Suppose that $\neg p$ is true. This means that at most three of the 22 days fall on the same day of the week. Because there are seven days of the week, this implies that at most 21 days could have been chosen, as for each of the days of the week, at most three of the chosen days could fall on that day. This contradicts the premise that we have 22 days under consideration. That is, if r is the statement that 22 days are chosen, then we have shown that $\neg p \rightarrow (r \wedge \neg r)$. Consequently, we know that p is true. We have proved that at least four of 22 chosen days fall on the same day of the week. ∎

EXAMPLE. 10

Prove that $\sqrt{2}$ is irrational by giving a proof by contradiction.

Solution: Let p be the proposition "$\sqrt{2}$ is irrational". To start a proof by contradiction, we suppose that $\neg p$ is true. Note that $\neg p$ is the statement "It is not the case that $\sqrt{2}$ is irrational", which says that $\sqrt{2}$ is rational. We will show that assuming that $\neg p$ is true leads to a contradiction.

If $\sqrt{2}$ is rational, there exist integers a and b with $\sqrt{2} = a/b$, where $b \neq 0$ and a and b have no common factors (so that the fraction a/b is in lowest terms.) (Here, we are using the fact that every rational number can be written in lowest terms.) Because $\sqrt{2} = a/b$, when both sides of this equation

are squared, it follows that

$$2 = \frac{a^2}{b^2}.$$

Hence

$$2b^2 = a^2.$$

By the definition of an even integer it follows that a^2 is even. We next use the fact that if a^2 is even, a must also be even. Furthermore, because a is even, by the definition of an even integer, $a = 2c$ for some integer c. Thus,

$$2b^2 = 4c^2.$$

Dividing both sides of this equation by 2 gives

$$b^2 = 2c^2.$$

By the definition of even, this means that b^2 is even. Again using the fact that if the square of an integer is even, then the integer itself must be even, we conclude that b must be even as well.

We have now shown that the assumption of $\neg p$ leads to the equation $\sqrt{2} = a/b$, where a and b have no common factors, but both a and b are even, that is, 2 divides both a and b. Note that the statement that $\sqrt{2} = a/b$, where a and b have no common factors, means, in particular, that 2 does not divide both a and b. Because our assumption of $\neg p$ leads to the contradiction that 2 divides both a and b and 2 does not divide both a and b, $\neg p$ must be false. That is, the statement p, "$\sqrt{2}$ is irrational", is true. We have proved that $\sqrt{2}$ is irrational. ∎

Proof by contradiction can be used to prove conditional statements. In such proofs, we first assume the negation of the conclusion. We then use the premises of the theorem and the negation of the conclusion to arrive at a contradiction. (The reason that such proofs are valid rests on the logical equivalence of $p \rightarrow q$ and $(p \wedge \neg q) \rightarrow \mathbf{F}$. To see that these statements are equivalent, simply note that each is false in exactly one case, namely when p is true and q is false.)

Note that we can rewrite a proof by contraposition of a conditional statement as a proof by contradiction. In a proof of $p \rightarrow q$ by contraposition, we assume that $\neg q$ is true. We then show that $\neg p$ must also be true. To rewrite a proof by contraposition of $p \rightarrow q$ as a proof by contradiction, we suppose that both p and $\neg q$ are true. Then, we use

the steps from the proof of $\neg q \rightarrow \neg p$ to show that $\neg p$ is true. This leads to the contradiction $p \wedge \neg p$, completing the proof. Example 11 illustrates how a proof by contraposition of a conditional statement can be rewritten as a proof by contradiction.

EXAMPLE. 11

Give a proof by contradiction of the theorem "If $3n + 2$ is odd, then n is odd".

Solution: Let p be "$3n + 2$ is odd" and q be "n is odd". To construct a proof by contradiction, assume that both p and $\neg q$ are true. That is, assume that $3n + 2$ is odd and that n is not odd. Because n is not odd, we know that it is even. Because n is even, there is an integer k such that $n = 2k$. This implies that $3n + 2 = 3(2k) + 2 = 6k + 2 = 2(3k + 1)$. Because $3n + 2$ is $2t$, where $t = 3k + 1$, $3n + 2$ is even. Note that the statement "$3n + 2$ is even" is equivalent to the statement $\neg p$, because an integer is even if and only if it is not odd. Because both p and $\neg p$ are true, we have a contradiction. This completes the proof by contradiction, proving that if $3n + 2$ is odd, then n is odd. ∎

PROOFS OF EQUIVALENCE

To prove a theorem that is a biconditional statement, that is, a statement of the form $p \leftrightarrow q$, we show that $p \rightarrow q$ and $q \rightarrow p$ are both true. The validity of this approach is based on the tautology

$$(p \leftrightarrow q) \leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p).$$

EXAMPLE. 12

Prove the theorem "If n is an integer, then n is odd if and only if n^2 is odd".

Solution: This theorem has the form "p if and only if q", where p is "n is odd" and q is "n^2 is odd". (As usual, we do not explicitly deal with the universal quantification.) To prove this theorem, we need to show that $p \rightarrow q$ and $q \rightarrow p$ are true.

We have already shown (in Example 1) that $p \rightarrow q$ is true and (in Example 8) that $q \rightarrow p$ is true.

Because we have shown that both $p \to q$ and $q \to p$ are true, we have shown that the theorem is true. ∎

Sometimes a theorem states that several propositions are equivalent. Such a theorem states that propositions $p_1, p_2, p_3, \ldots, p_n$ are equivalent. This can be written as

$$p_1 \leftrightarrow p_2 \leftrightarrow \ldots \leftrightarrow p_n,$$

which states that all n propositions have the same truth values, and consequently, that for all i and j with $1 \leq i \leq n$ and $1 \leq j \leq n$, p_i and p_j are equivalent. One way to prove these mutually equivalent is to use the tautology

$$p_1 \leftrightarrow p_2 \leftrightarrow \ldots \leftrightarrow p_n \leftrightarrow (p_1 \to p_2) \wedge (p_2 \to p_3) \wedge \ldots \wedge (p_n \to p_1).$$

This shows that if the n conditional statements $p_1 \to p_2$, $p_2 \to p_3, \ldots, p_n \to p_1$ can be shown to be true, then the propositions $p_1, p_2, \ldots, p_n$ are all equivalent.

This is much more efficient than proving that $p_i \to p_j$ for all $i \neq j$ with $1 \leq i \leq n$ and $1 \leq j \leq n$. (Note that there are $n^2 - n$ such conditional statements.)

When we prove that a group of statements are equivalent, we can establish any chain of conditional statements we choose as long as it is possible to work through the chain to go from any one of these statements to any other statement. For example, we can show that p_1, p_2, and p_3 are equivalent by showing that $p_1 \to p_3, p_3 \to p_2$, and $p_2 \to p_1$.

 EXAMPLE. 13

Show that these statements about the integer n are equivalent:

p_1 : n is even.
p_2 : $n - 1$ is odd.
p_3 : n^2 is even.

Solution: We will showthat these three statements are equivalent by showing that the conditional statements $p_1 \to p_2, p_2 \to p_3$, and $p_3 \to p_1$ are true.

We use a direct proof to show that $p_1 \rightarrow p_2$. Suppose that n is even. Then $n = 2k$ for some integer k. Consequently, $n-1 = 2k-1 = 2(k-1)+1$. This means that $n-1$ is odd because it is of the form $2m+1$, where m is the integer $k-1$.

We also use a direct proof to show that $p_2 \rightarrow p_3$. Now suppose $n-1$ is odd. Then $n-1 = 2k+1$ for some integer k. Hence, $n = 2k+2$ so that $n^2 = (2k+2)^2 = 4k^2+8k+4 = 2(2k^2+4k+2)$. This means that n^2 is twice the integer $2k^2+4k+2$, and hence is even.

To prove $p_3 \rightarrow p_1$, we use a proof by contraposition. That is, we prove that if n is not even, then n^2 is not even. This is the same as proving that if n is odd, then n^2 is odd, which we have already done in Example 1. This completes the proof. ∎

COUNTEREXAMPLES

In Section 1.4 we stated that to show that a statement of the form $\forall x P(x)$ is false, we need only find a counterexample, that is, an example x for which $P(x)$ is false. When presented with a statement of the form $\forall x P(x)$, which we believe to be false or which has resisted all proof attempts, we look for a counterexample. We illustrate the use of counterexamples in Example 14.

EXAMPLE. 14

Show that the statement "Every positive integer is the sum of the squares of two integers" is false.

Solution: To show that this statement is false, we look for a counterexample, which is a particular integer that is not the sum of the squares of two integers. It does not take long to find a counterexample, because 3 cannot be written as the sum of the squares of two integers. To show this is the case, note that the only perfect squares not exceeding 3 are $0^2 = 0$ and $1^2 = 1$. Furthermore, there is no way to get 3 as the sum of two terms each of which is 0 or 1. Consequently, we have shown that "Every positive integer is the sum of the squares of two integers" is false. ∎

1.7.8 Mistakes in Proofs

There are many common errors made in constructing mathematical proofs. We will briefly describe some of these here.Among the most common errors are mistakes in arithmetic and basic algebra. Even professional mathematicians make such errors, especially when working with complicated formulae.Whenever you use such computations you should check them as carefully as possible.

Each step of a mathematical proof needs to be correct and the conclusion needs to follow logically from the steps that precede it. Many mistakes result from the introduction of steps that do not logically follow from those that precede it. This is illustrated in Examples 15–17.

 EXAMPLE. 15

What is wrong with this famous supposed "proof" that $1 = 2$?

Proof: We use these steps, where a and b are two equal positive integers.

Step	**Reason**
1. $a = b$	Given
2. $a^2 = ab$	Multiply both sides of (1) by a
3. $a^2 - b^2 = ab - b^2$	Subtract b^2 from both sides of (2)
4. $(a - b)(a + b) = b(a - b)$	Factor both sides of (3)
5. $a + b = b$	Divide both sides of (4) by $a - b$
6. $2b = b$	Replace a by b in (5) because $a = b$ and simplify
7. $2 = 1$	Divide both sides of (6) by b

Solution: Every step is valid except for one, step 5 where we divided both sides by $a - b$. The error is that $a - b$ equals zero; division of both sides of an equation by the same quantity is valid as long as this quantity is not zero. ∎

 EXAMPLE. 16

What is wrong with this "proof?"

"Theorem:" If n^2 is positive, then n is positive.

Proof: Suppose that n^2 is positive. Because the conditional statement "If n is positive, then n^2 is positive" is true, we can conclude that n is positive.

Solution: Let $P(n)$ be "n is positive" and $Q(n)$ be "n^2 is positive." Then our hypothesis is $Q(n)$. The statement "If n is positive, then n^2 is positive" is the statement $\forall n(P(n) \rightarrow Q(n))$. From the hypothesis $Q(n)$ and the statement $\forall n(P(n) \rightarrow Q(n))$ we cannot conclude $P(n)$, because we are not using a valid rule of inference. Instead, this is an example of the fallacy of affirming the conclusion. A counterexample is supplied by $n = -1$ for which $n^2 = 1$ is positive, but n is negative. ∎

EXAMPLE. 17

What is wrong with this "proof?"

"Theorem:" If n is not positive, then n^2 is not positive. (This is the contrapositive of the "theorem" in Example 16.)

Proof: Suppose that n is not positive. Because the conditional statement "If n is positive, then n^2 is positive" is true, we can conclude that n^2 is not positive.

Solution: Let $P(n)$ and $Q(n)$ be as in the solution of Example 16. Then our hypothesis is $\neg P(n)$ and the statement "If n is positive, then n^2 is positive" is the statement $\forall n(P(n) \rightarrow Q(n))$. From the hypothesis $\neg P(n)$ and the statement $\forall n(P(n) \rightarrow Q(n))$ we cannot conclude $\neg Q(n)$, because we are not using a valid rule of inference. Instead, this is an example of the fallacy of denying the hypothesis. A counterexample is supplied by $n = -1$, as in Example 16. ∎

Finally, we briefly discuss a particularly nasty type of error. Many incorrect arguments are based on a fallacy called **begging the question**. This fallacy occurs when one or more steps of a proof are based on the truth of the statement being proved. In other words, this fallacy arises when a statement is proved using itself, or a statement equivalent to it. That is why this fallacy is also called **circular reasoning**.

EXAMPLE. 18

Is the following argument correct? It supposedly shows that n is an even integer whenever n^2 is an even integer.

Suppose that n^2 is even. Then $n^2 = 2k$ for some integer k. Let $n = 2l$ for

some integer l. This shows that n is even.

Solution: This argument is incorrect. The statement "let $n = 2l$ for some integer l" occurs in the proof. No argument has been given to show that n can be written as $2l$ for some integer l. This is circular reasoning because this statement is equivalent to the statement being proved, namely, "n is even". Of course, the result itself is correct; only the method of proof is wrong. ∎

Making mistakes in proofs is part of the learning process. When you make a mistake that someone else finds, you should carefully analyze where you went wrong and make sure that you do not make the same mistake again. Even professional mathematicians make mistakes in proofs. More than a few incorrect proofs of important results have fooled people for many years before subtle errors in them were found.

1.7.9 Just a Beginning

We have now developed a basic arsenal of proof methods. In the next section we will introduce other important proof methods.

In this section we introduced several methods for proving theorems of the form $\forall x(P(x) \rightarrow Q(x))$, including direct proofs and proofs by contraposition. There are many theorems of this type whose proofs are easy to construct by directly working through the hypotheses and definitions of the terms of the theorem. However, it is often difficult to prove a theorem without resorting to a clever use of a proof by contraposition or a proof by contradiction, or some other proof technique. We will describe various approaches that can be used to find proofs when straightforward approaches do not work. Constructing proofs is an art that can be learned only through experience, including writing proofs, having your proofs critiqued, and reading and analyzing other proofs.

1.8 Proof Methods and Strategy

1.8.1 Introduction

In Section 1.7 we introduced many methods of proof and illustrated how each method can be used. In this section we continue this effort.We will introduce several other commonly used proof methods, including

the method of proving a theorem by considering different cases separately. We will also discuss proofs where we prove the existence of objects with desired properties.

In Section 1.7 we briefly discussed the strategy behind constructing proofs. This strategy includes selecting a proof method and then successfully constructing an argument step by step, based on this method. In this section, after we have developed a versatile arsenal of proof methods, we will study some aspects of the art and science of proofs. We will provide advice on how to find a proof of a theorem. We will describe some tricks of the trade, including how proofs can be found by working backward and by adapting existing proofs.

When mathematicians work, they formulate conjectures and attempt to prove or disprove them. We will briefly describe this process here by proving results about tiling checkerboards with dominoes and other types of pieces. Looking at tilings of this kind, we will be able to quickly formulate conjectures and prove theorems without first developing a theory.

We will conclude the section by discussing the role of open questions. In particular, we will discuss some interesting problems either that have been solved after remaining open for hundreds of years or that still remain open

1.8.2 Exhaustive Proof and Proof by Cases

Sometimes we cannot prove a theorem using a single argument that holds for all possible cases. We now introduce a method that can be used to prove a theorem, by considering different cases separately. This method is based on a rule of inference that we will now introduce. To prove a conditional statement of the form

$$(p_1 \vee p_2 \vee \ldots \vee p_n) \rightarrow q$$

the tautology

$$[(p_1 \vee p_2 \vee \ldots \vee p_n) \rightarrow q] \leftrightarrow [(p_1 \rightarrow q) \wedge (p_2 \rightarrow q) \wedge \ldots \wedge (p_n \rightarrow q)]$$

can be used as a rule of inference. This shows that the original conditional statement with a hypothesis made up of a disjunction of the

propositions $p_1, p_2, \ldots, p_n$ can be proved by proving each of the n conditional statements $p_i \rightarrow q$, $i = 1, 2, \ldots, n$, individually. Such an argument is called a proof by cases. Sometimes to prove that a conditional statement $p \rightarrow q$ is true, it is convenient to use a disjunction $p_1 \vee p_2 \vee \ldots \vee p_n$ instead of p as the hypothesis of the conditional statement, where p and $p_1 \vee p_2 \vee \ldots \vee p_n$ are equivalent.

EXHAUSTIVEPROOF

Some theorems can be proved by examining a relatively small number of examples. Such proofs are called **exhaustive proofs**, or **proofs by exhaustion** because these proofs proceed by exhausting all possibilities. An exhaustive proof is a special type of proof by cases where each case involves checking a single example.We now provide some illustrations of exhaustive proofs.

EXAMPLE. 1

Prove that $(n + 1)3 \geq 3^n$ if n is a positive integer with $n \leq 4$.

Solution: We use a proof by exhaustion. We only need verify the inequality $(n+1)^3 \geq 3n$ when $n = 1, 2, 3$, and *4*. For $n = 1$, we have $(n+1)^3 = 2^3 = 8$ and $3^n = 3^1 = 3$; for $n = 2$, we have $(n + 1)^3 = 3^3 = 27$ and $3^n = 3^2 = 9$; for $n = 3$, we have $(n + 1)^3 = 4^3 = 64$ and $3^n = 3^3 = 27$; and for $n = 4$, we have $(n + 1)^3 = 5^3 = 125$ and $3^n = 3^4 = 81$. In each of these four cases, we see that $(n+1)^3 \geq 3^n$. We have used the method of exhaustion to prove that $(n + 1)^3 \geq 3^n$ if n is a positive integer with $n \leq 4$. ∎

EXAMPLE. 2

Prove that the only consecutive positive integers not exceeding 100 that are perfect powers are 8 and 9. (An integer is a **perfect power** if it equals n^a, where a is an integer greater than 1.)

Solution: We use a proof by exhaustion. In particular, we can prove this fact by examining positive integers n not exceeding 100, first checking whether n is a perfect power, and if it is, checking whether $n + 1$ is also a perfect power.A quicker way to do this is simply to look at all perfect powers not exceeding 100 and checking whether the next largest integer is also a

perfect power. The squares of positive integers not exceeding 100 are 1, 4, 9, 16, 25, 36, 49, 64, 81, and 100. The cubes of positive integers not exceeding 100 are 1, 8, 27, and 64. The fourth powers of positive integers not exceeding 100 are 1, 16, and 81. The fifth powers of positive integers not exceeding 100 are 1 and 32. The sixth powers of positive integers not exceeding 100 are 1 and 64. There are no powers of positive integers higher than the sixth power not exceeding 100, other than 1. Looking at this list of perfect powers not exceeding 100, we see that $n = 8$ is the only perfect power n for which $n+1$ is also a perfect power. That is, $2^3 = 8$ and $3^2 = 9$ are the only two consecutive perfect powers not exceeding 100. ∎

People can carry out exhaustive proofs when it is necessary to check only a relatively small number of instances of a statement. Computers do not complain when they are asked to check a much larger number of instances of a statement, but they still have limitations. Note that not even a computer can check all instances when it is impossible to list all instances to check.

PROOF BY CASES

A proof by cases must cover all possible cases that arise in a theorem. We illustrate proof by cases with a couple of examples. In each example, you should check that all possible cases are covered.

EXAMPLE. 3

Prove that if n is an integer, then $n^2 \geq n$.

Solution: We can prove that $n^2 \geq n$ for every integer by considering three cases, when $n = 0$, when $n \geq 1$, and when $n \leq -1$. We split the proof into three cases because it is straightforward to prove the result by considering zero, positive integers, and negative integers separately.

Case (i): When $n = 0$, because $0^2 = 0$, we see that $0^2 \geq 0$. It follows that $n^2 \geq n$ is true in this case.

Case (ii): When $n \geq 1$, when we multiply both sides of the inequality $n \geq 1$ by the positive integer n, we obtain $n \cdot n \geq n \cdot 1$. This implies that $n^2 \geq n$ for $n \geq 1$.

Case (iii): In this case $n \leq -1$. However, $n^2 \geq 0$. It follows that $n^2 \geq n$.

Because the inequality $n^2 \geq n$ holds in all three cases, we can conclude that if n is an integer, then $n^2 \geq n$. ∎

EXAMPLE. 4

Use a proof by cases to show that $|xy| = |x||y|$, where x and y are real numbers. (Recall that $|a|$, the absolute value of a, equals a when $a \geq 0$ and equals $-a$ when $a \leq 0$.)

Solution: In our proof of this theorem, we remove absolute values using the fact that $|a| = a$ when $a \geq 0$ and $|a| = -a$ when $a < 0$. Because both $|x|$ and $|y|$ occur in our formula, we will need four cases: *(i)* x and y both nonnegative, *(ii)* x nonnegative and y is negative, *(iii)* x negative and y nonnegative, and *(iv)* x negative and y negative. We denote by p_1, p_2, p_3, and p_4, the proposition stating the assumption for each of these four cases, respectively.

(Note that we can remove the absolute value signs by making the appropriate choice of signs within each case.)

Case (i): We see that $p_1 \rightarrow q$ because $xy \geq 0$ when $x \geq 0$ and $y \geq 0$, so that $|xy| = xy = |x||y|$.

Case (ii): To see that $p_2 \rightarrow q$, note that if $x \geq 0$ and $y < 0$, then $xy \leq 0$, so that $|xy| = -xy = x(-y) = |x||y|$. (Here, because $y < 0$, we have $|y| = -y$.)

Case (iii): To see that $p_3 \rightarrow q$, we follow the same reasoning as the previous case with the roles of x and y reversed.

Case (iv): To see that $p_4 \rightarrow q$, note that when $x < 0$ and $y < 0$, it follows that $xy > 0$. Hence, $|xy| = xy = (-x)(-y) = |x||y|$.

Because $|xy| = |x||y|$ holds in each of the four cases and these cases exhaust all possibilities, we can conclude that $|xy| = |x||y|$, whenever x and y are real numbers. ∎

LEVERAGING PROOF BY CASES

The examples we have presented illustrating proof by cases provide some insight into when to use this method of proof. In particular, when it is not possible to consider all cases of a proof at the same time, a proof by cases should be considered. When should you use such a proof? Generally, look for a proof by cases when there is no obvious way to begin a proof, but when extra information in each case helps move the proof forward. Example 5 illustrates how the method of proof by cases can be used effectively.

EXAMPLE. 5

Formulate a conjecture about the final decimal digit of the square of an integer and prove your result.

Solution: The smallest perfect squares are 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, and so on. We notice that the digits that occur as the final digit of a square are 0, 1, 4, 5, 6, and 9, with 2, 3, 7, and 8 never appearing as the final digit of a square.We conjecture this theorem: The final decimal digit of a perfect square is 0, 1, 4, 5, 6 or 9. How can we prove this theorem?

We first note that we can express an integer n as $10a + b$, where a and b are positive integers and b is 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. Here a is the integer obtained by subtracting the final decimal digit of n from n and dividing by 10. Next, note that $(10a + b)^2 = 100a^2 + 20ab + b^2 = 10(10a^2 + 2b) + b^2$, so that the final decimal digit of n^2 is the same as the final decimal digit of b^2. Furthermore, note that the final decimal digit of b^2 is the same as the final decimal digit of $(10 - b)^2 = 100 - 20b + b^2$. Consequently, we can reduce our proof to the consideration of six cases.

Case (i): The final digit of n is 1 or 9. Then the final decimal digit of n^2 is the final decimal digit of $1^2 = 1$ or $9^2 = 81$, namely 1.

Case (ii): The final digit of n is 2 or 8. Then the final decimal digit of n^2 is the final decimal digit of $2^2 = 4$ or $8^2 = 64$, namely 4.

Case (iii): The final digit of n is 3 or 7. Then the final decimal digit of n^2 is the final decimal digit of $3^2 = 9$ or $7^2 = 49$, namely 9.

Case (iv): The final digit of n is 4 or 6. Then the final decimal digit of n^2 is the final decimal digit of $4^2 = 16$ or $6^2 = 36$, namely 6.

Case (v): The final decimal digit of n is 5. Then the final decimal digit of n^2 is the final decimal digit of $5^2 = 25$, namely 5.

Case (vi): The final decimal digit of n is 0. Then the final decimal digit of n^2 is the final decimal digit of $0^2 = 0$, namely 0.

Because we have considered all six cases, we can conclude that the final decimal digit of n^2, where n is an integer is either 0, 1, 2, 4, 5, 6, or 9. ∎

Sometimes we can eliminate all but a few examples in a proof by cases, as Example 6 illustrates.

EXAMPLE. 6

Show that there are no solutions in integers x and y of $x^2 + 3y^2 = 8$.

Solution: We can quickly reduce a proof to checking just a few simple cases because $x^2 > 8$ when $|x| \geq 3$ and $3y^2 > 8$ when $|y| \geq 2$. This leaves the cases when x equals $-2, -1, 0, 1$, or 2 and y equals $-1, 0$, or 1. We can finish using an exhaustive proof. To dispense with the remaining cases, we note that possible values for x^2 are 0, 1, and 4, and possible values for $3y^2$ are 0 and 3, and the largest sum of possible values for x^2 and $3y^2$ is 7. Consequently, it is impossible for $x^2+3y^2 = 8$ to hold when x and y are integers. ∎

WITHOUT LOSS OF GENERALITY

In the proof in Example 4, we dismissed case *(iii)*, where $x < 0$ and $y \geq 0$, because it is the same as case *(ii)*, where $x \geq 0$ and $y < 0$, with the roles of x and y reversed. To shorten the proof, we could have proved cases *(ii)* and *(iii)* together by assuming, without loss of generality, that $x \geq 0$ and $y < 0$. Implicit in this statement is that we can complete the case with $x < 0$ and $y \geq 0$ using the same argument as we used for the case with $x \geq 0$ and $y < 0$, but with the obvious changes.

In general, when the phrase "without loss of generality" is used in a proof (often abbreviated as WLOG), we assert that by proving one case of a theorem, no additional argument is required to prove other specified cases. That is, other cases follow by making straightforward changes to the argument, or by filling in some straightforward initial step. Proofs by cases can often be made much more efficient when the notion of without loss of generality is employed. Of course, incorrect use of this principle can lead to unfortunate errors. Sometimes assumptions are made that lead to a loss in generality. Such assumptions can be made that do not take into account that one case may be substantially different from others. This can lead to an incomplete, and possibly unsalvageable, proof. In fact, many incorrect proofs of famous theorems turned out to rely on arguments that used the idea of "without loss of generality" to establish cases that could not be quickly proved from simpler cases.

We now illustrate a proof where without loss of generality is used effectively together with other proof techniques.

EXAMPLE. 7

Show that if x and y are integers and both xy and $x + y$ are even, then both x and y are even.

Solution: We will use proof by contraposition, the notion of without loss of generality, and proof by cases. First, suppose that x and y are not both even. That is, assume that x is odd or that y is odd (or both). Without loss of generality, we assume that x is odd, so that $x = 2m + 1$ for some integer k.

To complete the proof, we need to show that xy is odd or $x+y$ is odd. Consider two cases: *(i)* y even, and *(ii)* y odd. In *(i)*, $y = 2n$ for some integer n, so that $x+y = (2m+1)+2n = 2(m+n)+1$ is odd. In *(ii)*, $y = 2n+1$ for some integer n, so that $xy = (2m+1)(2n+1) = 4mn+2m+2n+1 = 2(2mn+m+n)+1$ is odd. This completes the proof by contraposition. (Note that our use of without loss of generality within the proof is justified because the proof when y is odd can be obtained by simply interchanging the roles of x and y in the proof we have given.) ∎

COMMON ERRORS WITH EXHAUSTIVE PROOF AND PROOF BY CASES

A common error of reasoning is to drawincorrect conclusions from examples. No matter howmany separate examples are considered, a theorem is not proved by considering examples unless every possible case is covered. The problem of proving a theorem is analogous to showing that a computer program always produces the output desired. No matter howmany input values are tested, unless all input values are tested, we cannot conclude that the program always produces the correct output.

EXAMPLE. 8

Is it true that every positive integer is the sum of 18 fourth powers of integers?

Solution: To determine whether a positive integer n can be written as the sum of 18 fourth powers of integers, we might begin by examining whether n is the sum of 18 fourth powers of integers for the smallest positive integers. Because the fourth powers of integers are $0, 1, 16, 81, \ldots$, if we can select 18 terms from these numbers that add up to n, then n is the sum of 18 fourth

powers. We can show that all positive integers up to 78 can be written as the sum of 18 fourth powers. (The details are left to the reader.) However, if we decided this was enough checking, we would come to the wrong conclusion. It is not true that every positive integer is the sum of 18 fourth powers because 79 is not the sum of 18 fourth powers (as the reader can verify). ∎

Another common error involves making unwarranted assumptions that lead to incorrect proofs by cases where not all cases are considered. This is illustrated in Example 9.

EXAMPLE. 9

What is wrong with this "proof?"

"Theorem:" If x is a real number, then x^2 is a positive real number.

Proof: Let p_1 be "x is positive", let p_2 be "x is negative", and let q be "x^2 is positive". To show that $p_1 \rightarrow q$ is true, note that when x is positive, x^2 is positive because it is the product of two positive numbers, x and x. To show that $p_2 \rightarrow q$, note that when x is negative, x^2 is positive because it is the product of two negative numbers, x and x. This completes the proof.

Solution: The problem with this "proof" is that we missed the case of $x = 0$. When $x = 0$, $x^2 = 0$ is not positive, so the supposed theorem is false. If p is "x is a real number", then we can prove results where p is the hypothesis with three cases, p_1, p_2, and p_3, where p_1 is "x is positive", p_2 is "x is negative", and p_3 is "$x = 0$" because of the equivalence $p \leftrightarrow p_1 \vee p_2 \vee p_3$. ∎

1.8.3 Existence Proofs

Many theorems are assertions that objects of a particular type exist. A theorem of this type is a proposition of the form $\exists x P(x)$, where P is a predicate. A proof of a proposition of the form $\exists x P(x)$ is called an **existence proof**. There are several ways to prove a theorem of this type. Sometimes an existence proof of $\exists x P(x)$ can be given by finding an element a, called a **witness**, such that *P(a)* is true. This type of existence proof is called **constructive**. It is also possible to give an existence proof that is **nonconstructive**; that is, we do not find an element a such that $P(a)$ is true, but rather prove that $\exists x P(x)$ is true in

some other way. One common method of giving a nonconstructive existence proof is to use proof by contradiction and show that the negation of the existential quantification implies a contradiction. The concept of a constructive existence proof is illustrated by Example 10 and the concept of a nonconstructive existence proof is illustrated by Example 11.

EXAMPLE. 10

A Constructive Existence Proof Show that there is a positive integer that can be written as the sum of cubes of positive integers in two different ways.

Solution: After considerable computation (such as a computer search) we find that

$$1729 = 10^3 + 9^3 = 12^3 + 1^3.$$

Because we have displayed a positive integer that can be written as the sum of cubes in two different ways, we are done.

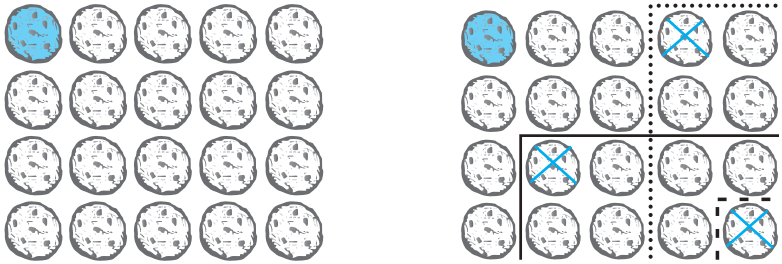
There is an interesting story pertaining to this example. The English mathematician G. H. Hardy, when visiting the ailing Indian prodigy Ramanujan in the hospital, remarked that 1729, the number of the cab he took, was rather dull. Ramanujan replied "No, it is a very interesting number; it is the smallest number expressible as the sum of cubes in two different ways". ∎

EXAMPLE. 11

A Nonconstructive Existence Proof Show that there exist irrational numbers x and y such that xy is rational.

Solution: We know that $\sqrt{2}$ is irrational. Consider the number $\sqrt{2}^{\sqrt{2}}$. If it is rational, we have two irrational numbers x and y with x^y rational, namely, $x = \sqrt{2}$ and $y = \sqrt{2}$. On the other hand if $\sqrt{2}^{\sqrt{2}}$ is irrational, then we can let $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$ so that $x^y = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{(\sqrt{2}\cdot\sqrt{2})} = \sqrt{2}^2 = 2$.

This proof is an example of a nonconstructive existence proof because we have not found irrational numbers x and y such that x^y is rational. Rather, we have shown that either the pair $x = \sqrt{2}$, $y = \sqrt{2}$ or the pair $x = \sqrt{2}^{\sqrt{2}}$,



a) b)

Figure 1.4: (a) Chomp (Top Left Cookie Poisoned). (b) Three Possible Moves.

$y = \sqrt{2}$ have the desired property, but we do not know which of these two pairs works! ∎

EXAMPLE. 12

Chomp is a game played by two players. In this game, cookies are laid out on a rectangular grid. The cookie in the top left position is poisoned, as shown in Figure 1.4(a). The two players take turns making moves; at each move, a player is required to eat a remaining cookie, together with all cookies to the right and/or below it (see Figure 1.4(b), for example). The loser is the player who has no choice but to eat the poisoned cookie.We ask whether one of the two players has a winning strategy. That is, can one of the players always make moves that are guaranteed to lead to a win?

Solution: We will give a nonconstructive existence proof of a winning strategy for the first player. That is, we will show that the first player always has a winning strategy without explicitly describing the moves this player must follow.

First, note that the game ends and cannot finish in a draw because with each move at least one cookie is eaten, so after no more than $m \times n$ moves the game ends, where the initial grid is $m \times n$. Now, suppose that the first player begins the game by eating just the cookie in the bottom right corner. There are two possibilities, this is the first move of a winning strategy for the first player, or the second player can make a move that is the first move of a winning strategy for the second player. In this second case, instead of eating just the cookie in the bottom right corner, the first player could have made

the same move that the second player made as the first move of a winning strategy (and then continued to follow that winning strategy). This would guarantee a win for the first player.

Note that we showed that a winning strategy exists, but we did not specify an actual winning strategy. Consequently, the proof is a nonconstructive existence proof. In fact, no one has been able to describe a winning strategy for that Chomp that applies for all rectangular grids by describing the moves that the first player should follow. However, winning strategies can be described for certain special cases, such as when the grid is square and when the grid only has two rows of cookies. ∎

1.8.4 Uniqueness Proofs

Some theorems assert the existence of a unique element with a particular property. In other words, these theorems assert that there is exactly one element with this property. To prove a statement of this type we need to show that an element with this property exists and that no other element has this property. The two parts of a **uniqueness proof** are:

Existence: We show that an element x with the desired property exists.

Uniqueness: We show that if $y \neq x$, then y does not have the desired property.

Equivalently, we can show that if x and y both have the desired property, then $x = y$.

> **Remark!** Showing that there is a unique element x such that $P(x)$ is the same as proving the statement $\exists x(P(x) \wedge \forall y(y \neq x \rightarrow \neg P(y)))$.

We illustrate the elements of a uniqueness proof in Example 13.

EXAMPLE. 13

Show that if a and b are real numbers and $a \neq 0$, then there is a unique real number r such that $ar + b = 0$.

Solution: First, note that the real number $r = -b/a$ is a solution of $ar + b = 0$ because $a(-b/a) + b = -b + b = 0$. Consequently, a real number r exists for which $ar + b = 0$. This is the existence part of the proof.

Second, suppose that s is a real number such that $as + b = 0$. Then $ar + b = as + b$, where $r = -b/a$. Subtracting b from both sides, we find that $ar = as$. Dividing both sides of this last equation by a, which is nonzero, we see that $r = s$. This means that if $s = r$, then $as + b = 0$. This establishes the uniqueness part of the proof. ∎

1.8.5 Proof Strategies

Finding proofs can be a challenging business. When you are confronted with a statement to prove, you should first replace terms by their definitions and then carefully analyze what the hypotheses and the conclusion mean. After doing so, you can attempt to prove the result using one of the available methods of proof. Generally, if the statement is a conditional statement, you should first try a direct proof; if this fails, you can try an indirect proof. If neither of these approaches works, you might try a proof by contradiction.

FORWARD AND BACKWARD REASONING

Whichever method you choose, you need a starting point for your proof. To begin a direct proof of a conditional statement, you start with the premises. Using these premises, together with axioms and known theorems, you can construct a proof using a sequence of steps that leads to the conclusion. This type of reasoning, called **forward reasoning**, is the most common type of reasoning used to prove relatively simple results. Similarly, with indirect reasoning you can start with the negation of the conclusion and, using a sequence of steps, obtain the negation of the premises.

Unfortunately, forward reasoning is often difficult to use to prove more complicated results, because the reasoning needed to reach the desired conclusion may be far from obvious. In such cases it may be helpful to use **backward reasoning**. To reason backward to prove a statement q, we find a statement p that we can prove with the property that $p \rightarrow q$. (Note that it is not helpful to find a statement r that

you can prove such that $q \to r$, because it is the fallacy of begging the question to conclude from $q \to r$ and r that q is true.) Backward reasoning is illustrated in Examples 14 and 15.

EXAMPLE. 14

Given two positive real numbers x and y, their **arithmetic mean** is $(x+y)/2$ and their **geometric mean** is $\sqrt{xy}$. When we compare the arithmetic and geometric means of pairs of distinct positive real numbers, we find that the arithmetic mean is always greater than the geometric mean. [For example, when $x = 4$ and $y = 6$, we have $5 = (4+6)/2 > \sqrt{4 \cdot 6} = \sqrt{24}$.] Can we prove that this inequality is always true?

Solution: To prove that $(x + y)/2 > \sqrt{xy}$ when x and y are distinct positive real numbers, we can work backward. We construct a sequence of equivalent inequalities. The equivalent inequalities are

$$\begin{aligned}
&(x + y)/2 > \sqrt{xy},\\
&(x + y)^2/4 > xy,\\
&(x + y)^2 > 4xy,\\
&x^2 + 2xy + y^2 > 4xy,\\
&x^2 - 2xy + y^2 > 0,\\
&(x - y)^2 > 0
\end{aligned}$$

Because $(x - y)^2 > 0$ when $x \neq y$, it follows that the final inequality is true. Because all these inequalities are equivalent, it follows that $(x + y)/2 > \sqrt{xy}$ when $x \neq y$. Once we have carried out this backward reasoning, we can easily reverse the steps to construct a proof using forward reasoning. We now give this proof.

Suppose that x and y are distinct positive real numbers. Then $(x-y)^2 > 0$ because the square of a nonzero real number is positive. Because $(x - y)^2 = x^2 - 2xy + y^2$, this implies that $x^2 - 2xy + y^2 > 0$. Adding $4xy$ to both sides, we obtain $x^2 + 2xy + y^2 > 4xy$. Because $x^2 + 2xy + y^2 = (x + y)^2$, this means that $(x + y)^2 \geq 4xy$. Dividing both sides of this equation by 4, we see that $(x + y)^2/4 > xy$. Finally, taking square roots of both sides (which preserves the inequality because both sides are positive) yields $(x + y)/2 > \sqrt{xy}$. We conclude that if x and y are distinct positive real numbers, then their arithmetic mean $(x + y)/2$ is greater than their geometric mean $\sqrt{xy}$. ∎

EXAMPLE. 15

Suppose that two people play a game taking turns removing one, two, or three stones at a time from a pile that begins with 15 stones. The person who removes the last stone wins the game. Show that the first player can win the game no matter what the second player does.

Solution: To prove that the first player can always win the game, we work backward. At the last step, the first player can win if this player is left with a pile containing one, two, or three stones. The second player will be forced to leave one, two, or three stones if this player has to remove stones from a pile containing four stones. Consequently, one way for the first person to win is to leave four stones for the second player on the next-to-last move. The first person can leave four stones when there are five, six, or seven stones left at the beginning of this player's move, which happens when the second player has to remove stones from a pile with eight stones. Consequently, to force the second player to leave five, six, or seven stones, the first player should leave eight stones for the second player at the second-to-last move for the first player. This means that there are nine, ten, or eleven stones when the first player makes this move. Similarly, the first player should leave twelve stones when this player makes the first move. We can reverse this argument to show that the first player can always make moves so that this player wins the game no matter what the second player does. These moves successively leave twelve, eight, and four stones for the second player. ∎

ADAPTING EXISTING PROOFS

An excellent way to look for possible approaches that can be used to prove a statement is to take advantage of existing proofs of similar results. Often an existing proof can be adapted to prove other facts. Even when this is not the case, some of the ideas used in existing proofs may be helpful. Because existing proofs provide clues for new proofs, you should read and understand the proofs you encounter in your studies. This process is illustrated in Example 16.

EXAMPLE. 16

We proved that $\sqrt{2}$ is irrational. We now conjecture that $\sqrt{3}$ is irrational.

Can we adapt the proof in Example 10 in Section 1.7 to show that $\sqrt{3}$ is irrational?

Solution: To adapt the proof, we begin by mimicking the steps in that proof, but with $\sqrt{2}$ replaced with $\sqrt{3}$. First, we suppose that $\sqrt{3} = d/c$ where the fraction c/d is in lowest terms. Squaring both sides tells us that $3 = c^2/d^2$, so that $3d^2 = c^2$. Can we use this equation to show that 3 must be a factor of both c and d, similar to how we used the equation $2b^2 = a^2$ in Example 10 in Section 1.7 to show that 2 must be a factor of both a and b? In turns out that we can, but we need some ammunition from number theory. Because 3 is a factor of c^2, it must also be a factor of c. Furthermore, because 3 is a factor of c, 9 is a factor of c^2, which means that 9 is a factor of $3d^2$. This implies that 3 is a factor of d^2, which means that 3 is a factor of that d. This makes 3 a factor of both c and d, which contradicts the assumption that c/d is in lowest terms. After we have filled in the justification for these steps, we will have shown that $\sqrt{3}$ is irrational by adapting the proof that $\sqrt{2}$ is irrational. Note that this proof can be extended to show that $\sqrt{n}$ is irrational whenever n is a positive integer that is not a perfect square. ∎

A good tip is to look for existing proofs that you might adapt when you are confronted with proving a new theorem, particularly when the new theorem seems similar to one you have already proved.

1.8.6 Looking for Counterexamples

When confronted with a conjecture, you might first try to prove this conjecture, and if your attempts are unsuccessful, you might try to find a counterexample, first by looking at the simplest, smallest examples. If you cannot find a counterexample, you might again try to prove the statement. In any case, looking for counterexamples is an extremely important pursuit, which often provides insights into problems. We will illustrate the role of counterexamples in Example 17.

EXAMPLE. 17

In Example 14 in Section 1.7 we showed that the statement "Every positive integer is the sum oftwo squares of integers" is false by finding a counterexample. That is, there are positive integers that cannot be written as the sum of

the squares of two integers. Although we cannot write every positive integer as the sum of the squares of two integers, maybe we can write every positive integer as the sum of the squares of three integers. That is, is the statement "Every positive integer is the sum of the squares of three integers" true or false?

Solution: Because we know that not every positive integer can be written as the sum of two squares of integers, we might initially be skeptical that every positive integer can be written as the sum of three squares of integers. So, we first look for a counterexample. That is, we can show that the statement "Every positive integer is the sum of three squares of integers" is false if we can find a particular integer that is not the sum of the squares of three integers. To look for a counterexample, we try to write successive positive integers as a sum of three squares. We find that $1 = 0^2 + 0^2 + 1^2$, $2 = 0^2 + 1^2 + 1^2$, $3 = 1^2 + 1^2 + 1^2$, $4 = 0^2 + 0^2 + 2^2$, $5 = 0^2 + 1^2 + 2^2$, $6 = 1^2 + 1^2 + 2^2$, but we cannot find a way to write 7 as the sum of three squares. To show that there are not three squares that add up to 7, we note that the only possible squares we can use are those not exceeding 7, namely, 0, 1, and 4. Because no three terms where each term is 0, 1, or 4 add up to 7, it follows that 7 is a counterexample. We conclude that the statement "Every positive integer is the sum of the squares of three integers" is false.

We have shown that not every positive integer is the sum of the squares of three integers. The next question to ask is whether every positive integer is the sum of the squares of four positive integers. Some experimentation provides evidence that the answer is yes. For example, $7 = 1^2 + 1^2 + 1^2 + 2^2$, $25 = 4^2 + 2^2 + 2^2 + 1^2$, and $87 = 9^2 + 2^2 + 1^2 + 1^2$. It turns out the conjecture "Every positive integer is the sum of the squares of four integers" is true. ∎

1.8.7 Proof Strategy in Action

Mathematics is generally taught as if mathematical facts were carved in stone. Mathematics texts (including the bulk of this book) formally present theorems and their proofs. Such presentations do not convey the discovery process in mathematics. This process begins with exploring concepts and examples, asking questions, formulating conjectures, and attempting to settle these conjectures either by proof or by counterexample. These are the day-to-day activities of mathematicians. Believe it or not, the material taught in textbooks was originally developed in this way.

People formulate conjectures on the basis of many types of possible evidence. The examination of special cases can lead to a conjecture, as can the identification of possible patterns. Altering the hypotheses and conclusions of known theorems also can lead to plausible conjectures. At other times, conjectures are made based on intuition or a belief that a result holds. No matter how a conjecture was made, once it has been formulated, the goal is to prove or disprove it. When mathematicians believe that a conjecture may be true, they try to find a proof. If they cannot find a proof, they may look for a counterexample. When they cannot find a counterexample, they may switch gears and once again try to prove the conjecture. Although many conjectures are quickly settled, a few conjectures resist attack for hundreds of years and lead to the development of new parts of mathematics. We will mention a few famous conjectures later in this section.

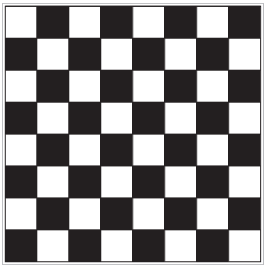


Figure 1.5: The Standard Checkerboard.

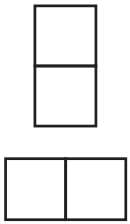


Figure 1.6: Two Dominoes

1.8.8 Tilings

We can illustrate aspects of proof strategy through a brief study of tilings of checkerboards. Looking at tilings of checkerboards is a fruitful way to quickly discover many different results and construct their proofs using a variety of proof methods. There are almost an endless number of conjectures that can be made and studied in this area too. To begin, we need to define some terms. A **checkerboard** is a rectangle divided into squares of the same size by horizontal and vertical lines. The game

of checkers is played on a board with 8 rows and 8 columns; this board is called the **standard checkerboard** and is shown in Figure 1.5. In this section we use the term **board** to refer to a checkerboard of any rectangular size as well as parts of checkerboards obtained by removing one or more squares. A **domino** is a rectangular piece that is one square by two squares, as shown in Figure 1.6. We say that a board is **tiled** by dominoes when all its squares are covered with no overlapping dominoes and no dominoes overhanging the board. We now develop some results about tiling boards using dominoes.

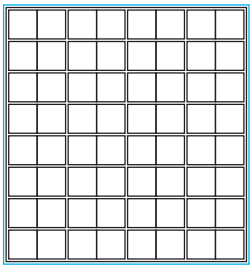


Figure 1.7: Tiling the Standard Checkerboard.

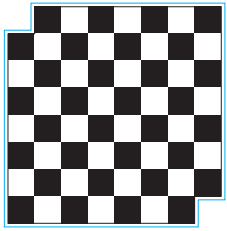


Figure 1.8: The Standard Checkerboard with the Upper Left and Lower Right Squares Removed.

 EXAMPLE. 18

Can we tile the standard checkerboard using dominoes?

Solution: We can find manyways to tile the standard checkerboard using dominoes. For example, we can tile it by placing 32 dominoes horizontally, as shown in Figure 1.7. The existence of one such tiling completes a constructive existence proof. Of course, there are a large number of other ways to do this tiling.We can place 32 dominoes vertically on the board or we can place some tiles vertically and some horizontally. But for a constructive existence proof we needed to find just one such tiling. ∎

EXAMPLE. 19

Can we tile a board obtained by removing one of the four corner squares of a standard checkerboard?

Solution: To answer this question, note that a standard checkerboard has 64 squares, so removing a square produces a board with 63 squares. Now suppose that we could tile a board obtained from the standard checkerboard by removing a corner square. The board has an even number of squares because each domino covers two squares and no two dominoes overlap and no dominoes overhang the board. Consequently, we can prove by contradiction that a standard checkerboard with one square removed cannot be tiled using dominoes because such a board has an odd number of squares. ∎

We now consider a trickier situation.

EXAMPLE. 20

Can we tile the board obtained by deleting the upper left and lower right corner squares of a standard checkerboard, shown in Figure 1.8?

Solution: A board obtained by deleting two squares of a standard checkerboard contains $64 - 2 = 62$ squares. Because 62 is even, we cannot quickly rule out the existence of a tiling of the standard checkerboard with its upper left and lower right squares removed, unlike Example 19, where we ruled out the existence of a tiling of the standard checkerboard with one corner square removed. Trying to construct a tiling of this board by successively placing dominoes might be a first approach, as the reader should attempt. However, no matter how much we try, we cannot find such a tiling. Because our efforts do not produce a tiling, we are led to conjecture that no tiling exists.

We might try to prove that no tiling exists by showing that we reach a dead end however we successively place dominoes on the board. To construct such a proof, we would have to consider all possible cases that arise as we run through all possible choices of successively placing dominoes. For example, we have two choices for covering the square in the second column of the first row, next to the removed top left corner.We could cover it with a horizontally placed tile or a vertically placed tile. Each of these two choices leads to further choices, and so on. It does not take long to see that this is not a fruitful plan of attack for a person, although a computer could be used to complete such

a proof by exhaustion.

We need another approach. Perhaps there is an easier way to prove there is no tiling of a standard checkerboard with two opposite corners removed. As with many proofs, a key observation can help. We color the squares of this checkerboard using alternating white and black squares, as in Figure 1.5. Observe that a domino in a tiling of such a board covers one white square and one black square. Next, note that this board has unequal numbers of white square and black squares.We can use these observations to prove by contradiction that a standard checkerboard with opposite corners removed cannot be tiled using dominoes.We now present such a proof.

Proof: Suppose we can use dominoes to tile a standard checkerboard with opposite corners removed. Note that the standard checkerboard with opposite corners removed contains $64 - 2 = 62$ squares. The tilingwould use $62/2 = 31$ dominoes. Note that each domino in this tiling covers one white and one black square. Consequently, the tiling covers 31 white squares and 31 black squares. However, when we remove two opposite corner squares, either 32 of the remaining squares are white and 30 are black or else 30 are white and 32 are black. This contradicts the assumption that we can use dominoes to cover a standard checkerboard with opposite corners removed, completing the proof. ∎

We can use other types of pieces besides dominoes in tilings. Instead of dominoes we can study tilings that use identically shaped pieces constructed from congruent squares that are connected along their edges.

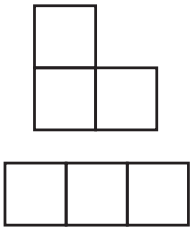


Figure 1.9: A Right Triomino and a Straight Triomino.

Such pieces are called polyominoes, a term coined in 1953 by the mathematician Solomon Golomb, the author of an entertaining book about them. We will consider two polyominoes with the same number of squares the same if we can rotate and/or flip one of the polyominoes to get the other one. For example, there are two types of triominoes (see Figure 1.9), which are polyominoes made up of three squares connected by their sides. One type of triomino, the straight triomino, has three horizontally connected squares; the other type, right triominoes, resembles the letter L in shape, flipped and/or rotated, if necessary. We

will study the tilings of a checkerboard by straight triominoes here.

EXAMPLE. 21

Can you use straight triominoes to tile a standard checkerboard?

Solution: The standard checkerboard contains 64 squares and each triomino covers three squares. Consequently, if triominoes tile a board, the number of squares of the board must be a multiple of 3. Because 64 is not a multiple of 3, triominoes cannot be used to cover an 8×8 checkerboard. ∎

In Example 22, we consider the problem of using straight triominoes to tile a standard checkerboard with one corner missing.

EXAMPLE. 22

Can we use straight triominoes to tile a standard checkerboard with one of its four corners removed? An 8×8 checkerboard with one corner removed contains $64 - 1 = 63$ squares. Any tiling by straight triominoes of one of these four boards uses $63/3 = 21$ triominoes. However, when we experiment, we cannot find a tiling of one of these boards using straight triominoes. A proof by exhaustion does not appear promising. Can we adapt our proof from Example 20 to prove that no such tiling exists?

Solution: We will color the squares of the checkerboard in an attempt to adapt the proof by contradiction we gave in Example 20 of the impossibility of using dominoes to tile a standard checkerboard with opposite corners removed. Because we are using straight triominoes rather than dominoes, we color the squares using three colors rather than two colors, as shown in Figure 1.10.

Note that there are 21 blue squares, 21 black squares, and 22 white squares in this coloring. Next, we make the crucial observation that when a straight triomino covers three squares of the checkerboard, it covers one blue square, one black square, and one white square. Next, note that each of the three colors appears in a corner square. Thus without loss of generality, we may assume that we have rotated the coloring so that the missing square is colored blue. Therefore, we assume that the remaining board contains 20 blue squares, 21 black squares, and 22 white squares. If we could tile this board using straight triominoes, then we would use $63/3 = 21$ straight triominoes. These triominoes would cover 21 blue squares, 21 black squares, and 21 white

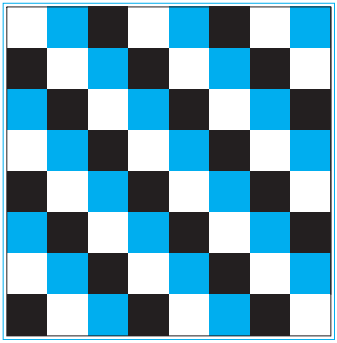


Figure 1.10: Coloring the Squares of the Standard Checkerboard with Three Colors.

squares. This contradicts the fact that this board contains 20 blue squares, 21 black squares, and 22 white squares. Therefore we cannot tile this board using straight triominoes. ∎

1.8.9 The Role of Open Problems

Many advances in mathematics have been made by people trying to solve famous unsolved problems. In the past 20 years, many unsolved problems have finally been resolved, such as the proof of a conjecture in number theory made more than 300 years ago. This conjecture asserts the truth of the statement known as **Fermat's last theorem.**

THEOREM 1.8.1: FERMAT'S LAST THEOREM

The equation

$$x^n + y^n = z^n$$

has no solutions in integers x,y, and z with $xyz = 0$ whenever n is an integer with $n > 2$.

Remark! The equation $x^2 + y^2 = z^2$ has infinitely many solutions in integers x, y, and z; these solutions are called Pythagorean triples and correspond to the lengths of the sides of right triangles with integer lengths.

This problem has a fascinating history. In the seventeenth century, Fermat jotted in the margin of his copy of the works of Diophantus that he had a "wondrous proof" that there are no integer solutions of $x^n+y^n = z^n$ when n is an integer greater than 2 with $xyz \neq 0$. However, he never published a proof (Fermat published almost nothing), and no proof could be found in the papers he left when he died. Mathematicians looked for a proof for three centuries without success, although many people were convinced that a relatively simple proof could be found. (Proofs of special cases were found, such as the proof of the case when $n = 3$ by Euler and the proof of the $n = 4$ case by Fermat himself.) Over the years, several established mathematicians thought that they had proved this theorem. In the nineteenth century, one of these failed attempts led to the development of the part of number theory called algebraic number theory. A correct proof, requiring hundreds of pages of advanced mathematics, was not found until the 1990s, when AndrewWiles used recently developed ideas from a sophisticated area of number theory called the theory of elliptic curves to prove Fermat's last theorem. Wiles's quest to find a proof of Fermat's last theorem using this powerful theory, described in a program in the Nova series on public television, took close to ten years! Moreover, his proof was based on major contributions of many mathematicians.

We now state an open problem that is simple to describe, but that seems quite difficult to resolve.

EXAMPLE. 23

The 3x + 1 Conjecture Let T be the transformation that sends an even integer x to $x/2$ and an odd integer x to $3x + 1$. A famous conjecture, sometimes known as the $3x+1$ **conjecture**, states that for all positive integers x, when we repeatedly apply the transformation T, we will eventually reach the integer 1. For example, starting with $x = 13$, we find $T(13) = 3 \cdot 13 + 1 = 40$, $T(40) = 40/2 = 20$, $T(20) = 20/2 = 10$, $T(10) = 10/2 = 5$, $T(5) = 3 \cdot 5 + 1 =$

16, $T(16) = 8$, $T(8) = 4$, $T(4) = 2$, and $T(2) = 1$. The $3x + 1$ conjecture has been verified using computers for all integers x up to $5.6 \cdot 10^{13}$.

The $3x + 1$ conjecture has an interesting history and has attracted the attention of mathematicians since the 1950s. The conjecture has been raised many times and goes by many other names, including the Collatz problem, Hasse's algorithm, Ulam's problem, the Syracuse problem, and Kakutani's problem. Many mathematicians have been diverted from theirwork to spend time attacking this conjecture. This led to the joke that this problem was part of a conspiracy to slow down American mathematical research. ∎

Chapter 2

BASIC STRUCTURES: SETS, FUNCTIONS, SEQUENCES, SUMS

Much of discrete mathematics is devoted to the study of discrete structures, used to represent discrete objects. Many important discrete structures are built using sets, which are collections of objects.Among the discrete structures built from sets are combinations, unordered collections of objects used extensively in counting; relations, sets of ordered pairs that represent relationships between objects; graphs, sets of vertices and edges that connect vertices; and finite state machines, used to model computing machines. These are some of the topics we will study in later chapters.

The concept of a function is extremely important in discrete mathematics.Afunction assigns to each element of a first set exactly one element of a second set, where the two sets are not necessarily distinct. Functions play important roles throughout discrete mathematics. They are used to represent the computational complexity of algorithms, to study the size of sets, to count objects, and in a myriad of other ways. Useful structures such as sequences and strings are special types of functions. In this chapter, we will introduce the notion of a sequence, which represents ordered lists of elements. Furthermore, we will introduce some important types of sequences and we will show how to

define the terms of a sequence using earlier terms.We will also address the problem of identifying a sequence from its first few terms.

In our study of discrete mathematics, we will often add consecutive terms of a sequence of numbers. Because adding terms from a sequence, as well as other indexed sets of numbers, is such a common occurrence, a special notation has been developed for adding such terms. In this chapter, we will introduce the notation used to express summations.We will develop formulae for certain types of summations that appear throughout the study of discrete mathematics. For instance, we will encounter such summations in the analysis of the number of steps used by an algorithm to sort a list of numbers so that its terms are in increasing order.

The relative sizes of infinite sets can be studied by introducing the notion of the size, or cardinality, of a set. We say that a set is countable when it is finite or has the same size as the set of positive integers. In this chapter we will establish the surprising result that the set of rational numbers is countable, while the set of real numbers is not.We will also show how the concepts we discuss can be used to show that there are functions that cannot be computed using a computer program in any programming language.

2.1 Sets

2.1.1 Introduction

In this section, we study the fundamental discrete structure on which all other discrete structures are built, namely, the set. Sets are used to group objects together. Often, but not always, the objects in a set have similar properties. For instance, all the students who are currently enrolled in your school make up a set. Likewise, all the students currently taking a course in discrete mathematics at any school make up a set. In addition, those students enrolled in your school who are taking a course in discrete mathematics form a set that can be obtained by taking the elements common to the first two collections. The language of sets is a means to study such collections in an organized fashion. We now provide a definition of a set. This definition is an

intuitive definition, which is not part of a formal theory of sets.

Definition 2.1.1 *A set is an unordered collection of objects, called* ***elements or members*** *of the set. A set is said to contain its elements. We write* $a \in A$ *to denote that* a *is an element of the set* A*. The notation* $a \notin A$ *denotes that* a *is not an element of the set* A*.*

It is common for sets to be denoted using uppercase letters. Lowercase letters are usually used to denote elements of sets.

There are several ways to describe a set. One way is to list all the members of a set, when this is possible. We use a notation where all members of the set are listed between braces. For example, the notation *a, b, c, d* represents the set with the four elements *a, b, c*, and *d*. This way of describing a set is known as the **roster method**.

EXAMPLE. 1

The set V of all vowels in the English alphabet can be written as $V = \{a, e, i, o, u\}$. ∎

EXAMPLE. 2

The set O of odd positive integers less than 10 can be expressed by $O = \{1, 3, 5, 7, 9\}$. ∎

EXAMPLE. 3

Although sets are usually used to group together elements with common properties, there is nothing that prevents a set from having seemingly unrelated elements. For instance, $a, 2, Fred, NewJersey$ is the set containing the four elements *a*, 2, Fred, and New Jersey. ∎

Sometimes the roster method is used to describe a set without listing all its members. Some members of the set are listed, and then *ellipses* (...) are used when the general pattern of the elements is obvious.

EXAMPLE. 4

The set of positive integers less than 100 can be denoted by $\{1, 2, 3, \ldots, 99\}$. ■

Another way to describe a set is to use **set builder** notation. We characterize all those elements in the set by stating the property or properties they must have to be members. For instance, the set O of all odd positive integers less than 10 can be written as

$$O = \{x | x \text{ is an odd positive integer less than } 10\},$$

or, specifying the universe as the set of positive integers, as

$$O = \{x \in \mathbf{Z}^+ | x \text{ is odd and } x < 10\}.$$

We often use this type of notation to describe sets when it is impossible to list all the elements of the set. For instance, the set Q^+ of all positive rational numbers can be written as

$$Q^+ = \{x \in \mathbf{R} | x = \frac{p}{q}, \text{ for some positive integers} p \text{ and } q\}.$$

These sets, each denoted using a boldface letter, play an important role in discrete mathematics:

$\mathbf{N} = \{0, 1, 2, 3, \ldots\}$, the set of natural numbers

$\mathbf{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$, the set of **integers**

$\mathbf{Z}^+ = \{1, 2, 3, \ldots\}$, the set of **positive integers**

$\mathbf{Q} = \{p/q | p \in \mathbf{Z}, q \in \mathbf{Z}, \text{ and} q \neq 0\}$, the set of **rational numbers**

R, the set of **real numbers**

$\mathbf{R}^+$, the set of **positive real numbers**

C, the set of **complex numbers**.

(Note that some people do not consider 0 a natural number, so be careful to check how the term natural numbers is used when you read other books.)

Recall the notation for **intervals** of real numbers. When a and b are real numbers with $a < b$, we write

$$[a, b] = \{x | a \leq x \leq b\}$$
$$[a, b) = \{x | a \leq x < b\}$$
$$(a, b] = \{x | a < x \leq b\}$$
$$(a, b) = \{x | a < x < b\}$$

Note that $[a, b]$ is called the **closed interval** from a to b and (a, b) is called the **open interval** from a to b.

Sets can have other sets as members, as Example 5 illustrates.

EXAMPLE. 5

The set **N, Z,Q,R** is a set containing four elements, each of which is a set. The four elements of this set are **N**, the set of natural numbers; **Z**, the set of integers; **Q**, the set of rational numbers; and **R**, the set of real numbers. ■

Remark! Note that the concept of a datatype, or type, in computer science is built upon the concept of a set. In particular, a **datatype** or **type** is the name of a set, together with a set of operations that can be performed on objects from that set. For example, boolean is the name of the set $\{0, 1\}$ together with operators on one or more elements of this set, such as AND, OR, and NOT.

Because many mathematical statements assert that two differently specified collections of objects are really the same set, we need to understand what it means for two sets to be equal.

Definition 2.1.2 *Two sets are* ***equal*** *if and only if they have the same elements. Therefore, if A and B are sets, then A and B are equal if and only if* $\forall x(x \in A \leftrightarrow x \in B)$. *We write* $A = B$ *if A and B are equal sets.*

EXAMPLE. 6

The sets $\{1, 3, 5\}$ and $\{3, 5, 1\}$ are equal, because they have the same elements. Note that the order in which the elements of a set are listed does not matter. Note also that it does not matter if an element of a set is listed more than once, so $\{1, 3, 3, 3, 5, 5, 5, 5\}$ is the same as the set $\{1, 3, 5\}$ because they have the same elements. ■

THE EMPTY SET

There is a special set that has no elements. This set is called the **empty set**, or **null set**, and is denoted by $\emptyset$. The empty set can also be denoted by $\{\}$ (that is, we represent the empty set with a pair of braces that encloses all the elements in this set). Often, a set of elements with certain properties turns out to be the null set. For instance, the set of all positive integers that are greater than their squares is the null set.

A set with one element is called a singleton set. A common error is to confuse the empty set $\emptyset$ with the set $\{\emptyset\}$, which is a singleton set. The single element of the set $\{\emptyset\}$ is the empty set itself!A useful analogy for remembering this difference is to think of folders in a computer file system. The empty set can be thought of as an empty folder and the set consisting of just the empty set can be thought of as a folder with exactly one folder inside, namely, the empty folder.

NAIVE SET THEORY

Note that the term *object* has been used in the definition of a set, Definition 1, without specifying what an object is. This description

of a set as a collection of objects, based on the intuitive notion of an object, was first stated in 1895 by the German mathematician Georg Cantor. The theory that results from this intuitive definition of a set, and the use of the intuitive notion that for any property whatever, there is a set consisting of exactly the objects with this property, leads to **paradoxes**, or logical inconsistencies. This was shown by the English philosopher Bertrand Russell in 1902. These logical inconsistencies can be avoided by building set theory beginning with axioms. However, we will use Cantor's original version of set theory, known as **naive set theory**, in this book because all sets considered in this book can be treated consistently using Cantor's original theory. Students will find familiarity with naive set theory helpful if they go on to learn about axiomatic set theory. They will also find the development of axiomatic set theory much more abstract than the material in this text.

2.1.2 Venn Diagrams

Sets can be represented graphically using Venn diagrams, named after the English mathematician JohnVenn, who introduced their use in 1881. In Venn diagrams the universal set U, which contains all the objects under consideration, is represented by a rectangle. (Note that the universal set varies depending on which objects are of interest.) Inside this rectangle, circles or other geometrical figures are used to represent sets. Sometimes points are used to represent the particular elements of the set.Venn diagrams are often used to indicate the relationships between sets.We show how a Venn diagram can be used in Example 7.

 EXAMPLE. 7

Draw a Venn diagram that represents V, the set of vowels in the English alphabet.

Solution: We draw a rectangle to indicate the universal set U, which is the set of the 26 letters of the English alphabet. Inside this rectangle we draw a circle to represent V. Inside this circle we indicate the elements of V with points (see Figure 2.1). ∎

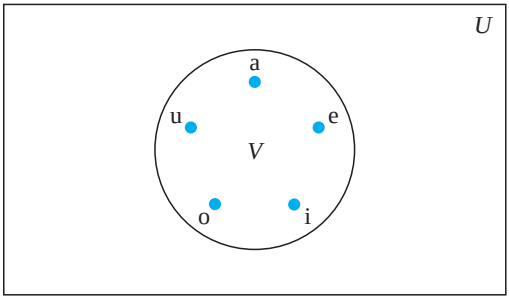


Figure 2.1: Venn Diagram for the Set of Vowels.

2.1.3 Subsets

It is common to encounter situations where the elements of one set are also the elements of a second set. We now introduce some terminology and notation to express such relationships between sets.

Definition 2.1.3 *The set A is a **subset** of B if and only if every element of A is also an element of B. We use the notation $A \subseteq B$ to indicate that A is a subset of the set B.*

We see that $A \subseteq B$ if and only if the quantification

$$\forall x(x \in A \rightarrow x \in B)$$

is true. Note that to show that A is not a subset of B we need only find one element $x \in A$ with $x \notin B$. Such an x is a counterexample to the claim that $x \in A$ implies $x \in B$.

We have these useful rules for determining whether one set is a subset of another:

Showing that A is a Subset of B To show that $A \subseteq B$, show that if x belongs to A then x also belongs to B.

Showing that A is Not a Subset of B To show that $A \nsubseteq B$, find a single $x \in A$ such that $x \notin B$.

EXAMPLE. 8

The set of all odd positive integers less than 10 is a subset of the set of all positive integers less than 10, the set of rational numbers is a subset of the set of real numbers, the set of all computer science majors at your school is a subset of the set of all students at your school, and the set of all people in China is a subset of the set of all people in China (that is, it is a subset of itself). Each of these facts follows immediately by noting that an element that belongs to the first set in each pair of sets also belongs to the second set in that pair. ∎

EXAMPLE. 9

The set of integers with squares less than 100 is not a subset of the set of nonnegative integers because -1 is in the former set [as $(-1)^2 < 100$], but not the later set. The set of people who have taken discrete mathematics at your school is not a subset of the set of all computer science majors at your school if there is at least one student who has taken discrete mathematics who is not a computer science major. ∎

Theorem 2.1.1 shows that every nonempty set S is guaranteed to have at least two subsets, the empty set and the set S itself, that is, $\emptyset \subseteq S$ and $S \subseteq S$.

THEOREM 2.1.1

For every set S, (i) $\emptyset \subseteq S$ and (ii) $S \subseteq S$.

Proof: We will prove *(i)* and leave the proof of *(ii)* as an exercise.

Let S be a set. To show that $\emptyset \subseteq S$, we must show that $\forall x(x \in \emptyset \rightarrow x \in S)$ is true. Because the empty set contains no elements, it follows that $x \in \emptyset$ is always false. It follows that the conditional statement $x \in \emptyset \rightarrow x \in S$ is always true, because its hypothesis is always false and a conditional statement with a false hypothesis is true. Therefore,

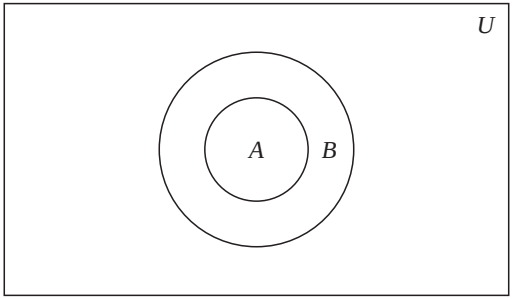


Figure 2.2: Venn Diagram Showing that A Is a Subset of B.

$\forall x(x \in \emptyset \rightarrow x \in S)$ is true. This completes the proof of *(i)*. Note that this is an example of a vacuous proof. ∎

When we wish to emphasize that a set A is a subset of a set B but that $A \neq B$, we write $A \subset B$ and say that A is a **proper subset** of B. For $A \subset B$ to be true, it must be the case that $A \subseteq B$ and there must exist an element x of B that is not an element of A. That is, A is a proper subset of B if and only if

$$\forall x(x \in A \rightarrow x \in B) \wedge \exists x(x \in B \wedge x \notin A)$$

is true. Venn diagrams can be used to illustrate that a set A is a subset of a set B. We draw the universal set U as a rectangle.Within this rectangle we draw a circle for B. Because A is a subset of B, we draw the circle for A within the circle for B. This relationship is shown in Figure 2.2.

A useful way to show that two sets have the same elements is to show that each set is a subset of the other. In other words, we can show that if A and B are sets with $A \subseteq B$ and $B \subseteq A$, then $A = B$. That is, $A = B$ if and only if $\forall x(x \in A \rightarrow x \in B)$ and $\forall x(x \in B \rightarrow x \in A)$ or equivalently if and only if $\forall x(x \in A \leftrightarrow x \in B)$, which is what it means for the A and B to be equal. Because this method of showing two sets are equal is so useful, we highlight it here.

Showing Two Sets are Equal To show that two sets A and B are equal, show that $A \subseteq B$ and $B \subseteq A$.

Sets may have other sets as members. For instance, we have the sets

$$A = \{\emptyset, \{a\}, \{b\}, \{a, b\}\} \text{ and } B = \{x | x \text{ is a subset of the set } \{a,b\}\}$$

Note that these two sets are equal, that is, $A = B$. Also note that $\{a\} \in A$, but $a \notin A$.

2.1.4 The Size of a Set

Sets are used extensively in counting problems, and for such applications we need to discuss the sizes of sets.

Definition 2.1.4 *Let S be a set. If there are exactly n distinct elements in S where n is a nonnegative integer, we say that S is a* ***finite*** *set and that n is the* ***cardinality*** *of S. The cardinality of S is denoted by $|S|$.*

> **Remark!** The term *cardinality* comes from the common usage of the term *cardinal number* as the size of a finite set.

EXAMPLE. 10

Let A be the set of odd positive integers less than 10. Then $|A| = 5$. ∎

EXAMPLE. 11

Let S be the set of letters in the English alphabet. Then $|S| = 26$. ∎

EXAMPLE. 12

Because the null set has no elements, it follows that $|\emptyset| = 0$. ∎

We will also be interested in sets that are not finite.

Definition 2.1.5 *A set is said to be **infinite** if it is not finite.*

EXAMPLE. 13

The set of positive integers is infinite. ∎

2.1.5 Power Sets

Many problems involve testing all combinations of elements of a set to see if they satisfy some property. To consider all such combinations of elements of a set S, we build a new set that has as its members all the subsets of S.

Definition 2.1.6 *Given a set S, the power set of S is the set of all subsets of the set S. The power set of S is denoted by $\mathcal{P}(S)$.*

EXAMPLE. 14

What is the power set of the set $\{0, 1, 2\}$?

Solution: The power set $\mathcal{P}(\{0, 1, 2\})$ is the set of all subsets of $\{0, 1, 2\}$. Hence,

$$\mathcal{P}(\{0, 1, 2\}) = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}\}.$$

The empty set and the set itself are members of this set of subsets. ∎

EXAMPLE. 15

What is the power set of the empty set? What is the power set of the set $\{\emptyset\}$?

Solution: The empty set has exactly one subset, namely, itself. Consequently,

$$\mathcal{P}(\emptyset) = \{\emptyset\}.$$

The set $\{\emptyset\}$ has exactly two subsets, namely, $\emptyset$ and the set $\{\emptyset\}$ itself. Therefore,

$$\mathcal{P}(\{\emptyset\}) = \{\emptyset, \{\emptyset\}\}. \quad \blacksquare$$

If a set has n elements, then its power set has 2^n elements. We will demonstrate this fact in several ways in subsequent sections of the text.

2.1.6 Cartesian Products

The order of elements in a collection is often important. Because sets are unordered, a different structure is needed to represent ordered collections. This is provided by ordered n-**tuples**.

Definition 2.1.7 *The* ***ordered n-tuple*** $(a_1, a_2, \ldots, a_n)$ *is the ordered collection that has* a_1 *as its first element,* a_2 *as its second element, ... , and* a_n *as its nth element.*

We say that two ordered n-tuples are equal if and only if each corresponding pair of their elements is equal. In other words, $(a_1, a_2, \ldots, a_n) = (b_1, b_2, \ldots, b_n)$ if and only if $a_i = b_i$, for $i = 1, 2, \ldots, n$. In particular, ordered 2-tuples are called **ordered pairs**. The ordered pairs (a, b) and (c, d) are equal if and only if $a = c$ and $b = d$. Note that (a, b) and (b, a) are not equal unless $a = b$.

Definition 2.1.8 *Let* A *and* B *be sets. The* ***Cartesian product*** *of* A *and* B*, denoted by* $A \times B$*, is the set of all ordered pairs* (a, b)*, where* $a \in A$ *and* $b \in B$*. Hence,*

$$A \times B = \{(a, b) | a \in A \wedge b \in B\}.$$

EXAMPLE. 16

Let A represent the set of all students at a university, and let B represent the set of all courses offered at the university. What is the Cartesian product $A \times B$ and how can it be used?

Solution: The Cartesian product $A \times B$ consists of all the ordered pairs of the form (a, b), where a is a student at the university and b is a course offered at the university. One way to use the set $A \times B$ is to represent all possible enrollments of students in courses at the university. ∎

 EXAMPLE. 17

What is the Cartesian product of $A = \{1, 2\}$ and $B = \{a, b, c\}$?

Solution: The Cartesian product $A \times B$ is

$$A \times B = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c)\}. \quad ∎$$

Note that the Cartesian products $A \times B$ and $B \times A$ are not equal, unless $A = \emptyset$ or $B = \emptyset$ (so that $A \times B = \emptyset$) or $A = B$. This is illustrated in Example 18.

 EXAMPLE. 18

Show that the Cartesian product $B \times A$ is not equal to the Cartesian product $A \times B$, where A and B are as in Example 17.

Solution: The Cartesian product $B \times A$ is

$$B \times A = \{(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}.$$

This is not equal to $A \times B$, which was found in Example 17. ∎

The Cartesian product of more than two sets can also be defined.

Definition 2.1.9 *The **Cartesian product** of the sets $A_1, A_2, \ldots, A_n$, denoted by $A_1 \times A_2 \times \ldots \times A_n$, is the set of ordered n-tuples $(a_1, a_2, \ldots, a_n)$, where a_i belongs to A_i for $i = 1, 2, \ldots, n$. In other words,*

$$A_1 \times A_2 \times \ldots \times A_n = \{(a_1, a_2, \ldots, a_n) | a_i \in A_i \text{ for } i = 1, 2, \ldots, n\}. \quad ∎$$

 EXAMPLE. 19

What is the Cartesian product $A \times B \times C$, where $A = \{0, 1\}$, $B = \{1, 2\}$, and $C = \{0, 1, 2\}$?

Solution: The Cartesian product $A \times B \times C$ consists of all ordered triples (a, b, c), where $a \in A$, $b \in B$, and $c \in C$. Hence,

$$A \times B \times C = \{(0,1,0), (0,1,1), (0,1,2), (0,2,0), (0,2,1), (0,2,2), (1,1,0), (1,1,1), (1,1,2), (1,2,0), (1,2,1), (1,2,2)\}$$ ■.

Remark! Note that when A, B, and C are sets, $(A \times B) \times C$ is not the same as $A \times B \times C$.

We use the notation A^2 to denote $A \times A$, the Cartesian product of the set A with itself. Similarly, $A_3 = A \times A \times A$, $A_4 = A \times A \times A \times A$, and so on. More generally,

$$A_n = \{(a_1, a_2, \ldots, a_n) | a_i \in A \text{ for} i = 1, 2, \ldots, n\}.$$

EXAMPLE. 20

Suppose that $A = \{1, 2\}$. It follows that $A_2 = \{(1,1), (1,2), (2,1), (2,2)\}$ and $A_3 = \{(1,1,1), (1,1,2), (1,2,1), (1,2,2), (2,1,1), (2,1,2), (2,2,1), (2,2,2)\}$. ■

A subset R of the Cartesian product $A \times B$ is called a **relation** from the set A to the set B. The elements of R are ordered pairs, where the first element belongs to A and the second to B. For example, $R = \{(a, 0), (a, 1), (a, 3), (b, 1), (b, 2), (c, 0), (c, 3)\}$ is a relation from the set $\{a, b, c\}$ to the set $\{0, 1, 2, 3\}$. A relation from a set A to itself is called a relation on A.

EXAMPLE. 21

What are the ordered pairs in the less than or equal to relation, which contains (a, b) if $a \leq b$, on the set $\{0, 1, 2, 3\}$?

Solution: The ordered pair (a, b) belongs to R if and only if both a and b belong to $\{0, 1, 2, 3\}$ and $a \leq b$. Consequently, the ordered pairs in R are $(0, 0), (0, 1), (0, 2), (0, 3), (1, 1), (1, 2), (1, 3), (2, 2), (2, 3)$, and $(3, 3)$. ∎

2.1.7 Using Set Notation with Quantifiers

Sometimes we restrict the domain of a quantified statement explicitly by making use of a particular notation. For example, $\forall x \in S(P(x))$ denotes the universal quantification of $P(x)$ over all elements in the set S. In other words, $\forall x \in S(P(x))$ is shorthand for $\forall x(x \in S \rightarrow P(x))$. Similarly, $\exists x \in S(P(x))$ denotes the existential quantification of $P(x)$ over all elements in S. That is, $\exists x \in S(P(x))$ is shorthand for $\exists x(x \in S \wedge P(x))$.

EXAMPLE. 22

What do the statements $\forall x \in \mathbf{R}(x^2 \geq 0)$ and $\exists x \in \mathbf{Z}(x^2 = 1)$ mean?

Solution: The statement $\forall x \in \mathbf{R}(x^2 \geq 0)$ states that for every real number $x, x^2 \geq 0$. This statement can be expressed as "The square of every real number is nonnegative". This is a true statement.

The statement $\exists x \in \mathbf{Z}(x^2 = 1)$ states that there exists an integer x such that $x^2 = 1$. This statement can be expressed as "There is an integer whose square is 1". This is also a true statement because $x = 1$ is such an integer (as is -1). ∎

2.1.8 Truth Sets and Quantifiers

We will now tie together concepts from set theory and from predicate logic. Given a predicate P, and a domain D, we define the truth set of P to be the set of elements x in D for which $P(x)$ is true. The truth set of $P(x)$ is denoted by $\{x \in D | P(x)\}$.

EXAMPLE. 23

What are the truth sets of the predicates $P(x), Q(x)$, and $R(x)$, where the

domain is the set of integers and $P(x)$ is "$|x| = 1$," $Q(x)$ is "$x^2 = 2$," and $R(x)$ is "$|x| = x$."

Solution: The truth set of P, $\{x \in \mathbf{Z} | |x| = 1\}$, is the set of integers for which $|x| = 1$. Because $|x| = 1$ when $x = 1$ or $x = -1$, and for no other integers x, we see that the truth set of P is the set $\{-1, 1\}$.

The truth set of Q, $\{x \in \mathbf{Z} | x^2 = 2\}$, is the set of integers for which $x^2 = 2$. This is the empty set because there are no integers x for which $x^2 = 2$.

The truth set of R, $\{x \in \mathbf{Z} | |x| = x\}$, is the set of integers for which $|x| = x$. Because $|x| = x$ if and only if $x \geq 0$, it follows that the truth set of R is N, the set of nonnegative integers. ∎

Note that $\forall x P(x)$ is true over the domain U if and only if the truth set of P is the set U. Likewise, $\exists x P(x)$ is true over the domain U if and only if the truth set of P is nonempty.

2.2 Set Operations

2.2.1 Introduction

Two, or more, sets can be combined in many different ways. For instance, starting with the set of mathematics majors at your school and the set of computer science majors at your school, we can form the set of students who are mathematics majors or computer science majors, the set of students who are joint majors in mathematics and computer science, the set of all students not majoring in mathematics, and so on.

Definition 2.2.1 *Let A and B be sets. The union of the sets A and B, denoted by $A \cup B$, is the set that contains those elements that are either in A or in B, or in both.*

An element x belongs to the union of the sets A and B if and only if x belongs to A or x belongs to B. This tells us that

$$A \cup B = \{x | x \in A \vee x \in B\}.$$

The Venn diagram shown in Figure 2.3 represents the union of two sets A and B. The area that represents $A \cup B$ is the shaded area within either the circle representing A or the circle representing B.

We will give some examples of the union of sets.

EXAMPLE. 1

The union of the sets $\{1, 3, 5\}$ and $\{1, 2, 3\}$ is the set $\{1, 2, 3, 5\}$; that is, $\{1, 3, 5\} \cup \{1, 2, 3\} = \{1, 2, 3, 5\}$. ∎

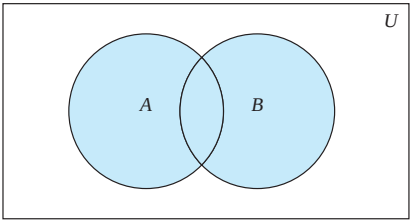
EXAMPLE. 2

The union of the set of all computer science majors at your school and the set of all mathematics majors at your school is the set of students at your school who are majoring either in mathematics or in computer science. ∎

Definition 2.2.2 *Let A and B be sets. The intersection of the sets A and B, denoted by $A \cap B$, is the set containing those elements in both A and B.*

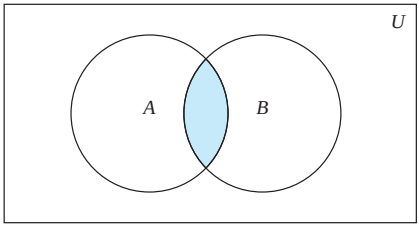
An element x belongs to the intersection of the sets A and B if and only if x belongs to A and x belongs to B. This tells us that

$$A \cap B = \{x | x \in A \wedge x \in B\}. \quad ∎$$



$A \cup B$ is shaded

Figure 2.3: Venn Diagram of the Union of A and B.



$A \cap B$ is shaded

Figure 2.4: Venn Diagram of the Intersection of A and B.

TheVenn diagram shown in Figure 2.4 represents the intersection of two sets A and B. The shaded area that is within both the circles

representing the sets A and B is the area that represents the intersection of A and B.

We give some examples of the intersection of sets.

EXAMPLE. 3

The intersection of the sets $\{1,3,5\}$ and $\{1,2,3\}$ is the set $\{1,3\}$; that is, $\{1,3,5\} \cap \{1,2,3\} = \{1,3\}$. ∎

EXAMPLE. 4

The intersection of the set of all computer science majors at your school and the set of all mathematics majors is the set of all students who are joint majors in mathematics and computer science. ∎

Definition 2.2.3 *Two sets are called* ***disjoint*** *if their intersection is the empty set.*

EXAMPLE. 5

Let $A = \{1,3,5,7,9\}$ and $B = \{2,4,6,8,10\}$. Because $A \cap B = \emptyset$, A and B are disjoint. ∎

We are often interested in finding the cardinality of a union of two finite sets A and B. Note that $|A|+|B|$ counts each element that is in A but not in B or in B but not in A exactly once, and each element that is in both A and B exactly twice. Thus, if the number of elements that are in both A and B is subtracted from $|A| + |B|$, elements in $A \cap B$ will be counted only once.

Hence,

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

The generalization of this result to unions of an arbitrary number of sets is called the **principle of inclusion–exclusion**. The principle of inclusion–exclusion is an important technique used in enumeration.

There are other important ways to combine sets.

Definition 2.2.4 *Let A and B be sets. The **difference** of A and B, denoted by $A - B$, is the set containing those elements that are in A but not in B. The difference of A and B is also called the **complement of B with respect to A**.*

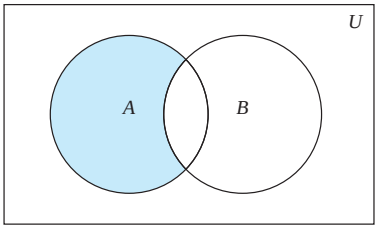
Remark! The difference of sets A and B is sometimes denoted by $A \backslash B$.

An element x belongs to the difference of A and B if and only if $x \in A$ and $x \notin B$. This tells us that

$$A - B = \{x | x \in A \wedge x \notin B\}.$$

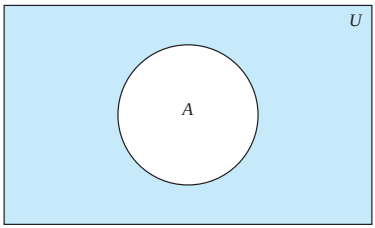
The Venn diagram shown in Figure 2.5 represents the difference of the sets A and B. The shaded area inside the circle that represents A and outside the circle that represents B is the area that represents $A - B$.

We give some examples of differences of sets.



$A - B$ is shaded

Figure 2.5: Venn Diagram for the Difference of A and B.



$\bar{A}$ is shaded

Figure 2.6: Venn Diagram for the Complement of the Set A.

EXAMPLE. 6

The difference of $\{1,3,5\}$ and $\{1,2,3\}$ is the set $\{5\}$; that is, $\{1,3,5\} - \{1,2,3\} = \{5\}$. This is different from the difference of $\{1,2,3\}$ and $\{1,3,5\}$, which is the set $\{2\}$. ∎

EXAMPLE. 7

The difference of the set of computer science majors at your school and the set of mathematics majors at your school is the set of all computer science majors at your school who are not also mathematics majors. ∎

Once the universal set U has been specified, the complement of a set can be defined.

Definition 2.2.5 *Let U be the universal set. The **complement** of the set A, denoted by $\bar{A}$, is the complement of A with respect to U. Therefore, the complement of the set A is $U - A$.*

An element belongs to A if and only if $x \notin A$. This tells us that

$$A = \{x \in U | x \notin A\}.$$

In Figure 2.6 the shaded area outside the circle representing A is the area representing A.

We give some examples of the complement of a set.

EXAMPLE. 8

Let $A = \{a, e, i, o, u\}$ (where the universal set is the set of letters of the English alphabet). Then $\bar{A} = \{b, c, d, f, g, h, j, k, l, m, n, p, q, r, s, t, v, w, x, y, z\}$. ∎

EXAMPLE. 9

Let A be the set of positive integers greater than 10 (with universal set the set of all positive integers). Then $\bar{A} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. ∎

It is left to show that we can express the difference of A and B as the intersection of A and the complement of B. That is,

$$A - B = A \cap \bar{B}.$$

2.2.2 Set Identities

Table 2.1 lists the most important set identities. We will prove several of these identities here, using three different methods. These methods are presented to illustrate that there are often many different approaches to the solution of a problem. The proofs of the remaining identities will be left as exercises. In fact, the set identities given can be proved directly from the corresponding logical equivalences. Furthermore, both are special cases of identities that hold for Boolean algebra.

One way to show that two sets are equal is to show that each is a subset of the other. Recall that to show that one set is a subset of a second set, we can show that if an element belongs to the first set, then it must also belong to the second set.We generally use a direct proof to do this. We illustrate this type of proof by establishing the first of De Morgan's laws.

EXAMPLE. 10

Prove that $\overline{A \cap B} = \bar{A} \cup \bar{B}$.

Solution: We will prove that the two sets $\overline{A \cap B}$ and $\bar{A} \cup \bar{B}$ are equal by showing that each set is a subset of the other.

First, we will show that $\overline{A \cap B} \subseteq \bar{A} \cup \bar{B}$. We do this by showing that if x is in $\overline{A \cap B}$, then it must also be in $\bar{A} \cup \bar{B}$. Now suppose that $x \in \overline{A \cap B}$. By the definition of complement, $x \notin A \cap B$. Using the definition of intersection, we see that the proposition $\neg((x \in A) \wedge (x \in B))$ is true.

By applying De Morgan's law for propositions, we see that $\neg(x \in A)$ or $\neg(x \in B)$. Using the definition of negation of propositions, we have $x \notin A$ or $x \notin B$. Using the definition of the complement of a set, we see that this implies that $x \in \bar{A}$ or $x \in \bar{B}$. Consequently, by the definition of union, we see that $x \in \bar{A} \cup \bar{B}$. We have now shown that $\overline{A \cap B} \subseteq \bar{A} \cup \bar{B}$.

Next, we will show that $\bar{A} \cup \bar{B} \subseteq \overline{A \cap B}$. We do this by showing that if x is in $\bar{A} \cup \bar{B}$, then it must also be in $\overline{A \cap B}$. Now suppose that $x \in \bar{A} \cup \bar{B}$. By the definition of union, we know that $x \in \bar{A}$ or $x \in \bar{B}$. Using the definition

Table 2.1: Set Identities

Identity	**Name**
$A \cap U = a$ $A \cup \emptyset = A$	Identity laws
$A \cup U = U$ $A \cap \emptyset = \emptyset$	Domination laws
$A \cup A = A$ $A \cap A = A$	Idempotent laws
$\overline{(\bar{A})} = A$	Complementation law
$A \cup B = B \cup A$ $A \cap B = B \cap A$	Commutative laws
$A \cup (B \cup C) = (A \cup B) \cup C$ $A \cap (B \cap C) = (A \cap B) \cap C$	Associative laws
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	Distributive laws
$A \bar{\cap} B = \bar{A} \cup \bar{B}$ $A \bar{\cup} B = \bar{A} \cap \bar{B}$	De Morgan's laws
$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$	Absorption laws
$A \cup \bar{A} = U$ $A \cap \bar{A} = \emptyset$	Complement laws

of complement, we see that $x \notin A$ or $x \notin B$. Consequently, the proposition $\neg(x \in A) \vee \neg(x \in B)$ is true.

By De Morgan's law for propositions, we conclude that $\neg((x \in A) \wedge (x \in B))$ is true. By the definition of intersection, it follows that $\neg(x \in A \cap B)$. We now use the definition of complement to conclude that $x \in \overline{A \cap B}$. This shows that $\bar{A} \cup \bar{B} \subseteq \overline{A \cap B}$. Because we have shown that each set is a subset of the other, the two sets are equal, and the identity is proved. ∎

We can more succinctly express the reasoning used in Example 10 using set builder notation, as Example 11 illustrates.

EXAMPLE. 11

Use set builder notation and logical equivalences to establish the first De Morgan law $\overline{A \cap B} = \bar{A} \cup \bar{B}$.

Solution: We can prove this identity with the following steps.

$\overline{A \cap B}$	$= \{x \mid x \notin A \cap B\}$	by definition of complement
	$= \{x \mid \neg(x \in (A \cap B))\}$	by definition of does not belong symbol
	$= \{x \mid \neg(x \in A \wedge x \in B)\}$	by definition of intersection
	$= \{x \mid \neg(x \in A) \vee \neg(x \in B)\}$	by the first De Morgan law for logical equivalences
	$= \{x \mid x \notin A \vee x \notin B\}$	by definition of does not belong symbol
	$= \{x \mid x \in \bar{A} \vee x \in \bar{B}\}$	by definition of complement
	$= \{x \mid x \in \bar{A} \cup \bar{B}\}$	by definition of union
	$= \bar{A} \cup \bar{B}$	by meaning of set builder notation

Note that besides the definitions of complement, union, set membership, and set builder notation, this proof uses the second De Morgan law for logical equivalences. ∎

Proving a set identity involving more than two sets by showing each side of the identity is a subset of the other often requires that we keep track of different cases, as illustrated by the proof in Example 12 of one of the distributive laws for sets.

EXAMPLE. 12

Prove the second distributive law from Table 2.1, which states that $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ for all sets A, B, and C.

Solution: We will prove this identity by showing that each side is a subset of the other side.

Suppose that $x \in A \cap (B \cup C)$. Then $x \in A$ and $x \in B \cup C$. By the definition of union, it follows that $x \in A$, and $x \in B$ or $x \in C$ (or both). In other words, we know that the compound proposition $(x \in A) \wedge ((x \in B) \vee (x \in C))$ is true. By the distributive law for conjunction over disjunction, it follows that $((x \in A) \wedge (x \in B)) \vee ((x \in A) \wedge (x \in C))$. We conclude that either $x \in A$ and $x \in B$, or $x \in A$ and $x \in C$. By the definition of intersection, it follows that $x \in A \cap B$ or $x \in A \cap C$. Using the definition of union, we conclude that $x \in (A \cap B) \cup (A \cap C)$. We conclude that $A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$.

Now suppose that $x \in (A \cap B) \cup (A \cap C)$. Then, by the definition of union, $x \in A \cap B$ or $x \in A \cap C$. By the definition of intersection, it follows that $x \in A$ and $x \in B$ or that $x \in A$ and $x \in C$. From this we see that $x \in A$, and $x \in B$ or $x \in C$. Consequently, by the definition of union we see that $x \in A$ and $x \in B \cup C$. Furthermore, by the definition of intersection, it follows that $x \in A \cap (B \cup C)$. We conclude that $(A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$. This completes the proof of the identity. ∎

Set identities can also be proved using **membership tables**. We consider each combination of sets that an element can belong to and verify that elements in the same combinations of sets belong to both the sets in the identity. To indicate that an element is in a set, a 1 is used; to indicate that an element is not in a set, a 0 is used.

EXAMPLE. 13

Use a membership table to show that $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

Solution: The membership table for these combinations of sets is shown in Table 2.2. This table has eight rows. Because the columns for $A \cap (B \cup C)$ and $(A \cap B) \cup (A \cap C)$ are the same, the identity is valid. ∎

Additional set identities can be established using those thatwehave already proved. Consider Example 14.

Table 2.2: A Membership Table for the Distributive Property.

A	B	C	$B \cup C$	$A \cap (B \cup C)$	$A \cap B$	$A \cap C$	$(A \cap B) \cup (A \cap C)$
1	1	1	1	1	1	1	1
1	1	0	1	1	1	0	1
1	0	1	1	1	0	1	1
1	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0

EXAMPLE. 14

Let A, B, and C be sets. Show that $\overline{A \cup (B \cap C)} = (\bar{C} \cup \bar{B}) \cap \bar{A}$.

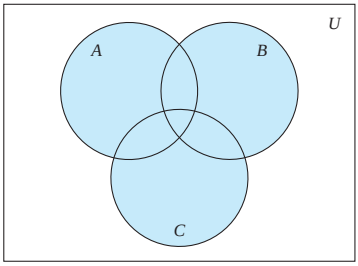
Solution: We have

$$\begin{aligned}
\overline{A \cup (B \cap C)} &= \bar{A} \cap (\overline{B \cap C}) && \text{by the first De Morgan law} \\
&= \bar{A} \cap (\bar{B} \cup \bar{C}) && \text{by the second De Morgan law} \\
&= (\bar{B} \cup \bar{C}) \cap \bar{A} && \text{by the commutative law for intersections} \\
&= (\bar{C} \cup \bar{B}) \cap \bar{A} && \text{by the commutative law for unions.}
\end{aligned}$$

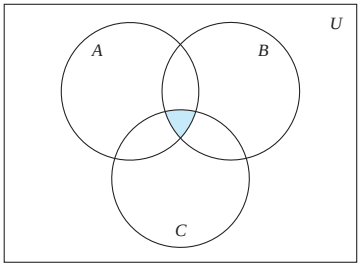
■

2.2.3 Generalized Unions and Intersections

Because unions and intersections of sets satisfy associative laws, the sets $A \cup B \cup C$ and $A \cap B \cap C$ are well defined; that is, the meaning of this notation is unambiguous when A, B, and C are sets. That is, we do not have to use parentheses to indicate which operation comes first because $A \cup (B \cup C) = (A \cup B) \cup C$ and $A \cap (B \cap C) = (A \cap B) \cap C$. Note that $A \cup B \cup C$ contains those elements that are in at least one of the sets A, B, and C, and that $A \cap B \cap C$ contains those elements that are in all of A, B, and C. These combinations of the three sets, A, B, and C, are shown in Figure 2.7.



a)$A \cup B \cup C$ is shaded



b)$A \cap B \cap C$ is shaded

Figure 2.7: The Union and Intersection of A, B, and C .

EXAMPLE. 15

Let $A = \{0, 2, 4, 6, 8\}$, $B = \{0, 1, 2, 3, 4\}$, and $C = \{0, 3, 6, 9\}$. What are $A \cup B \cup C$ and $A \cap B \cap C$?

Solution: The set $A \cup B \cup C$ contains those elements in at least one of A, B, and C. Hence,

$$A \cup B \cup C = \{0, 1, 2, 3, 4, 6, 8, 9\}.$$

The set $A \cap B \cap C$ contains those elements in all three of A, B, and C. Thus,

$$A \cap B \cap C = \{0\}. \quad \blacksquare$$

We can also consider unions and intersections of an arbitrary number of sets.We introduce these definitions.

Definition 2.2.6 *The **union** of a collection of sets is the set that contains those elements that are members of at least one set in the collection.*

We use the notation

$$A_1 \cup A_2 \cup \ldots \cup A_n = \bigcup_{i=1}^{n} A_i$$

to denote the union of the sets $A_1, A_2, \ldots, A_n$.

Definition 2.2.7 *The **intersection** of a collection of sets is the set that contains those elements that are members of all the sets in the collection.*

We use the notation

$$A_1 \cap A_2 \cap \ldots \cap A_n = \bigcap_{i=1}^{n} A_i$$

to denote the union of the sets $A_1, A_2, \ldots, A_n$. We illustrate generalized unions and intersections with Example 16.

 EXAMPLE. 16

For $i = 1, 2, \ldots$, let $A_i = \{i, i+1, i+2, \ldots\}$. Then

$$\bigcup_{i=1}^{n} A_i = \bigcup_{i=1}^{n} \{i, i+1, i+2, \ldots\} = \{1, 2, 3, \ldots\},$$

and

$$\bigcap_{i=1}^{n} A_i = \bigcap_{i=1}^{n} \{i, i+1, i+2, \ldots\} = \{n, n+1, n+2, \ldots\} = A_n \quad \blacksquare$$

We can extend the notation we have introduced for unions and intersections to other families of sets. In particular, we use the notation

$$A_1 \cup A_2 \cup \ldots \cup A_n \cup \ldots = \bigcup_{i=1}^{\infty} A_i$$

to denote the union of the sets $A_1, A_2, \ldots, A_n, \ldots$. Similarly, the intersection of these sets is denoted by

$$A_1 \cap A_2 \cap \ldots \cap A_n \cap \ldots = \bigcap_{i=1}^{\infty} A_i$$

More generally, when I is a set, the notations $\bigcap_{i \in I} A_i$ and $\bigcup_{i \in I} A_i$ are used to denote the intersection and union of the sets A_i for $i \in I$,

respectively. Note that we have $\bigcap_{i\in I} A_i = \{x|\forall i \in I(x \in A_i)\}$ and $\bigcup_{i\in I} A_i = \{x|\exists i \in I(x \in A_i)\}$.

EXAMPLE. 17

Suppose that $A_i = \{1, 2, 3, \ldots, i\}$ for $i = 1, 2, 3, \ldots$. Then,

$$\bigcup_{i=1}^{\infty} A_i = \bigcup_{i=1}^{\infty} \{1, 2, 3, \ldots, i\} = \mathbf{Z}^+$$

and

$$\bigcap_{i=1}^{\infty} A_i = \bigcap_{i=1}^{\infty} \{1, 2, 3, \ldots, i\} = \{1\}$$

To see that the union of these sets is the set of positive integers, note that every positive integer n is in at least one of the sets, because it belongs to $A_n = \{1, 2, \ldots, n\}$, and every element of the sets in the union is a positive integer. To see that the intersection of these sets is the set $\{1\}$, note that the only element that belongs to all the sets $A_1, A_2, \ldots$ is 1. To see this note that $A_1 = \{1\}$ and $1 \in A_i$ for $i = 1, 2, \ldots$. ∎

2.2.4 Computer Representation of Sets

There are various ways to represent sets using a computer. One method is to store the elements of the set in an unordered fashion. However, if this is done, the operations of computing the union, intersection, or difference of two sets would be time-consuming, because each of these operations would require a large amount of searching for elements. We will present a method for storing elements using an arbitrary ordering of the elements of the universal set. This method of representing sets makes computing combinations of sets easy.

Assume that the universal set U is finite (and of reasonable size so that the number of elements of U is not larger than the memory size of the computer being used). First, specify an arbitrary ordering of the elements of U, for instance $a_1, a_2, \ldots, a_n$. Represent a subset A of U with the bit string of length n, where the ith bit in this string is 1 if ai belongs to A and is 0 if ai does not belong to A. Example 18 illustrates this technique.

EXAMPLE. 18

Let $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, and the ordering of elements of U has the elements in increasing order; that is, $a_i = i$. What bit strings represent the subset of all odd integers in U, the subset of all even integers in U, and the subset of integers not exceeding 5 in U?

Solution: The bit string that represents the set of odd integers in U, namely, $\{1, 3, 5, 7, 9\}$, has a one bit in the first, third, fifth, seventh, and ninth positions, and a zero elsewhere. It is

10 1010 1010.

(We have split this bit string of length ten into blocks of length four for easy reading.) Similarly, we represent the subset of all even integers in U, namely, $\{2, 4, 6, 8, 10\}$, by the string

01 0101 0101.

The set of all integers in U that do not exceed 5, namely, $\{1, 2, 3, 4, 5\}$, is represented by the string

11 1110 0000. ∎

Using bit strings to represent sets, it is easy to find complements of sets and unions, intersections, and differences of sets. To find the bit string for the complement of a set from the bit string for that set, we simply change each 1 to a 0 and each 0 to 1, because $x \in A$ if and only if $x \notin A$. Note that this operation corresponds to taking the negation of each bit when we associate a bit with a truth value—with 1 representing true and 0 representing false.

EXAMPLE. 19

We have seen that the bit string for the set $\{1, 3, 5, 7, 9\}$ (with universal set $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$) is

10 1010 1010.

What is the bit string for the complement of this set?

Solution: The bit string for the complement of this set is obtained by replacing 0s with 1s and vice versa. This yields the string

$$01\ 0101\ 0101,$$

which corresponds to the set $\{2, 4, 6, 8, 10\}$. ∎

To obtain the bit string for the union and intersection of two sets we perform bitwise Boolean operations on the bit strings representing the two sets. The bit in the ith position of the bit string of the union is 1 if either of the bits in the ith position in the two strings is 1 (or both are 1), and is 0 when both bits are 0. Hence, the bit string for the union is the bitwise *OR* of the bit strings for the two sets. The bit in the ith position of the bit string of the intersection is 1 when the bits in the corresponding position in the two strings are both 1, and is 0 when either of the two bits is 0 (or both are). Hence, the bit string for the intersection is the bitwise *AND* of the bit strings for the two sets.

EXAMPLE. 20

The bit strings for the sets $\{1, 2, 3, 4, 5\}$ and $\{1, 3, 5, 7, 9\}$ are 11 1110 0000 and 10 1010 1010, respectively. Use bit strings to find the union and intersection of these sets.

Solution: The bit string for the union of these sets is

$$11\ 1110\ 0000 \vee 10\ 1010\ 1010 = 11\ 1110\ 1010,$$

which corresponds to the set $\{1, 2, 3, 4, 5, 7, 9\}$. The bit string for the intersection of these sets is

$$11\ 1110\ 0000 \wedge 10\ 1010\ 1010 = 10\ 1010\ 0000,$$

which corresponds to the set $\{1, 3, 5\}$. ∎

2.3 Functions

Introduction

In many instances we assign to each element of a set a particular element of a second set (which may be the same as the first). For example, suppose that each student in a discrete mathematics class is assigned a letter grade from the set $\{A, B, C, D, F\}$. And suppose that the grades are A for Adams, C for Chou, B for Goodfriend, A for Rodriguez, and F for Stevens. This assignment of grades is illustrated in Figure 2.8.

This assignment is an example of a function. The concept of a function is extremely important in mathematics and computer science. For example, in discrete mathematics functions are used in the definition of such discrete structures as sequences and strings. Functions are also used to represent how long it takes a computer to solve problems of a given size. Many computer programs and subroutines are designed to calculate values of functions. Recursive functions, which are functions defined in terms of themselves, are used throughout computer science. This section reviews the basic concepts involving functions needed in discrete mathematics.

Definition 2.3.1 *Let A and B be nonempty sets. A **function** f from A to B is an assignment of exactly one element of B to each element of A. We write $f(a) = b$ if b is the unique element of B assigned by the function f to the element a of A. If f is a function from A to B, we write $f : A \to B$.*

> **Remark!** Functions are sometimes also called **mappings** or **transformations**.

Functions are specified in many different ways. Sometimes we explicitly state the assignments, as in Figure 2.8. Often we give a formula, such as $f(x) = x + 1$, to define a function. Other times we use a computer program to specify a function.

A function $f : A \to B$ can also be defined in terms of a relation from A to B. Recall from Section 2.1 that a relation from A to B is

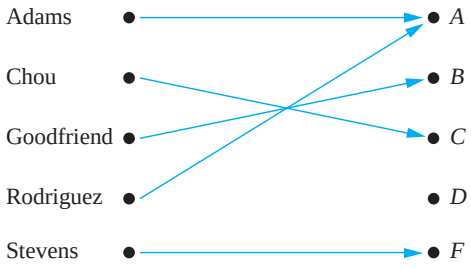


Figure 2.8: Assignment of Grades in a Discrete Mathematics Class.

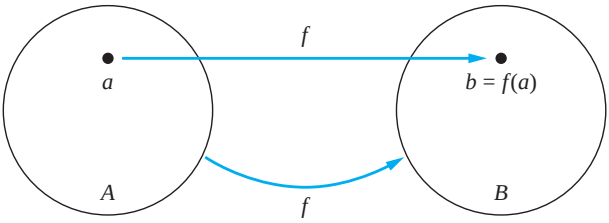


Figure 2.9: The Function f Maps A to B.

just a subset of $A \times B$. A relation from A to B that contains one, and only one, ordered pair (a, b) for every element $a \in A$, defines a function f from A to B. This function is defined by the assignment $f(a) = b$, where (a, b) is the unique ordered pair in the relation that has a as its first element.

Definition 2.3.2 *If f is a function from A to B, we say that A is the **domain** of f and B is the **codomain** of f. If $f(a) = b$, we say that b is the **image** of a and a is a **preimage** of b. The **range**, or image, of f is the set of all images of elements of A. Also, if f is a function from A to B, we say that f **maps** A to B.*

Figure 2.9 represents a function f from A to B.

When we define a function we specify its domain, its codomain, and the mapping of elements of the domain to elements in the codomain. Two functions are **equal** when they have the same domain, have the same codomain, and map each element of their common domain to the

same element in their common codomain. Note that if we change either the domain or the codomain of a function, then we obtain a different function. If we change the mapping of elements, then we also obtain a different function.

Examples 1–5 provide examples of functions. In each case, we describe the domain, the codomain, the range, and the assignment of values to elements of the domain.

 EXAMPLE. 1

What are the domain, codomain, and range of the function that assigns grades to students described in the first paragraph of the introduction of this section?

Solution: Let G be the function that assigns a grade to a student in our discrete mathematics class. Note that $G(\text{Adams}) = A$, for instance. The domain of G is the set $\{Adams, Chou, Goodfriend, Rodriguez, Stevens\}$, and the codomain is the set $\{A, B, C, D, F\}$. The range of G is the set $\{A, B, C, F\}$, because each grade except D is assigned to some student. ∎

 EXAMPLE. 2

Let R be the relation with ordered pairs (Abdul, 22), (Brenda, 24), (Carla, 21), (Desire, 22), (Eddie, 24), and (Felicia, 22). Here each pair consists of a graduate student and this student's age. Specify a function determined by this relation.

Solution: If f is a function specified by R, then $f(Abdul) = 22$, $f(Brenda) = 24$, $f(Carla) = 21$, $f(Desire) = 22$, $f(Eddie) = 24$, and $f(Felicia) = 22$. (Here, $f(x)$ is the age of x, where x is a student.) For the domain, we take the set $\{Abdul, Brenda, Carla, Desire, Eddie, Felicia\}$. We also need to specify a codomain, which needs to contain all possible ages of students. Because it is highly likely that all students are less than 100 years old, we can take the set of positive integers less than 100 as the codomain. (Note that we could choose a different codomain, such as the set of all positive integers or the set of positive integers between 10 and 90, but that would change the function. Using this codomain will also allow us to extend the function by adding the names and ages of more students later.) The range of the function we have specified is the set of different ages of these students, which is the set

$\{21, 22, 24\}$. ∎

EXAMPLE. 3

Let f be the function that assigns the last two bits of a bit string of length 2 or greater to that string. For example, $f(11010) = 10$. Then, the domain of f is the set of all bit strings of length 2 or greater, and both the codomain and range are the set $\{00,\ 01,\ 10,\ 11\}$. ∎

EXAMPLE. 4

Let $f : \mathbf{Z} \to \mathbf{Z}$ assign the square of an integer to this integer. Then, $f(x) = x^2$, where the domain of f is the set of all integers, the codomain of f is the set of all integers, and the range of f is the set of all integers that are perfect squares, namely, $\{0, 1, 4, 9, \ldots\}$. ∎

EXAMPLE. 5

The domain and codomain of functions are often specified in programming languages. For instance, the Java statement

$$int\ \mathbf{floor}(float\ real)\{\ldots\}$$

and the C++ function statement

$$int\ \mathbf{function}(float\ x)\{\ldots\}$$

both tell us that the domain of the floor function is the set of real numbers (represented by floating point numbers) and its codomain is the set of integers. ∎

A function is called **real-valued** if its codomain is the set of real numbers, and it is called **integer-valued** if its codomain is the set of integers. Two real-valued functions or two integervalued functions with the same domain can be added, as well as multiplied.

Definition 2.3.3 *Let f_1 and f_2 be functions from A to $\mathbf{R}$. Then $f_1 + f_2$ and $f_1 \cdot f_2$ are also functions from A to $\mathbf{R}$ defined for all $x \in A$ by*

$$(f_1 + f_2)(x) = f_1(x) + f_2(x),$$

$$(f_1 \cdot f_2)(x) = f_1(x) \cdot f_2(x).$$

Note that the functions $f_1 + f_2$ and $f_1 \cdot f_2$ have been defined by specifying their values at x in terms of the values of f_1 and f_2 at x.

EXAMPLE. 6

Let f_1 and f_2 be functions from $\mathbf{R}$ to $\mathbf{R}$ such that $f_1(x) = x^2$ and $f_2(x) = x - x^2$. What are the functions $f_1 + f_2$ and $f_1 \cdot f_2$?

Solution: From the definition of the sum and product of functions, it follows that

$$(f_1 + f_2)(x) = f_1(x) + f_2(x) = x^2 + (x - x^2) = x$$

and

$$(f_1 \cdot f_2)(x) = x^2(x - x^2) = x^3 - x^4. \quad \blacksquare$$

When f is a function from A to B, the image of a subset of A can also be defined.

Definition 2.3.4 *Let f be a function from A to B and let S be a subset of A. The **image** of S under the function f is the subset of B that consists of the images of the elements of S. We denote the image of S by $f(S)$, so*

$$f(S) = \{t | \exists s \in S(t = f(s))\}.$$

We also use the shorthand $\{f(s) | s \in S\}$ to denote this set.

Remark! The notation $f(S)$ for the image of the set S under the function f is potentially ambiguous. Here, $f(S)$ denotes a set, and not the value of the function f for the set S.

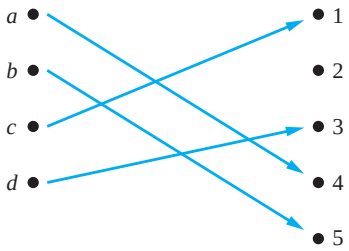


Figure 2.10: A One-to-One Function.

EXAMPLE. 7

Let $A = \{a, b, c, d, e\}$ and $B = \{1, 2, 3, 4\}$ with $f(a) = 2$, $f(b) = 1$, $f(c) = 4$, $f(d) = 1$, and $f(e) = 1$. The image of the subset $S = \{b, c, d\}$ is the set $f(S) = \{1, 4\}$. ∎

2.3.1 One-to-One and Onto Functions

Some functions never assign the same value to two different domain elements. These functions are said to be **one-to-one**.

Definition 2.3.5 *A function f is said to be **one-to-one**, or an injunction, if and only if $f(a) = f(b)$ implies that $a = b$ for all a and b in the domain of f. A function is said to be **injective** if it is one-to-one.*

Note that a function f is one-to-one if and only if $f(a) \neq f(b)$ whenever $a \neq b$. This way of expressing that f is one-to-one is obtained by taking the contrapositive of the implication in the definition.

> **Remark!** We can express that f is one-to-one using quantifiers as $\forall a \forall b(f(a) = f(b) \rightarrow a = b)$ or equivalently $\forall a \forall b(a \neq b \rightarrow f(a) \neq f(b))$, where the universe of discourse is the domain of the function.

We illustrate this concept by giving examples of functions that are one-to-one and other functions that are not one-to-one.

EXAMPLE. 8

Determine whether the function f from $\{a, b, c, d\}$ to $\{1, 2, 3, 4, 5\}$ with $f(a) = 4$, $f(b) = 5$, $f(c) = 1$, and $f(d) = 3$ is one-to-one.

Solution: The function f is one-to-one because f takes on different values at the four elements of its domain. This is illustrated in Figure 2.10. ∎

EXAMPLE. 9

Determine whether the function $f(x) = x^2$ from the set of integers to the set of integers is one-to-one.

Solution: The function $f(x) = x^2$ is not one-to-one because, for instance, $f(1) = f(-1) = 1$, but $1 \neq -1$.

Note that the function $f(x) = x^2$ with its domain restricted to $\mathbf{Z}^+$ is one-to-one. (Technically, when we restrict the domain of a function, we obtain a new function whose values agree with those of the original function for the elements of the restricted domain. The restricted function is not defined for elements of the original domain outside of the restricted domain.) ∎

EXAMPLE. 10

Determine whether the function $f(x) = x + 1$ from the set of real numbers to itself is one-to-one.

Solution: The function $f(x) = x + 1$ is a one-to-one function. To demonstrate this, note that $x + 1 \neq y + 1$ when $x \neq y$. ∎

EXAMPLE. 11

Suppose that each worker in a group of employees is assigned a job from a set of possible jobs, each to be done by a single worker. In this situation, the function f that assigns a job to each worker is one-to-one. To see this, note that if x and y are two different workers, then $f(x) \neq f(y)$ because the two

workers x and y must be assigned different jobs. ∎

We now give some conditions that guarantee that a function is one-to-one.

Definition 2.3.6 *A function f whose domain and codomain are subsets of the set of real numbers is called* ***increasing*** *if $f(x) \leq f(y)$, and strictly increasing if $f(x) < f(y)$, whenever $x < y$ and x and y are in the domain of f. Similarly, f is called* ***decreasing*** *if $f(x) \geq f(y)$, and strictly decreasing if $f(x) > f(y)$, whenever $x < y$ and x and y are in the domain of f. (The word* ***strictly*** *in this definition indicates a strict inequality.)*

Remark! A function f is increasing if $\forall x \forall y(x < y \rightarrow f(x) \leq f(y))$, strictly increasing if $\forall x \forall y(x < y \rightarrow f(x) < f(y))$, decreasing if $\forall x \forall y(x < y \rightarrow f(x) \geq f(y))$, and strictly decreasing if $\forall x \forall y(x < y \rightarrow f(x) > f(y))$, where the universe of discourse is the domain of f.

From these definitions, it can be shown that a function that is either strictly increasing or strictly decreasing must be one-to-one. However, a function that is increasing, but not strictly increasing, or decreasing, but not strictly decreasing, is not one-to-one.

For some functions the range and the codomain are equal. That is, every member of the codomain is the image of some element of the domain. Functions with this property are called **onto** functions.

Definition 2.3.7 *A function f from A to B is called* ***onto****, or a surjection, if and only if for every element $b \in B$ there is an element $a \in A$ with $f(a) = b$. A function f is called* ***surjective*** *if it is onto.*

Remark! A function f is onto if $\forall y \exists x(f(x) = y)$, where the domain for x is the domain of the function and the domain for y is the codomain of the function.

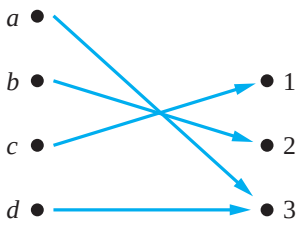


Figure 2.11: An Onto Function.

We now give examples of onto functions and functions that are not onto.

EXAMPLE. 12

Let f be the function from $\{a, b, c, d\}$ to $\{1, 2, 3\}$ defined by f (a) = 3, f (b) = 2, f (c) = 1, and $f(d) = 3$. Is f an onto function?

Solution: Because all three elements of the codomain are images of elements in the domain, we see that f is onto. This is illustrated in Figure 2.11. Note that if the codomain were $\{1, 2, 3, 4\}$, then f would not be onto. ∎

EXAMPLE. 13

Is the function $f(x) = x^2$ from the set of integers to the set of integers onto?

Solution: The function f is not onto because there is no integer x with $x^2 = -1$, for instance. ∎

EXAMPLE. 14

Is the function $f(x) = x + 1$ from the set of integers to the set of integers onto?

Solution: This function is onto, because for every integer y there is an integer x such that $f(x) = y$. To see this, note that $f(x) = y$ if and only if $x + 1 = y$, which holds if and only if $x = y - 1$. ∎

Consider the function f in Example 11 that assigns jobs to workers. The function f is onto if for every job there is a worker assigned this job. The function f is not onto when there is at least one job that has no worker assigned it.

Definition 2.3.8 *The function f is a* ***one-to-one correspondence****, or a* ***bijection****, if it is both one-to-one and onto. We also say that such a function is* ***bijective****.*

Examples 16 and 17 illustrate the concept of a bijection.

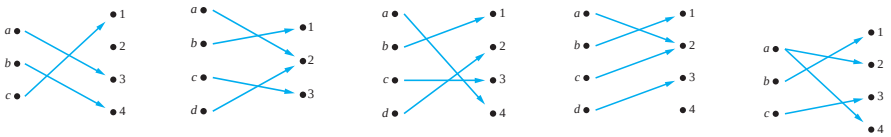


Figure 2.12: Examples of Different Types of Correspondences.

EXAMPLE. 16

Let f be the function from $\{a, b, c, d\}$ to $\{1, 2, 3, 4\}$ with $f(a) = 4$, $f(b) = 2$, $f(c) = 1$, and $f(d) = 3$. Is f a bijection?

Solution: The function f is one-to-one and onto. It is one-to-one because no two values in the domain are assigned the same function value. It is onto because all four elements of the codomain are images of elements in the domain. Hence, f is a bijection. ∎

Figure 2.12 displays four functions where the first is one-to-one but not onto, the second is onto but not one-to-one, the third is both one-to-one and onto, and the fourth is neither one-to-one nor onto. The fifth correspondence in Figure 2.12 is not a function, because it sends an element to two different elements.

Suppose that f is a function from a set A to itself. If A is finite, then f is one-to-one if and only if it is onto. This is not necessarily the case if A is infinite.

EXAMPLE. 17

Let A be a set. The identity function on A is the function $\iota_A : A \to A$, where $\iota_A(x) = x$ for all $x \in A$. In other words, the identity function ι_A is the function that assigns each element to itself. The function ι_A is one-to-one and onto, so it is a bijection. (Note that ι is the Greek letter iota.) ∎

For future reference, we summarize what needs be to shown to establish whether a function is one-to-one and whether it is onto. It is instructive to review Examples 8–17 in light of this summary.

Suppose that $f : A \to B$.

To show that f is injective Show that if $f(x) = f(y)$ for arbitrary $x, y \in A$ with $x \neq y$, then $x = y$.

To show that f is not injective Find particular elements $x, y \in A$ such that $x \neq y$ and $f(x) = f(y)$.

To show that f is surjective Consider an arbitrary element $y \in B$ and find an element $x \in A$ such that $f(x) = y$.

To show that f is not surjective Find a particular $y \in B$ such that $f(x) \neq y$ for all $x \in A$.

2.3.2 Inverse Functions and Compositions of Functions

Now consider a one-to-one correspondence f from the set A to the set B. Because f is an onto function, every element of B is the image of some element in A. Furthermore, because f is also a one-to-one function, every element of B is the image of a unique element of A. Consequently, we can define a new function from B to A that reverses the correspondence given by f.

Definition 2.3.9 *Let f be a one-to-one correspondence from the set A to the set B. The **inverse function** of f is the function that assigns to an element b belonging to B the unique element a in A such that $f(a) = b$. The inverse function of f is denoted by f^{-1}. Hence, $f^{-1}(b) = a$ when $f(a) = b$.*

Remark! Be sure not to confuse the function f^{-1} with the function $1/f$, which is the function that assigns to each x in the domain the value $1/f(x)$. Notice that the latter makes sense only when $f(x)$ is a non-zero real number.

Figure 2.13 illustrates the concept of an inverse function.

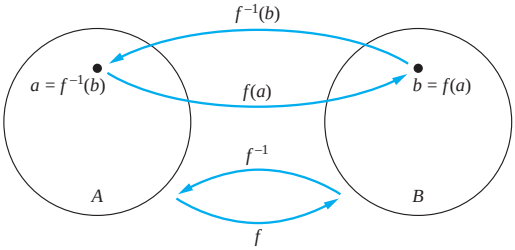


Figure 2.13: The Function f^{-1} Is the Inverse of Function f

When f is not a one-to-one correspondence, either it is not one-to-one or it is not onto. If f is not one-to-one, some element b in the codomain is the image of more than one element in the domain. If f is not onto, for some element b in the codomain, no element a in the domain exists for which $f(a) = b$. Consequently, if f is not a one-to-one correspondence, we cannot assign to each element b in the codomain a unique element a in the domain such that $f(a) = b$ (because for some b there is either more than one such a or no such a).

A one-to-one correspondence is called **invertible** because we can define an inverse of this function. A function is **not invertible** if it is not a one-to-one correspondence, because the inverse of such a function does not exist.

EXAMPLE. 18

Let f be the function from $\{a, b, c\}$ to $\{1, 2, 3\}$ such that $f(a) = 2$, $f(b) = 3$, and $f(c) = 1$. Is f invertible, and if it is, what is its inverse?

Solution: The function f is invertible because it is a one-to-one correspondence. The inverse function f^{-1} reverses the correspondence given by f

, so $f^{-1}(1) = c$, $f^{-1}(2) = a$, and $f^{-1}(3) = b$. ∎

EXAMPLE. 19

Let $f : \mathbf{Z} \to \mathbf{Z}$ be such that $f(x) = x + 1$. Is f invertible, and if it is, what is its inverse?

Solution: The function f has an inverse because it is a one-to-one correspondence, as follows from Examples 10 and 14. To reverse the correspondence, suppose that y is the image of x, so that $y = x + 1$. Then $x = y - 1$. This means that $y - 1$ is the unique element of $\mathbf{Z}$ that is sent to y by f. Consequently, $f^{-1}(y) = y - 1$. ∎

EXAMPLE. 20

Let f be the function from $\mathbf{R}$ to $\mathbf{R}$ with $f(x) = x^2$. Is f invertible?

Solution: Because $f(-2) = f(2) = 4$, f is not one-to-one. If an inverse function were defined, it would have to assign two elements to 4. Hence, f is not invertible. (Note we can also show that f is not invertible because it is not onto.) ∎

Sometimes we can restrict the domain or the codomain of a function, or both, to obtain an invertible function, as Example 21 illustrates.

EXAMPLE. 21

Show that if we restrict the function $f(x) = x^2$ in Example 20 to a function from the set of all nonnegative real numbers to the set of all nonnegative real numbers, then f is invertible.

Solution: The function $f(x) = x^2$ from the set of nonnegative real numbers to the set of nonnegative real numbers is one-to-one. To see this, note that if $f(x) = f(y)$, then $x^2 = y^2$, so $x^2 - y^2 = (x + y)(x - y) = 0$. This means that $x + y = 0$ or $x - y = 0$, so $x = -y$ or $x = y$. Because both x and y are nonnegative, we must have $x = y$. So, this function is one-to-one. Furthermore, $f(x) = x^2$ is onto when the codomain is the set of all nonnega-

tive real numbers, because each nonnegative real number has a square root. That is, if y is a nonnegative real number, there exists a nonnegative real number x such that $x = \sqrt{y}$, which means that $x^2 = y$. Because the function $f(x) = x^2$ from the set of nonnegative real numbers to the set of nonnegative real numbers is one-to-one and onto, it is invertible. Its inverse is given by the rule $f^{-1}(y) = \sqrt{y}$. ∎

Definition 2.3.10 *Let g be a function from the set A to the set B and let f be a function from the set B to the set C. The* ***composition*** *of the functions f and g, denoted for all $a \in A$ by $f \circ g$, is defined by*

$$(f \circ g)(a) = f(g(a)).$$

In other words, $f \circ g$ is the function that assigns to the element a of A the element assigned by f to $g(a)$. That is, to find $(f \circ g)(a)$ we first apply the function g to a to obtain $g(a)$ and then we apply the function f to the result $g(a)$ to obtain $(f \circ g)(a) = f(g(a))$. Note that the composition $f \circ g$ cannot be defined unless the range of g is a subset of the domain of f . In Figure 2.14 the composition of functions is shown.

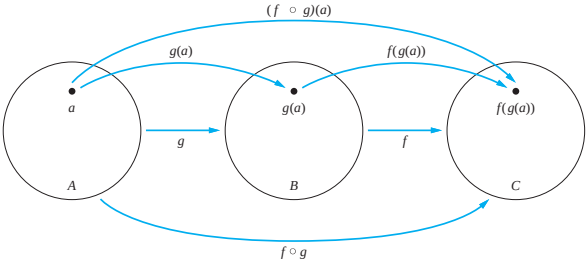


Figure 2.14: The Composition of the Functions f and g.

EXAMPLE. 22

Let g be the function from the set $\{a, b, c\}$ to itself such that $g(a) = b$, $g(b) = c$, and $g(c) = a$. Let f be the function from the set $\{a, b, c\}$ to the set $\{1, 2, 3\}$

such that $f(a) = 3$, $f(b) = 2$, and $f(c) = 1$. What is the composition of f and g, and what is the composition of g and f ?

Solution: The composition $f \circ g$ is defined by $(f \circ g)(a) = f(g(a)) = f(b) = 2$, $(f \circ g)(b) = f(g(b)) = f(c) = 1$, and $(f \circ g)(c) = f(g(c)) = f(a) = 3$.

Note that $g \circ f$ is not defined, because the range of f is not a subset of the domain of g. ∎

EXAMPLE. 23

Let f and g be the functions from the set of integers to the set of integers defined by $f(x) = 2x + 3$ and $g(x) = 3x + 2$. What is the composition of f and g? What is the composition of g and f ?

Solution: Both the compositions $f \circ g$ and $g \circ f$ are defined. Moreover,

$$(f \circ g)(x) = f(g(x)) = f(3x + 2) = 2(3x + 2) + 3 = 6x + 7$$

and

$$(g \circ f)(x) = g(f(x)) = g(2x + 3) = 3(2x + 3) + 2 = 6x + 11. \quad ∎$$

Remark! Note that even though $f \circ g$ and $g \circ f$ are defined for the functions f and g in Example 23, $f \circ g$ and $g \circ f$ are not equal. In other words, the commutative law does not hold for the composition of functions.

When the composition of a function and its inverse is formed, in either order, an identity function is obtained. To see this, suppose that f is a one-to-one correspondence from the set A to the set B. Then the inverse function f^{-1} exists and is a one-to-one correspondence from B to A. The inverse function reverses the correspondence of the original function, so $f^{-1}(b) = a$ when $f(a) = b$, and $f(a) = b$ when $f^{-1}(b) = a$. Hence,

$$(f^{-1} \circ f)(a) = f^{-1}(f(a)) = f^{-1}(b) = a,$$

and

$$(f \circ f^{-1})(b) = f(f^{-1}(b)) = f(a) = b.$$

Consequently ($f^{-1} \circ f = \iota_A$ and $f \circ f^{-1} = \iota_B$, where ι_A and ι_B are the identity functions on the sets A and B, respectively. That is, $(f^{-1})^{-1} = f$.

2.3.3 The Graphs of Functions

We can associate a set of pairs in $A \times B$ to each function from A to B. This set of pairs is called the **graph** of the function and is often displayed pictorially to aid in understanding the behavior of the function.

Definition 2.3.11 *Let f be a function from the set A to the set B. The **graph** of the function f is the set of ordered pairs $\{(a, b)|a \in A \ and f(a) = b\}$.*

From the definition, the graph of a function f from A to B is the subset of $A \times B$ containing the ordered pairs with the second entry equal to the element of B assigned by f to the first entry. Also, note that the graph of a function f from A to B is the same as the relation from A to B determined by the function f.

EXAMPLE. 24

Display the graph of the function $f(n) = 2n + 1$ from the set of integers to the set of integers.

Solution: The graph of f is the set of ordered pairs of the form $(n, 2n+1)$, where n is an integer. This graph is displayed in Figure 2.15. ∎

EXAMPLE. 25

Display the graph of the function $f(x) = x^2$ from the set of integers to the set of integers.

Solution: The graph of f is the set of ordered pairs of the form $(x, f(x)) = (x, x^2)$, where x is an integer. This graph is displayed in Figure 2.16. ∎

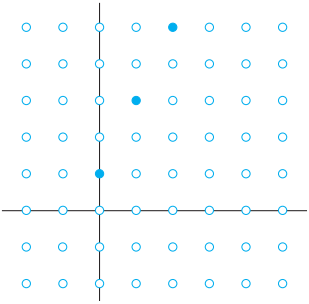


Figure 2.15: The Graph of $f(n) = 2n + 1$ from Z to Z.

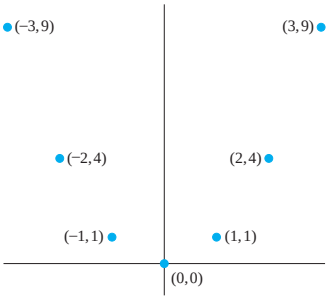


Figure 2.16: The Graph of $f(x) = x^2$ from Z to Z.

2.3.4 Some Important Functions

Next, weintroduce two important functions in discrete mathematics, namely, the floor and ceiling functions. Let x be a real number. The floor function rounds x down to the closest integer less than or equal to x, and the ceiling function rounds x up to the closest integer greater than or equal to x. These functions are often used when objects are counted. They play an important role in the analysis of the number of steps used by procedures to solve problems of a particular size.

Definition 2.3.12 *The **floor function** assigns to the real number x the largest integer that is less than or equal to x. The value of the floor function at x is denoted by $\lfloor x \rfloor$. The **ceiling function** assigns to the real number x the smallest integer that is greater than or equal to x. The value of the ceiling function at x is denoted by $\lceil x \rceil$.*

Remark! The floor function is often also called the greatest integer function. It is often denoted by $[x]$.

EXAMPLE. 26

These are some values of the floor and ceiling functions:

$$\lfloor \frac{1}{2} \rfloor = 1,\ \lceil \frac{1}{2} \rceil = 1,\ \lfloor -\frac{1}{2} \rfloor = -1,\ \lceil -\frac{1}{2} \rceil = 0,$$
$$\lfloor 3.1 \rfloor = 3, \lceil 3.1 \rceil = 4,\ \lfloor 7 \rfloor = 7,\ \lceil 7 \rceil = 7. \quad \blacksquare$$

We display the graphs of the floor and ceiling functions in Figure 2.17. In Figure 2.17a we display the graph of the floor function $\lfloor x \rfloor$. Note that this function has the same value throughout the interval $[n, n+1)$, namely n, and then it jumps up to $n+1$ when $x = n+1$. In Figure 2.17b we display the graph of the ceiling function $\lceil x \rceil$. Note that this function has the same value throughout the interval $(n, n+1]$, namely $n+1$, and then jumps to $n+2$ when x is a little larger than $n+1$.

The floor and ceiling functions are useful in a wide variety of applications, including those involving data storage and data transmission. Consider Examples 27 and 28, typical of basic calculations done when database and data communications problems are studied.

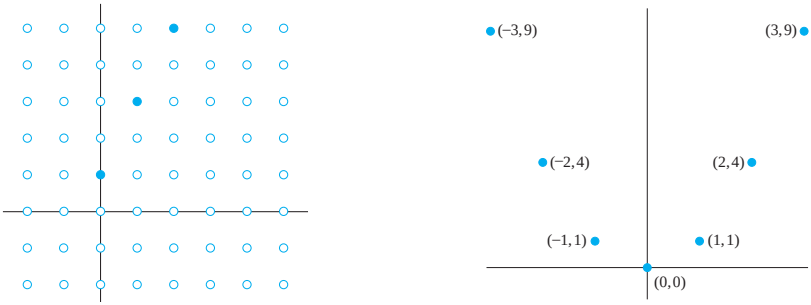
EXAMPLE. 27

Data stored on a computer disk or transmitted over a data network are usually represented as a string of bytes. Each byte is made up of 8 bits. How many bytes are required to encode 100 bits of data?

Solution: To determine the number of bytes needed, we determine the smallest integer that is at least as large as the quotient when 100 is divided by 8, the number of bits in a byte. Consequently, $\lceil 100/8 \rceil = \lceil 12.5 \rceil = 13$ bytes are required. $\blacksquare$

EXAMPLE. 28

In asynchronous transfer mode (ATM) (a communications protocol used on backbone networks), data are organized into cells of 53 bytes. How many ATM cells can be transmitted in 1 minute over a connection that transmits data at the rate of 500 kilobits per second?



a) $y = \lfloor x \rfloor$ b) $y = \lceil x \rceil$

Figure 2.17: Graphs of the (a) Floor and (b) Ceiling Functions.

Solution: In 1 minute, this connection can transmit $500,000 \cdot 60 = 30,000,000$ bits. EachATM cell is 53 bytes long, which means that it is $53 \cdot 8 = 424$ bits long. To determine the number of cells that can be transmitted in 1 minute, we determine the largest integer not exceeding the quotient when 30,000,000 is divided by 424. Consequently, $\lfloor 30,000,000/424 \rfloor = 70,754$ ATM cells can be transmitted in 1 minute over a 500 kilobit per second connection. ∎

Table 2.3, with x denoting a real number, displays some simple but important properties of the floor and ceiling functions. Because these functions appear so frequently in discrete mathematics, it is useful to look over these identities. Each property in this table can be established using the definitions of the floor and ceiling functions. Properties (1a), (1b), (1c), and (1d) follow directly from these definitions. For example, (1a) states that $\lfloor x \rfloor = n$ if and only if the integer n is less than or equal to x and $n + 1$ is larger than x. This is precisely what it means for n to be the greatest integer not exceeding x, which is the definition of $\lfloor x \rfloor = n$. Properties (1b), (1c), and (1d) can be established similarly.We will prove property (4a) using a direct proof.

Proof: Suppose that $\lfloor x \rfloor = m$, where m is a positive integer. By property (1a), it follows that $m \leq x < m + 1$. Adding n to all three quantities in this chain of two inequalities shows that $m + n \leq x + n < m + n + 1$. Using property (1a) again, we see that $\lfloor x + n \rfloor = m + n = \lfloor x \rfloor + n$. This completes the proof. Proofs of the other properties are left as exercises. ∎

Table 2.3: **Useful Properties of the Floor and Ceiling Functions.** (n is an integer, x is a real number)

(1a) $\lfloor x \rfloor = n$ if and only if $n \leq x < n+1$
(1b) $\lceil x \rceil = n$ if and only if $n-1 < x \leq n$
(1c) $\lfloor x \rfloor = n$ if and only if $x-1 < n \leq x$
(1d) $\lceil x \rceil = n$ if and only if $x \leq n < x+1$
(2) $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$
(3a) $\lfloor -x \rfloor = -\lceil x \rceil$
(3b) $\lceil -x \rceil = -\lfloor x \rfloor$
(4a) $\lfloor x + n \rfloor = \lfloor x \rfloor + n$
(4b) $\lceil x + n \rceil = \lceil x \rceil + n$

The floor and ceiling functions enjoy many other useful properties besides those displayed in Table 2.3. There are also many statements about these functions that may appear to be correct, but actually are not.We will consider statements about the floor and ceiling functions in Examples 29 and 30.

A useful approach for considering statements about the floor function is to let $x = n+\epsilon$, where $n = \lfloor x \rfloor$ is an integer, and ϵ, the fractional part of x, satisfies the inequality $0 \leq \epsilon < 1$. Similarly, when considering statements about the ceiling function, it is useful to write $x = n - \epsilon$, where $n = \lceil x \rceil$ is an integer and $0 \leq \epsilon < 1$.

EXAMPLE. 29

Prove that if x is a real number, then $\lfloor 2x \rfloor = \lfloor x \rfloor + \lfloor x + \frac{1}{2} \rfloor$.

Solution: To prove this statement we let $x = n + \epsilon$, where n is an integer and $0 \leq \epsilon < 1$. There are two cases to consider, depending on whether ϵ is less than, or greater than or equal to $\frac{1}{2}$. (The reason we choose these two cases will be made clear in the proof.)

We first consider the case when $0 \leq \epsilon < \frac{1}{2}$. In this case, $2x = 2n + 2\epsilon$ and $\lfloor 2x \rfloor = 2n$ because $0 \leq 2\epsilon < 1$. Similarly, $x + \frac{1}{2} = n + (\frac{1}{2} + \epsilon)$, so $\lfloor x + \frac{1}{2} \rfloor = n$, because $0 < \frac{1}{2} + \epsilon < 1$. Consequently, $\lfloor 2x \rfloor = 2n$ and $\lfloor x \rfloor + \lfloor x + \frac{1}{2} \rfloor = n + n = 2n$.

Next, we consider the case when $\frac{1}{2} \leq \epsilon < 1$. In this case, $2x = 2n + 2\epsilon = (2n + 1) + (2\epsilon - 1)$. Because $0 \leq 2\epsilon - 1 < 1$, it follows that $\lfloor 2x \rfloor = 2n + 1$. Because $\lfloor x + \frac{1}{2} \rfloor = \lfloor n + (\frac{1}{2} + \epsilon) \rfloor = \lfloor n + 1 + (\epsilon - \frac{1}{2}) \rfloor$ and $0 \leq \epsilon - \frac{1}{2} < 1$, it follows that $\lfloor x + \frac{1}{2} \rfloor = n + 1$. Consequently, $\lfloor 2x \rfloor = 2n + 1$ and $\lfloor x \rfloor + \lfloor x + \frac{1}{2} \rfloor = n + (n + 1) = 2n + 1$. This concludes the proof. ∎

EXAMPLE. 30

Prove or disprove that $\lceil x + y \rceil = \lceil x \rceil + \lceil y \rceil$ for all real numbers x and y.

Solution: Although this statement may appear reasonable, it is false. A counterexample is supplied by $x = \frac{1}{2}$ and $y = \frac{1}{2}$. With these values we find that $\lceil x + y \rceil = \lceil \frac{1}{2} + \frac{1}{2} \rceil = \lceil 1 \rceil = 1$, but $\lceil x \rceil + \lceil y \rceil = \lceil \frac{1}{2} \rceil + \lceil \frac{1}{2} \rceil = 1 + 1 = 2$ ∎.

There are certain types of functions that will be used throughout the text. These include polynomial, logarithmic, and exponential functions. In this book the notation $\log x$ will be used to denote the logarithm to the base 2 of x, because 2 is the base that we will usually use for logarithms. We will denote logarithms to the base b, where b is any real number greater than 1, by $log_b x$, and the natural logarithm by lnx.

Another function we will use throughout this text is the **factorial function** $f : \mathbf{N} \to \mathbf{Z}^+$, denoted by $f(n) = n!$. The value of $f(n) = n!$ is the product of the first n positive integers, so $f(n) = 1 \cdot 2 \ldots (n-1) \cdot n[$ and$f(0) = 0! = 1]$.

EXAMPLE. 31

We have $f(1) = 1! = 1$, $f(2) = 2! = 1 \cdot 2 = 2$, $f(6) = 6! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720$, and $f(20) = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 8 \cdot 9 \cdot 10 \cdot 11 \cdot 12 \cdot 13 \cdot 14 \cdot 15 \cdot 16 \cdot 17 \cdot 18 \cdot 19 \cdot 20 = 2,432,902,008,176,640,000$. ∎

Example 31 illustrates that the factorial function grows extremely rapidly as n grows. The rapid growth of the factorial function is made clearer by Stirling's formula, a result from higher mathematics that tell us that $n! \sim \sqrt{2\pi n}(n/e)^n$. Here, we have used the notation $f(n) \sim$

$g(n)$, which means that the ratio $f(n)/g(n)$ approaches 1 as n grows without bound (that is, $lim_{n\to\infty} f(n)/g(n) = 1$). The symbol $\sim$ is read "is asymptotic to". Stirling's formula is named after James Stirling, a Scottish mathematician of the eighteenth century.

2.3.5 Partial Functions

A program designed to evaluate a function may not produce the correct value of the function for all elements in the domain of this function. For example, a program may not produce a correct value because evaluating the function may lead to an infinite loop or an overflow. Similarly, in abstract mathematics, we often want to discuss functions that are defined only for a subset of the real numbers, such as $1/x$, $\sqrt{x}$, and $arcsin(x)$. Also, we may want to use such notions as the "youngest child" function, which is undefined for a couple having no children, or the "time of sunrise", which is undefined for some days above the Arctic Circle. To study such situations, we use the concept of a partial function.

Definition 2.3.13 *A **partial function** f from a set A to a set B is an assignment to each element a in a subset of A, called the **domain of definition** of f, of a unique element b in B. The sets A and B are called the **domain** and **codomain** of f, respectively. We say that f is **undefined** for elements in A that are not in the domain of definition of f. When the domain of definition of f equals A, we say that f is a **total function**.*

> **Remark!** We write $f : A \to B$ to denote that f is a partial function from A to B. Note that this is the same notation as is used for functions. The context in which the notation is used determines whether f is a partial function or a total function.

EXAMPLE. 32

The function $f : \mathbf{Z} \to \mathbf{R}$ where $f(n) = \sqrt{n}$ is a partial function from $\mathbf{Z}$ to $\mathbf{R}$

where the domain of definition is the set of nonnegative integers. Note that f is undefined for negative integers. ∎

2.4 Sequences and Summations

2.4.1 Introduction

Sequences are ordered lists of elements, used in discrete mathematics in many ways. We will often need to work with sums of terms of sequences in our study of discrete mathematics. This section reviews the use of summation notation, basic properties of summations, and formulas for the sums of terms of some particular types of sequences.

The terms of a sequence can be specified by providing a formula for each term of the sequence. In this section we describe another way to specify the terms of a sequence using a recurrence relation, which expresses each term as a combination of the previous terms. We will introduce one method, known as iteration, for finding a closed formula for the terms of a sequence specified via a recurrence relation. Identifying a sequence when the first few terms are provided is a useful skill when solving problems in discrete mathematics. We will provide some tips, including a useful tool on theWeb, for doing so.

2.4.2 Sequences

A sequence is a discrete structure used to represent an ordered list. For example, 1, 2, 3, 5, 8 is a sequence with five terms and 1, 3, 9, 27, 81, ..., 3^n, ... is an infinite sequence.

Definition 2.4.1 *A **sequence** is a function from a subset of the set of integers (usually either the set $\{0, 1, 2, \ldots\}$ or the set $\{1, 2, 3, \ldots\}$) to a set S. We use the notation a_n to denote the image of the integer n. We call a_n a **term** of the sequence.*

We use the notation $\{a_n\}$ to describe the sequence. (Note that an represents an individual term of the sequence $\{a_n\}$. Be aware that the notation $\{a_n\}$ for a sequence conflicts with the notation for a set.

However, the context in which we use this notation will always make it clear when we are dealing with sets and when we are dealing with sequences. Moreover, although we have used the letter a in the notation for a sequence, other letters or expressions may be used depending on the sequence under consideration. That is, the choice of the letter a is arbitrary.)

We describe sequences by listing the terms of the sequence in order of increasing subscripts.

EXAMPLE. 1

Consider the sequence $\{a_n\}$, where

$$a_n = \frac{1}{n}.$$

The list of the terms of this sequence, beginning with a_1, namely,

$$a_1,\ a_2,\ a_3,\ a_4, \ldots,$$

starts with

$$1,\ \frac{1}{2},\ \frac{1}{3},\ \frac{1}{4}, \ldots. \quad \blacksquare$$

Definition 2.4.2 *A geometric progression is a sequence of the form*

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

where the ***initial term*** *a and the* ***common ratio*** *r are real numbers.*

> **Remark!** A geometric progression is a discrete analogue of the exponential function $f(x) = ar^x$.

EXAMPLE. 2

The sequences $\{b_n\}$ with $b_n = (-1)^n$, $\{c_n\}$ with $c_n = 2 \cdot 5^n$, and $\{d_n\}$ with $d_n = 6 \cdot (1/3)^n$ are geometric progressions with initial term and common ratio

equal to 1 and -1; 2 and 5; and 6 and 1/3, respectively, if we start at $n = 0$. The list of terms $b_0,\ b_1,\ b_2,\ b_3,\ b_4, \ldots$ begins with

$$1, -1, 1, -1, 1, \ldots;$$

the list of terms $c_0,\ c_1,\ c_2,\ c_3,\ c_4, \ldots$ begins with

$$2,\ 10,\ 50,\ 250,\ 1250,\ \ldots;$$

and the list of terms $d_0,\ d_1,\ d_2,\ d_3,\ d_4,\ \ldots$ begins with

$$6,\ 2,\ \frac{2}{3},\ \frac{2}{9},\ \frac{2}{27},\ \ldots. \quad \blacksquare$$

Definition 2.4.3 *An **arithmetic progression** is a sequence of the form*

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

*where the **initial term** a and the **common difference** d are real numbers.*

> **Remark!** An arithmetic progression is a discrete analogue of the linear function $f(x) = dx + a$.

EXAMPLE. 3

The sequences $\{s_n\}$ with $s_n = -1 + 4n$ and $\{t_n\}$ with $t_n = 7 - 3n$ are both arithmetic progressions with initial terms and common differences equal to -1 and 4, and 7 and -3, respectively, if we start at $n = 0$. The list of terms $s_0, s_1, s_2, s_3, \ldots$ begins with

$$-1,\ 3,\ 7,\ 11, \ldots,$$

and the list of terms $t_0, t_1, t_2, t_3, \ldots$ begins with

$$7, 4, 1, -2, \ldots \quad \blacksquare$$

Sequences of the form $a_1, a_2, \ldots, a_n$ are often used in computer science. These finite sequences are also called **strings**. This string is also denoted by $a_1a_2 \ldots a_n$. The **length** of a string is the number of terms in this string. The **empty string**, denoted by λ, is the string that has no terms. The empty string has length zero.

EXAMPLE. 4

The string *abcd* is a string of length four. ∎

2.4.3 Recurrence Relations

In Examples 1–3 we specified sequences by providing explicit formulas for their terms. There are many other ways to specify a sequence. For example, another way to specify a sequence is to provide one or more initial terms together with a rule for determining subsequent terms from those that precede them.

Definition 2.4.4 *A **recurrence relation** for the sequence $\{a_n\}$ is an equation that expresses a_n in terms of one or more of the previous terms of the sequence, namely, a_0, a_1, $\ldots, a_{n-1}$, for all integers n with $n \geq n_0$, where n_0 is a nonnegative integer. A sequence is called a **solution** of a recurrence relation if its terms satisfy the recurrence relation.*

EXAMPLE. 5

Let $\{a_n\}$ be a sequence that satisfies the recurrence relation $a_n = a_{n-1} + 3$ for $n = 1, 2, 3, \ldots$, and suppose that $a_0 = 2$. What are a_1, a_2 and a_3?

Solution: We see from the recurrence relation that $a_1 = a_0 + 3 = 2 + 3 = 5$. It then follows that $a_2 = 5 + 3 = 8$ and $a_3 = 8 + 3 = 11$. ∎

EXAMPLE. 6

Let $\{a_n\}$ be a sequence that satisfies the recurrence relation $a_n = a_{n-1} - a_{n-2}$ for $n = 2, 3, 4, \ldots$, and suppose that $a_0 = 3$ and $a_1 = 5$. What are a_2 and a_3?

Solution: We see from the recurrence relation that $a_2 = a_1 - a_0 = 5 - 3 = 2$ and $a_3 = a_2 - a_1 = 2 - 5 = -3$. We can find a_4, a_5, and each successive term in a similar way. ∎

The **initial conditions** for a recursively defined sequence specify the terms that precede the first term where the recurrence relation takes effect. For instance, the initial condition in Example 5 is $a_0 = 2$, and the initial conditions in Example 6 are $a_0 = 3$ and $a_1 = 5$. Using mathematical induction, it can be shown that a recurrence relation together with its initial conditions determines a unique solution.

Next, we define a particularly useful sequence defined by a recurrence relation, known as the **Fibonacci sequence**, after the Italian mathematician Fibonacci who was born in the 12th century.

Definition 2.4.5 *The **Fibonacci sequence**, f_0, f_1, f_2, ..., is defined by the initial conditions $f_0 = 0$, $f_1 = 1$, and the recurrence relation*

$$f_n = f_{n-1} + f_{n-2}$$

for $n = 2, 3, 4, \ldots$.

EXAMPLE. 7

Find the Fibonacci numbers f_2, f_3, f_4, f_5, and f_6.

Solution: The recurrence relation for the Fibonacci sequence tells us that we find successive terms by adding the previous two terms. Because the initial conditions tell us that $f_0 = 0$ and $f_1 = 1$, using the recurrence relation in the definition we find that

$$\begin{aligned} f_2 &= f_1 + f_0 = 1 + 0 = 1, \\ f_3 &= f_2 + f_1 = 1 + 1 = 2, \\ f_4 &= f_3 + f_2 = 2 + 1 = 3, \\ f_5 &= f_4 + f_3 = 3 + 2 = 5, \\ f_6 &= f_5 + f_4 = 5 + 3 = 8. \end{aligned}$$

∎

EXAMPLE. 8

Suppose that $\{a_n\}$ is the sequence of integers defined by $a_n = n!$, the value of the factorial function at the integer n, where $n = 1, 2, 3, \ldots$. Because $n! = n((n-1)(n-2)\ldots 2 \cdot 1) = n(n-1)! = na_{n-1}$, we see that the sequence of factorials satisfies the recurrence relation $a_n = na_{n-1}$, together with the initial condition $a_1 = 1$. ∎

We say that we have solved the recurrence relation together with the initial conditions when we find an explicit formula, called a **closed formula**, for the terms of the sequence.

EXAMPLE. 9

Determine whether the sequence $\{a_n\}$, where $a_n = 3n$ for every nonnegative integer n, is a solution of the recurrence relation $a_n = 2a_{n-1} - a_{n-2}$ for $n = 2, 3, 4, \ldots$. Answer the same question where $a_n = 2^n$ and where $a_n = 5$.

Solution: Suppose that $a_n = 3n$ for every nonnegative integer n. Then, for $n \geq 2$, we see that $2a_{n-1} - a_{n-2} = 2(3(n-1)) - 3(n-2) = 3n = a_n$. Therefore, $\{a_n\}$, where $a_n = 3n$, is a solution of the recurrence relation.

Suppose that $a_n = 2^n$ for every nonnegative integer n. Note that $a_0 = 1$, $a_1 = 2$, and $a_2 = 4$. Because $2a_1 - a_0 = 2 \cdot 2 - 1 = 3 \neq a_2$, we see that $\{a_n\}$, where $a_n = 2^n$, is not a solution of the recurrence relation.

Suppose that $a_n = 5$ for every nonnegative integer n. Then for $n \geq 2$, we see that $a_n = 2a_{n-1} - a_{n-2} = 2 \cdot 5 - 5 = 5 = a_n$. Therefore, $\{a_n\}$, where $a_n = 5$, is a solution of the recurrence relation. ∎

Many methods have been developed for solving recurrence relations. Here, we will introduce a straightforward method known as iteration via several examples.

EXAMPLE. 10

Solve the recurrence relation and initial condition in Example 5.

Solution: We can successively apply the recurrence relation in Example

5, starting with the initial condition $a_1 = 2$, and working upward until we reach a_n to deduce a closed formula for the sequence. We see that

$$\begin{aligned} a_2 &= 2 + 3 \\ a_3 &= (2 + 3) + 3 = 2 + 3 \cdot 2 \\ a_4 &= (2 + 2 \cdot 3) + 3 = 2 + 3 \cdot 3 \\ &\vdots \\ a_n &= a_{n-1} + 3 = (2 + 3 \cdot (n - 2)) + 3 = 2 + 3(n - 1). \end{aligned}$$

We can also successively apply the recurrence relation in Example 5, starting with the term an and working downward until we reach the initial condition $a_1 = 2$ to deduce this same formula. The steps are

$$\begin{aligned} a_n &= a_{n-1} + 3 \\ &= (a_{n-2} + 3) + 3 = a_{n-2} + 3 \cdot 2 \\ &= (a_{n-3} + 3) + 3 \cdot 2 = a_{n-3} + 3 \cdot 3 \\ &\vdots \\ &= a_2 + 3(n - 2) = (a_1 + 3) + 3(n - 2) = 2 + 3(n - 1). \end{aligned}$$

At each iteration of the recurrence relation, we obtain the next term in the sequence by adding 3 to the previous term. We obtain the nth term after $n - 1$ iterations of the recurrence relation. Hence, we have added $3(n - 1)$ to the initial term $a_0 = 2$ to obtain a_n. This gives us the closed formula $a_n = 2 + 3(n - 1)$. Note that this sequence is an arithmetic progression. ∎

The technique used in Example 10 is called **iteration**. We have iterated, or repeatedly used, the recurrence relation. The first approach is called **forward substitution** – we found successive terms beginning with the initial condition and ending with a_n. The second approach is called **backward substitution**, because we began with a_n and iterated to express it in terms of falling terms of the sequence until we found it in terms of a_1. Note that when we use iteration, we essential guess a formula for the terms of the sequence.

EXAMPLE. 11

Compound Interest Suppose that a person deposits $10,000 in a savings account at a bank yielding 11% per year with interest compounded annually. How much will be in the account after 30 years?

Solution: To solve this problem, let P_n denote the amount in the account after n years. Because the amount in the account after n years equals the amount in the account after $n-1$ years plus interest for the nth year, we see that the sequence $\{P_n\}$ satisfies the recurrence relation

$$P_n = P_{n-1} + 0.11P_{n-1} = (1.11)P_{n-1}.$$

The initial condition is $P_0 = 10,000$.

We can use an iterative approach to find a formula for P_n. Note that

$$\begin{aligned} P_1 &= (1.11)P_0 \\ P_2 &= (1.11)P_1 = (1.11)^2 P_0 \\ P_3 &= (1.11)P_2 = (1.11)^3 P_0 \\ &\vdots \\ P_n &= (1.11)P_{n-1} = (1.11)^n P_0. \end{aligned}$$

When we insert the initial condition $P_0 = 10,000$, the formula $P_n = (1.11)^n 10,000$ is obtained. Inserting $n = 30$ into the formula $P_n = (1.11)^n 10,000$ shows that after 30 years the account contains

$$P_30 = (1.11)^{30} 10,000 = \$228,922.97. \quad \blacksquare$$

2.4.4 Special Integer Sequences

A common problem in discrete mathematics is finding a closed formula, a recurrence relation, or some other type of general rule for constructing the terms of a sequence. Sometimes only a few terms of a sequence solving a problem are known; the goal is to identify the sequence. Even though the initial terms of a sequence do not determine the entire sequence (after all, there are infinitely many different sequences that start with any finite set of initial terms), knowing the first few terms may help you make an educated conjecture about the identity of your sequence. Once you have made this conjecture, you can try to verify that you have the correct sequence.

When trying to deduce a possible formula, recurrence relation, or some other type of rule for the terms of a sequence when given the initial terms, try to find a pattern in these terms.You might also see

whether you can determine how a term might have been produced from those preceding it. There are many questions you could ask, but some of the more useful are:

- Are there runs of the same value? That is, does the same value occur many times in a row?
- Are terms obtained from previous terms by adding the same amount or an amount that depends on the position in the sequence?
- Are terms obtained from previous terms by multiplying by a particular amount?
- Are terms obtained by combining previous terms in a certain way?
- Are there cycles among the terms?

 EXAMPLE. 12

Find formulae for the sequences with the following first five terms: (a) 1, 1/2, 1/4, 1/8, 1/16; (b) 1, 3, 5, 7, 9; (c) 1, -1, 1, -1, 1.

Solution: (a) We recognize that the denominators are powers of 2. The sequence with $a_n = 1/2^n,\ n = 0,\ 1,\ 2,\ \ldots$ is a possible match. This proposed sequence is a geometric progression with $a = 1$ and $r = 1/2$.

(b) We note that each term is obtained by adding 2 to the previous term. The sequence with $a_n = 2n + 1,\ n = 0,\ 1,\ 2,\ \ldots$ is a possible match. This proposed sequence is an arithmetic progression with $a = 1$ and $d = 2$.

(c) The terms alternate between 1 and -1. The sequence with $a_n = (-1)^n,\ n = 0,\ 1,\ 2\ldots$ is a possible match. This proposed sequence is a geometric progression with $a = 1$ and $r = -1$. ∎

Examples 13–15 illustrate how we can analyze sequences to find how the terms are constructed.

 EXAMPLE. 13

How can we produce the terms of a sequence if the first 10 terms are 1, 2, 2, 3, 3, 3, 4, 4, 4, 4?

Solution: In this sequence, the integer 1 appears once, the integer 2 appears twice, the integer 3 appears three times, and the integer 4 appears four

times. A reasonable rule for generating this sequence is that the integer n appears exactly n times, so the next five terms of the sequence would all be 5, the following six terms would all be 6, and so on. The sequence generated this way is a possible match. ∎

EXAMPLE. 14

How can we produce the terms of a sequence if the first 10 terms are 5, 11, 17, 23, 29, 35, 41, 47, 53, 59?

Solution: Note that each of the first 10 terms of this sequence after the first is obtained by adding 6 to the previous term. (We could see this by noticing that the difference between consecutive terms is 6.) Consequently, the nth term could be produced by starting with 5 and adding 6 a total of $n-1$ times; that is, a reasonable guess is that the nth term is $5+6(n-1) = 6n-1$. (This is an arithmetic progression with $a = 5$ and $d = 6$.) ∎

EXAMPLE. 15

How can we produce the terms of a sequence if the first 10 terms are 1, 3, 4, 7, 11, 18, 29, 47, 76, 123?

Solution: Observe that each successive term of this sequence, starting with the third term, is the sum of the two previous terms. That is, $4 = 3+1$, $7 = 4+3$, $11 = 7+4$, and so on. Consequently, if Ln is the nth term of this sequence, we guess that the sequence is determined by the recurrence relation $L_n = L_{n-1} + L_{n-2}$ with initial conditions $L_1 = 1$ and $L_2 = 3$ (the same recurrence relation as the Fibonacci sequence, but with different initial conditions). This sequence is known as the **Lucas sequence**, after the French mathematician Francois Edouard Lucas. Lucas studied this sequence and the Fibonacci sequence in the nineteenth century. ∎

Another useful technique for finding a rule for generating the terms of a sequence is to compare the terms of a sequence of interest with the terms of a well-known integer sequence, such as terms of an arithmetic progression, terms of a geometric progression, perfect squares, perfect

Table 2.4: Some Useful Sequences.

nth Term	First 10 Terms
n^2	1,4,9,16,25,36,49,64,81,100, ...
n^3	1,8,27,64,125,216,343,512,729,1000, ...
n^4	1,16,81,256,625,1296,2401,4096,6561,10000, ...
2^n	2,4,8,16,32,64,128,256,512,1024, ...
3^n	3,9,27,81,243,729,2187,6561,19683,59049, ...
$n!$	1,2,6,24,120,720,5040,40320,362880,3628800, ...
f_n	1,1,2,3,5,8,13,21,34,55,89, ...

cubes, and so on. The first 10 terms of some sequences you may want to keep in mind are displayed in Table 2.4.

 EXAMPLE. 16

Conjecture a simple formula for an if the first 10 terms of the sequence $\{a_n\}$ are 1, 7, 25, 79, 241, 727, 2185, 6559, 19681, 59047.

Solution: To attack this problem, we begin by looking at the difference of consecutive terms, but we do not see a pattern. When we form the ratio of consecutive terms to see whether each term is a multiple of the previous term, we find that this ratio, although not a constant, is close to 3. So it is reasonable to suspect that the terms of this sequence are generated by a formula involving 3^n. Comparing these terms with the corresponding terms of the sequence $\{3^n\}$, we notice that the nth term is 2 less than the corresponding power of 3. We see that $a_n = 3^n - 2$ for $1 \leq n \leq 10$ and conjecture that this formula holds for all n. ∎

2.4.5 Summations

Next, we consider the addition of the terms of a sequence. For this we introduce **summation notation**. We begin by describing the notation used to express the sum of the terms

$$a_m,\ a_{m+1}, \ldots, a_n$$

from the sequence $\{a_n\}$. We use the notation

$$\sum_{j=m}^{n} a_j, \text{ or } \sum_{m \le j \le n} a_j$$

(read as the sum from $j = m$ to $j = n$ of a_j) to represent

$$a_m + a_{m+1} + \ldots + a_n.$$

Here, the variable j is called the **index of summation**, and the choice of the letter j as the variable is arbitrary; that is, we could have used any other letter, such as i or k. Or, in notation,

$$\sum_{j=m}^{n} a_j = \sum_{j=m}^{n} a_i = \sum_{k=m}^{n} a_k$$

Here, the index of summation runs through all integers starting with its **lower limitmand** ending with its **upper limit** n. A large uppercase Greek letter sigma,$\sum$, is used to denote summation.

The usual laws for arithmetic apply to summations. For example, when a and b are real numbers, we have $\sum_{j=1}^{n}(ax_j + by_j) = \sum_{y=1}^{n} x_j + \sum_{j=1}^{n} y_j)$ where $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_n$ are real numbers. We give some examples of summation notation.

EXAMPLE. 17

Use summation notation to express the sum of the first 100 terms of the sequence $\{a_j\}$, where $a_j = 1/j$ for $j = 1, 2, 3, \ldots$.

Solution: The lower limit for the index of summation is 1, and the upper limit is 100.We write this sum as

$$\sum_{j=1}^{100} \frac{1}{j} \quad \blacksquare$$

EXAMPLE. 18

What is the value of $\sum_{j=1}^{5} j^2$?

Solution: We have

$$\begin{aligned}\sum_{j=1}^{5} j^2 &= 1^2 + 2^2 + 3^2 + 4^2 + 5^2 \\ &= 1 + 4 + 9 = 16 + 25 \\ &= 55 \quad \blacksquare\end{aligned}$$

EXAMPLE. 19

What is the value of $\sum_{k=4}^{8} (-1)^k$?

Solution: We have

$$\begin{aligned}\sum_{k=4}^{8} (-1)^k &= (-1)^4 + (-1)^5 + (-1)^6 + (-1)^7 + (-1)^8 \\ &= 1 + (-1) + 1 + (-1) + 1 \\ &= 1 \quad \blacksquare\end{aligned}$$

Sometimes it is useful to shift the index of summation in a sum. This is often done when two sums need to be added but their indices of summation do not match. When shifting an index of summation, it is important to make the appropriate changes in the corresponding summand. This is illustrated by Example 20.

EXAMPLE. 20

Suppose we have the sum

$$\sum_{j=1}^{5} j^2$$

but want the index of summation to run between 0 and 4 rather than from 1 to 5. To do this, we let $k = j - 1$. Then the new summation index runs from 0 (because $k = 1 - 0 = 0$ when $j = 1$) to 4 (because $k = 5 - 1 = 4$ when $j = 5$), and the term j^2 becomes $(k + 1)^2$. Hence,

$$\sum_{j=1}^{5} j^2 = \sum_{k=0}^{4} (k + 1)^2$$

It is easily checked that both sums are $1 + 4 + 9 + 16 + 25 = 55$. ∎

Sums of terms of geometric progressions commonly arise (such sums are called **geometric series**). Theorem 2.4.1 gives us a formula for the sum of terms of a geometric progression.

THEOREM 2.4.1

If a and r are real numbers and $r \neq 0$, then

$$\sum_{j=0}^{n} ar^j = \begin{cases} \frac{ar^{n+1}-a}{r-1} & \text{if } r \neq 1 \\ (n+1)a & \text{if } r = 1 \end{cases}$$

Proof: Let

$$S_n = \sum_{j=0}^{n} ar^j$$

To compute S, first multiply both sides of the equality by r and then manipulate the resulting sum as follows:

rS_n	$=$	$r\sum_{j=0}^{n} ar^j$	substituting summation formula for$S$
	$=$	$\sum_{j=0}^{n} ar^{j+1}$	by the distributive property
	$=$	$\sum_{k=1}^{n+1} ar^k$	shifting the index of summation, with $k = j + 1$
	$=$	$(\sum_{k=0}^{n} ar^k) + (ar^{n+1} - a)$	removing $k = n + 1$ term and adding $k = 0$ term
	$=$	$S_n + (ar^{n+1} - a)$	substituting S for summation formula

From these equalities, we see that

$$rS_n = S_n + (ar^{n+1} - a).$$

Solving for S_n shows that if $r \neq 1$, then

$$S_n = \frac{ar^{n+1} - a}{r - 1}$$

If $r = 1$, then the $S_n = \sum_{j=0}^{n} ar^j = \sum_{j=0}^{n} a = (n+1)a$ ∎

EXAMPLE. 21

Double summations arise in many contexts (as in the analysis of nested loops in computer programs). An example of a double summation is

$$\sum_{i=1}^{4}\sum_{j=1}^{3} ij$$

To evaluate the double sum, first expand the inner summation and then continue by computing the outer summation:

$$\begin{aligned} \textstyle\sum_{i=1}^{4}\sum_{j=1}^{3} ij &= \textstyle\sum_{i=1}^{4}(i+2i+3i) \\ &= \textstyle\sum_{i=1}^{4} 6i \\ &= 6+12+18+24 = 60. \end{aligned}$$ ∎

We can also use summation notation to add all values of a function, or terms of an indexed set, where the index of summation runs over all values in a set. That is, we write

$$\sum_{x \in S} f(S)$$

to represent the sum of the values $f(s)$, for all members s of S.

EXAMPLE. 22

What is the value of $\sum_{x \in [0,2,4]} s$?

Solution: Because $\sum_{x \in [0,2,4]} s$ represents the sum of the values of s for all the members of the set $\{0, 2, 4\}$, it follows that

$$\sum_{x \in [0,2,4]} s = 0 + 2 + 4 = 6$$ ∎

Certain sums arise repeatedly throughout discrete mathematics. Having a collection of formulae for such sums can be useful; Table 2.5 provides a small table of formulae for commonly occurring sums.

Table 2.5: Some Useful Summation Formulae.

Sum	Closed Form
$\sum_{k=0}^{n} ar^k \ (r \neq 0)$	$\frac{ar^{n+1}-a}{r-1},\ r \neq 1$
$\sum_{k=1}^{n} k$	$\frac{n(n+1)}{2}$
$\sum_{k=1}^{n} k^2$	$\frac{n(n+1)(2n+1)}{6}$
$\sum_{k=1}^{n} k^3$	$\frac{n^2(n+1)^2}{4}$
$\sum_{k=0}^{\infty} x^k,\ \lvert x\rvert < 1$	$\frac{1}{1-x}$
$\sum_{k=1}^{\infty} kx^{k-1},\ \lvert x\rvert < 1$	$\frac{1}{(1-x)^2}$

We derived the first formula in this table in Theorem 2.4.1. The next three formulae give us the sum of the first n positive integers, the sum of their squares, and the sum of their cubes. These three formulae can be derived in many different ways. Also note that each of these formulae, once known, can easily be proved using mathematical induction. The last two formulae in the table involve infinite series and will be discussed shortly.

Example 23 illustrates how the formulae in Table 2.5 can be useful.

EXAMPLE. 23

Find $\sum_{k=50}^{100} k^2$.

Solution: First note that because $\sum_{k=1}^{100} k^2 = \sum_{k=1}^{49} k^2 + \sum_{k=50}^{100} k^2$, we have

$$\sum_{k=50}^{100} k^2 = \sum_{k=1}^{100} k^2 - \sum_{k=1}^{49} k^2$$

Using the formula $\sum_{k=1}^{n} k^2 = n(n+1)(2n+1)/6$ from Table 2.5 we see that

$$\sum_{k=50}^{100} k^2 = \frac{100 \cdot 101 \cdot 201}{6} - \frac{49 \cdot 50 \cdot 99}{6} = 338,350 - 40,425 = 297,925. \quad \blacksquare$$

SOME INFINITE SERIES

Although most of the summations in this book are finite sums, infinite series are important in some parts of discrete mathematics. Infinite series are usually studied in a course in calculus and even the definition of these series requires the use of calculus, but sometimes they arise in discrete mathematics, because discrete mathematics deals with infinite collections of discrete elements.

EXAMPLE. 24

(*Requires calculus*) Let x be a real number with $|x| < 1$. Find $\sum_{n=0}^{\infty} x^n$.

Solution: By Theorem 1 with $a = 1$ and $r = x$ we see that $\sum_{n=0}^{\infty} x^n = \frac{x^{k+1}-1}{x-1}$. Because $|x| < 1$, x^{k+1} approaches 0 as k approaches infinity. It follows that

$$\sum_{n=0}^{\infty} x^n = \lim_{k\to\infty} \frac{x^{k+1}-1}{x-1} = \frac{0-1}{x-1} = \frac{1}{1-x}. \quad \blacksquare$$

We can produce newsummation formulae by differentiating or integrating existing formulae.

EXAMPLE. 25

(Requires calculus) Differentiating both sides of the equation

$$\sum_{n=0}^{\infty} x^k = \frac{1}{1-x},$$

from Example 24 we find that

$$\sum_{k=1}^{\infty} kx^{k-1} = \frac{1}{(1-x)^2}.$$

(This differentiation is valid for $|x| < 1$ by a theorem about infinite series.) $\blacksquare$

2.5 Cardinality of Sets

2.5.1 Introduction

In Definition 2.1.4 of Section 2.1 we defined the cardinality of a finite set as the number of elements in the set. We use the cardinalities of finite sets to tell us when they have the same size, or when one is bigger than the other. In this section we extend this notion to infinite sets. That is, we will define what it means for two infinite sets to have the same cardinality, providing us with a way to measure the relative sizes of infinite sets.

We will be particularly interested in countably infinite sets, which are sets with the same cardinality as the set of positive integers. We will establish the surprising result that the set of rational numbers is countably infinite. We will also provide an example of an uncountable set when we show that the set of real numbers is not countable.

The concepts developed in this section have important applications to computer science. A function is called uncomputable if no computer program can be written to find all its values, even with unlimited time and memory.We will use the concepts in this section to explain why uncomputable functions exist.

We now define what it means for two sets to have the same size, or cardinality.

Definition 2.5.1 *The sets A and B have the same **cardinality** if and only if there is a one-to-one correspondence from A to B. When A and B have the same cardinality, we write $|A| = |B|$.*

For infinite sets the definition of cardinality provides a relative measure of the sizes of two sets, rather than a measure of the size of one particular set.We can also define what it means for one set to have a smaller cardinality than another set.

Definition 2.5.2 *If there is a one-to-one function from A to B, the cardinality of A is less than or the same as the cardinality of B and we write $|A| \leq |B|$. Moreover, when $|A| \leq |B|$ and A and B have different cardinality, we say that the cardinality of A is less than the cardinality of B and we write $|A| < |B|$.*

2.5.2 Countable Sets

We will now split infinite sets into two groups, those with the same cardinality as the set of natural numbers and those with a different cardinality.

Definition 2.5.3 *A set that is either finite or has the same cardinality as the set of positive integers is called* ***countable****. A set that is not countable is called* ***uncountable****. When an infinite set S is countable, we denote the cardinality of S by $\aleph_0$ (where $\aleph$ is aleph, the first letter of the Hebrew alphabet). We write $|S| = \aleph_0$ and say that S has cardinality "aleph null".*

We illustrate how to show a set is countable in the next example.

EXAMPLE. 1

Show that the set of odd positive integers is a countable set.

Solution: To show that the set of odd positive integers is countable, we will exhibit a one-to-one correspondence between this set and the set of positive integers. Consider the function

$$f(n) = 2n - 1$$

from $\mathbf{Z}^+$ to the set of odd positive integers. We show that f is a one-to-one correspondence by showing that it is both one-to-one and onto. To see that it is one-to-one, suppose that $f(n) = f(m)$. Then $2n - 1 = 2m - 1$, so $n = m$. To see that it is onto, suppose that t is an odd positive integer. Then t is 1 less than an even integer $2k$, where k is a natural number. Hence $t = 2k-1 = f(k)$. We display this one-to-one correspondence in Figure 2.18.

An infinite set is countable if and only if it is possible to list the elements of the set in a sequence (indexed by the positive integers). The reason for this is that a one-to-one correspondence f from the set of positive integers to a set S can be expressed in terms of a sequence $a_1,\ a_2,\ \ldots, a_n, \ldots$, where $a_1 = f(1),\ a_2 = f(2), \ldots, a_n = f(n), \ldots$

HILBERT'S GRAND HOTEL

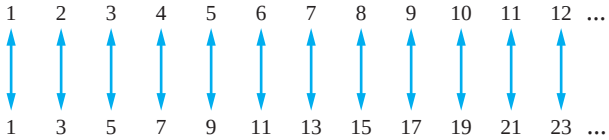


Figure 2.18: A One-to-One Correspondence Between $\mathbf{Z}^+$ and the Set of Odd Positive Integers.

We now describe a paradox that shows that something impossible with finite sets may be possible with infinite sets. The famous mathematician David Hilbert invented the notion of the **Grand Hotel**, which has a countably infinite number of rooms, each occupied by a guest. When a new guest arrives at a hotel with a finite number of rooms, and all rooms are occupied, this guest cannot be accommodated without evicting a current guest. However, we can always accommodate a new guest at the Grand Hotel, even when all rooms are already occupied, as we show in Example 2.

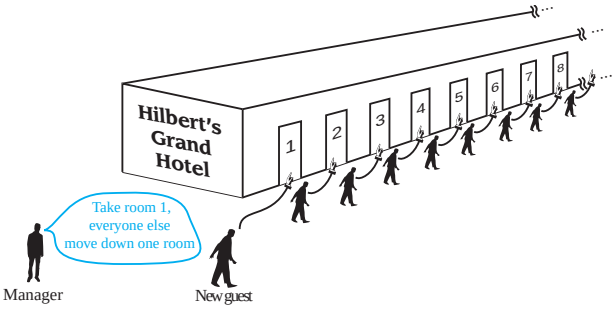


Figure 2.19: A New Guest Arrives at Hilbert's Grand Hotel.

EXAMPLE. 2

How can we accommodate a new guest arriving at the fully occupied Grand Hotel without removing any of the current guests?

Solution: Because the rooms of the Grand Hotel are countable, we can list them as Room 1, Room 2, Room 3, and so on. When a new guest arrives,

we move the guest in Room 1 to Room 2, the guest in Room 2 to Room 3, and in general, the guest in Room n to Room $n+1$, for all positive integers n. This frees up Room 1, which we assign to the new guest, and all the current guests still have rooms.We illustrate this situation in Figure 2.19.

When there are finitely many room in a hotel, the notion that all rooms are occupied is equivalent to the notion that no new guests can be accommodated. However, Hilbert's paradox of the Grand Hotel can be explained by noting that this equivalence no longer holds when there are infinitely many room.

EXAMPLES OF COUNTABLE AND UNCOUNTABLE SETS

We will now show that certain sets of numbers are countable.We begin with the set of all integers. Note that we can show that the set of all integers is countable by listing its members.

 EXAMPLE. 3

Show that the set of all integers is countable.

Solution: We can list all integers in a sequence by starting with 0 and alternating between positive and negative integers: $0, 1, -1, 2, -2, \ldots$ Alternatively, we could find a one-to-one correspondence between the set of positive integers and the set of all integers.We leave it to the reader to show that the function $f(n) = n/2$ when n is even and $f(n) = -(n-1)/2$ when n is odd is such a function. Consequently, the set of all integers is countable. ∎

It is not surprising that the set of odd integers and the set of all integers are both countable sets (as shown in Examples 1 and 3). Many people are amazed to learn that the set of rational numbers is countable, as Example 4 demonstrates.

 EXAMPLE. 4

Show that the set of positive rational numbers is countable.

Solution: It may seem surprising that the set of positive rational numbers is countable, but we will show how we can list the positive rational numbers as

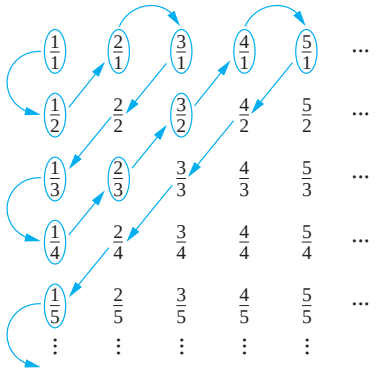


Figure 2.20: The Positive Rational Numbers Are Countable.

a sequence $r_1, r_2, \ldots, r_n, \ldots$. First, note that every positive rational number is the quotient p/q of two positive integers. We can arrange the positive rational numbers by listing those with denominator $q = 1$ in the first row, those with denominator $q = 2$ in the second row, and so on, as displayed in Figure 2.20.

The key to listing the rational numbers in a sequence is to first list the positive rational numbers p/q with $p+q = 2$, followed by those with $p+q = 3$, followed by those with $p+q = 4$, and so on, following the path shown in Figure 2.20. Whenever we encounter a number p/q that is already listed, we do not list it again. For example, when we come to $2/2 = 1$ we do not list it because we have already listed $1/1 = 1$. The initial terms in the list of positive rational numbers we have constructed are 1, 1/2, 2, 3, 1/3, 1/4, 2/3, 3/2, 4, 5, and so on. These numbers are shown circled; the uncircled numbers in the list are those we leave out because they are already listed. Because all positive rational numbers are listed once, as the reader can verify, we have shown that the set of positive rational numbers is countable. ∎

2.5.3 An Uncountable Set

We have seen that the set of positive rational numbers is a countable set. Do we have a promising candidate for an uncountable set? The first place we might look is the set of real numbers. In Example 5 we use an important proof method, introduced in 1879 by Georg Cantor

and known as the **Cantor diagonalization argument**, to prove that the set of real numbers is not countable. This proof method is used extensively in mathematical logic and in the theory of computation.

EXAMPLE. 5

Show that the set of real numbers is an uncountable set.

Solution: To show that the set of real numbers is uncountable, we suppose that the set of real numbers is countable and arrive at a contradiction. Then, the subset of all real numbers that fall between 0 and 1 would also be countable. Under this assumption, the real numbers between 0 and 1 can be listed in some order, say, $r_1,\ r_2,\ r_3,\ \ldots$. Let the decimal representation of these real numbers be

$$\begin{aligned} r_1 &= 0.d_{11}d_{12}d_{13}d_{14}\ldots \\ r_2 &= 0.d_{21}d_{22}d_{23}d_{24}\ldots \\ r_3 &= 0.d_{31}d_{32}d_{33}d_{34}\ldots \\ r_4 &= 0.d_{41}d_{42}d_{43}d_{44}\ldots \\ &\vdots \end{aligned}$$

where $d_{ij} \in \{0,1,2,3,4,5,6,7,8,9\}$. (For example, if $r_1 = 0.23794102\ldots$, we have $d_{11} = 2,\ d_{12} = 3,\ d_{13} = 7$, and so on.) Then, form a new real number with decimal expansion $r = 0.d_1d_2d_3d_4\ldots$, where the decimal digits are determined by the following rule:

$$d_i = \begin{cases} 4 & \text{if } d_{ii} \neq 4 \\ 5 & \text{if } d_{ii} \neq 4. \end{cases}$$

(As an example, suppose that $r_1 = 0.23794102\ldots,\ r_2 = 0.44590138\ldots,\ r_3 = 0.09118764\ldots,\ r_4 = 0.80553900\ldots$, and so on. Then we have $r = 0.d_1d_2d_3d_4\ldots = 0.4544\ldots$, where $d_1 = 4$ because $d_{11} \neq 4,\ d_2 = 5$ because $d_{22} = 4,\ d_3 = 4$ because $d_{33} \neq 4,\ d_4 = 4$ because $d_{44} \neq 4$, and so on.)

Every real number has a unique decimal expansion (when the possibility that the expansion has a tail end that consists entirely of the digit 9 is excluded). Therefore, the real number r is not equal to any of $r_1, r_2, \ldots$ because the decimal expansion of r differs from the decimal expansion of r_i in the ith place to the right of the decimal point, for each i.

Because there is a real number r between 0 and 1 that is not in the list, the assumption that all the real numbers between 0 and 1 could be listed must be false. Therefore, all the real numbers between 0 and 1 cannot be listed, so the

set of real numbers between 0 and 1 is uncountable.Any set with an uncountable subset is uncountable. Hence, the set of real numbers is uncountable. ∎

RESULTS ABOUT CARDINALITY We will now discuss some results about the cardinality of sets. First, we will prove that the union of two countable sets is also countable.

THEOREM 2.5.1

If A and B are countable sets, then $A \cup B$ is also countable.

Proof: Suppose that A and B are both countable sets. Without loss of generality, we can assume thatAandB are disjoint. (If they are not, we can replace B by $B-A$, because $A \cap (B-A) = \emptyset$ and $A \cup (B-A) = A \cup B$.) Furthermore, without loss of generality, if one of the two sets is countably infinite and other finite, we can assume that B is the one that is finite.

There are three cases to consider: (i) A and B are both finite, (ii) A is infinite and B is finite, and (iii) A and B are both countably infinite.

Case (i): Note that when A and B are finite, $A \cup B$ is also finite, and therefore, countable.

Case (ii): Because A is countably infinite, its elements can be listed in an infinite sequence $a_1, a_2, a_3, \ldots, a_n, \ldots$ and because B is finite, its terms can be listed as $b_1, b_2, \ldots, b_m$ for some positive integer m. We can list the elements of $A \cup B$ as $b_1, b_2, \ldots, b_m, a_1, a-2, a_3, \ldots, a_n, \ldots$. This means that $A \cup B$ is countably infinite.

Case (iii): Because both A and B are countably infinite, we can list their elements as $a_1, a_2, a_3, \ldots, a_n, \ldots$ and$b_1, b_2, \ldots, b_n, \ldots$, respectively. By alternating terms of these two sequences we can list the elements of $A \cup B$ in the infinite sequence $a_1, b_1, a_2, b_2, a_3, b_3, \ldots, a_n, b_n, \ldots$. This means $A \cup B$ must be countably infinite.

We have completed the proof, as we have shown that $A \cup B$ is countable in all three cases. ∎

THEOREM 2.5.2: SCHRÖDER-BERNSTEIN

If A and B are sets with $|A| \leq |B|$ and $|B| \leq |A|$, then $|A| = |B|$.

In other words, if there are one-to-one functions f from A to B and g from B to A, then there is a one-to-one correspondence between A and B.

Because Theorem 2.5.2 seems to be quite straightforward, we might expect that it has an easy proof. However, even though it can be proved without using advanced mathematics, no known proof is easy to explain. Consequently, we omit a proof here.

We illustrate the use of 2.5.2 with an example.

EXAMPLE. 6

Show that the $|(0,1)| = |(0,1]|$.

Solution: It is not at all obvious how to find a one-to-one correspondence between $(0,1)$ and $(0,1]$ to show that $|(0,1)| = |(0,1]|$. Fortunately, we can use the Schröder-Bernstein theorem instead. Finding a one-to-one function from $(0,1)$ to $(0,1]$ is simple. Because $(0,1) \subset (0,1]$, $f(x) = x$ is a one-to-one function from $(0,1)$ to $(0,1]$. Finding a one-to-one function from $(0,1]$ to $(0,1)$ is also not difficult. The function $g(x) = x/2$ is clearly one-to-one and maps $(0,1]$ to $(0,1/2] \subset (0,1)$. As we have found one-to-one functions from $(0,1)$ to $(0,1]$ and from $(0,1]$ to $(0,1)$, the Schröder-Bernstein theorem tells us that $|(0,1)| = |(0,1]|$. ∎

UNCOMPUTABLE FUNCTIONS

We will now describe an important application of the concepts of this section to computer science. In particular, we will show that there are functions whose values cannot be computed by any computer program.

Definition 2.5.4 *We say that a function is* ***computable*** *if there is a computer program in some programming language that finds the values of this function. If a function is not computable we say it is* ***uncomputable****.*

To show that there are uncomputable functions, we need to establish two results. First, we need to show that the set of all computer programs in any particular programming language is countable. This can be

proved by noting that a computer programs in a particular language can be thought of as a string of characters from a finite alphabet. Next, we show that there are uncountably many different functions from a particular countably infinite set to itself.

THE CONTINUUM HYPOTHESIS

We conclude this section with a brief discussion of a famous open question about cardinality. It can be shown that the power set of $\mathbf{Z}^+$ and the set of real numbers $\mathbf{R}$ have the same cardinality. In other words, we know that $|\mathcal{P}(\mathbf{Z}^+)| = |\mathbf{R}| = c$, where c denotes the cardinality of the set of real numbers.

An important theorem of Cantor states that the cardinality of a set is always less than the cardinality of its power set. Hence, $|\mathbf{Z}^+| < |\mathcal{P}(\mathbf{Z}^+)|$. We can rewrite this as $\aleph_0 < 2^{\aleph_0}$, using the notation $2|S|$ to denote the cardinality of the power set of the set S. Also, note that the relationship $|\mathcal{P}(\mathbf{Z}^+)| = |\mathbf{R}|$ can be expressed as $2^{\aleph_0} = c$.

This leads us to the famous **continuum hypothesis**, which asserts that there is no cardinal number X between $\aleph_0$ and c. In other words, the continuum hypothesis states that there is no set A such that $\aleph_0$, the cardinality of the set of positive integers, is less than $|A|$ and $|A|$ is less than c, the cardinality of the set of real numbers. It can be shown that the smallest infinite cardinal numbers form an infinite sequence $\aleph_0 < \aleph_1 < \aleph_2 < \ldots$. If we assume that the continuum hypothesis is true, it would follow that $c = \aleph_1$, so that $2^{\aleph_0} = \aleph_1$.

The continuum hypothesiswas stated by Cantor in 1877. He labored unsuccessfully to prove it, becoming extremely dismayed that he could not. By 1900, settling the continuum hypothesis was considered to be among the most important unsolved problems in mathematics. It was the first problem posed by David Hilbert in his famous 1900 list of open problems in mathematics.

The continuum hypothesis is still an open question and remains an area for active research. However, it has been shown that it can be neither proved nor disproved under the standard set theory axioms in modern mathematics, the Zermelo-Fraenkel axioms. The Zermelo-Fraenkel axioms were formulated to avoid the paradoxes of naive set theory, such as Russell's paradox, but there is much controversy whether they should be replaced by some other set of axioms for set theory.

Chapter 3

Algorithms

Many problems can be solved by considering them as special cases of general problems. For instance, consider the problem of locating the largest integer in the sequence 101, 12, 144, 212, 98. This is a specific case of the problem of locating the largest integer in a sequence of integers. To solve this general problem we must give an algorithm, which specifies a sequence of steps used to solve this general problem.We will study algorithms for solving many different types of problems in this book. For example, in this chapter we will introduce algorithms for two of the most important problems in computer science, searching for an element in a list and sorting a list so its elements are in some prescribed order, such as increasing, decreasing, or alphabetic.

We will also introduce the notion of an algorithmic paradigm, which provides a general method for designing algorithms. We will also discuss greedy algorithms, a class of algorithms used to solve optimization problems. Proofs are important in the study of algorithms. In this chapter we illustrate this by proving that a particular greedy algorithm always finds an optimal solution.

One important consideration concerning an algorithm is its computational complexity, which measures the processing time and computer memory required by the algorithm to solve problems of a particular size. To measure the complexity of algorithms we use big-O and big-*Theta* notation, which we develop in this chapter.We will illustrate the analysis of the complexity of algorithms in this chapter, focusing on

the time an algorithm takes to solve a problem. Furthermore, we will discuss what the time complexity of an algorithm means in practical and theoretical terms.

3.1 Algorithms

3.1.1 Introduction

There are many general classes of problems that arise in discrete mathematics. For instance: given a sequence of integers, find the largest one; given a set, list all its subsets; given a set of integers, put them in increasing order; given a network, find the shortest path between two vertices. When presented with such a problem, the first thing to do is to construct a model that translates the problem into a mathematical context. Discrete structures used in such models include sets, sequences, and functions—structures discussed in Chapter 2—as well as such other structures as permutations, relations, graphs, trees, networks, and finite state machines— concepts that will be discussed in later chapters.

Setting up the appropriate mathematical model is only part of the solution. To complete the solution, a method is needed that will solve the general problem using the model. Ideally, what is required is a procedure that follows a sequence of steps that leads to the desired answer. Such a sequence of steps is called an **algorithm**.

Definition 3.1.1 *An **algorithm** is a finite sequence of precise instructions for performing a computation or for solving a problem.*

The term *algorithm* is a corruption of the name *al-Khowarizmi*, a mathematician of the ninth century, whose book on Hindu numerals is the basis of modern decimal notation. Originally, the word algorism was used for the rules for performing arithmetic using decimal notation. *Algorism* evolved into the word *algorithm* by the eighteenth century. With the growing interest in computing machines, the concept of an algorithm was given a more general meaning, to include all definite procedures for solving problems, not just the procedures for performing arithmetic.

We will discuss algorithms that solve a wide variety of problems. In this section we will use the problem of finding the largest integer in a finite sequence of integers to illustrate the concept of an algorithm and the properties algorithms have. Also, we will describe algorithms for locating a particular element in a finite set. In subsequent sections, procedures for finding the greatest common divisor of two integers, for finding the shortest path between two points in a network, for multiplying matrices, and so on, will be discussed.

EXAMPLE. 1

Describe an algorithm for finding the maximum (largest) value in a finite sequence of integers.

Even though the problemof finding the maximum element in a sequence is relatively trivial, it provides a good illustration of the concept of an algorithm. Also, there are many instances where the largest integer in a finite sequence of integers is required. For instance, a university may need to find the highest score on a competitive exam taken by thousands of students. Or a sports organization may want to identify the member with the highest rating each month. We want to develop an algorithm that can be used whenever the problem of finding the largest element in a finite sequence of integers arises.

We can specify a procedure for solving this problem in several ways. One method is simply to use the English language to describe the sequence of steps used. We now provide such a solution.

Solution: We perform the following steps.

1. Set the temporary maximum equal to the first integer in the sequence. (The temporary maximum will be the largest integer examined at any stage of the procedure.)

2. Compare the next integer in the sequence to the temporary maximum, and if it is larger than the temporary maximum, set the temporary maximum equal to this integer.

3. Repeat the previous step if there are more integers in the sequence.

4. Stop when there are no integers left in the sequence. The temporary maximum at this point is the largest integer in the sequence. ■

An algorithm can also be described using a computer language. However, when that is done, only those instructions permitted in the language can be used. This often leads to a description of the algo-

rithm that is complicated and difficult to understand. Furthermore, because many programming languages are in common use, it would be undesirable to choose one particular language. So, instead of using a particular computer language to specify algorithms, a form of **pseudocode**. (We will also describe algorithms using the English language.) Pseudocode provides an intermediate step between an English language description of an algorithm and an implementation of this algorithm in a programming language. The steps of the algorithm are specified using instructions resembling those used in programming languages. However, in pseudocode, the instructions used can include any well-defined operations or statements. A computer program can be produced in any computer language using the pseudocode description as a starting point.

ALGORITHM 1
Finding the Maximum Element in a Finite Sequence.

procedure $max(a_1,\ a_2, \ldots, a_n$:integers)
$max := a_1$
  for $i := 2$ to n
    if max $< a_i$ then max $:= a_i$
return max\{ max is the largest element\}

The pseudocode used in this book is designed to be easily understood. It can serve as an intermediate step in the construction of programs implementing algorithms in one of a variety of different programming languages. Although this pseudocode does not follow the syntax of Java, C, C++, or any other programming language, students familiar with a modern programming language will find it easy to follow. A key difference between this pseudocode and code in a programming language is that we can use any well-defined instruction even if it would take many lines of code to implement this instruction. The reader should refer to this appendix whenever the need arises.

A pseudocode description of the algorithm for finding the maximum element in a finite sequence follows.

This algorithm first assigns the initial term of the sequence, a_1, to

the variable *max*. The "for" loop is used to successively examine terms of the sequence. If a term is greater than the current value of max, it is assigned to be the new value of max. The algorithm terminates after all terms have been examined. The value of max on termination is the maximum element in the sequence.

To gain insight into how an algorithm works it is useful to construct a trace that shows its steps when given specific input. For instance, a trace of Algorithm 1 with input 8, 4, 11, 3, 10 begins with the algorithm setting *max* to 8, the value of the initial term. It then compares 4, the second term, with 8, the current value of *max*. Because $4 \leq 8$, *max* is unchanged. Next, the algorithm compares the third term, 11, with 8, the current value of *max*. Because $8 < 11$, *max* is set equal to 11. The algorithm then compares 3, the fourth term, and 11, the current value of *max*. Because $3 \leq 11$, *max* is unchanged. Finally, the algorithm compares 10, the first term, and 11, the current value of *max*. As $10 \leq 11$, *max* remains unchanged. Because there are five terms, we have $n = 5$. So after examining 10, the last term, the algorithm terminates, with $max = 11$. When it terminates, the algorithms reports that 11 is the largest term in the sequence.

PROPERTIES OF ALGORITHMS

There are several properties that algorithms generally share. They are useful to keep in mind when algorithms are described. These properties are:

- *Input.* An algorithm has input values from a specified set.
- *Output.* From each set of input values an algorithm produces output values from a specified set. The output values are the solution to the problem.
- *Definiteness.* The steps of an algorithm must be defined precisely.
- *Correctness.* An algorithm should produce the correct output values for each set of input values.
- *Finiteness.* An algorithm should produce the desired output after a finite (but perhaps large) number of steps for any input in the set.
- *Effectiveness.* It must be possible to perform each step of an algorithm exactly and in a finite amount of time.

- *Generality.* The procedure should be applicable for all problems of the desired form, not just for a particular set of input values.

EXAMPLE. 2

Show that Algorithm 1 for finding the maximum element in a finite sequence of integers has all the properties listed.

Solution: The input to Algorithm 1 is a sequence of integers. The output is the largest integer in the sequence. Each step of the algorithm is precisely defined, because only assignments, a finite loop, and conditional statements occur. To show that the algorithm is correct, we must show that when the algorithm terminates, the value of the variable max equals the maximum of the terms of the sequence. To see this, note that the initial value of max is the first term of the sequence; as successive terms of the sequence are examined, max is updated to the value of a term if the term exceeds the maximum of the terms previously examined. This argument shows that when all the terms have been examined, max equals the value of the largest term. The algorithm uses a finite number of steps, because it terminates after all the integers in the sequence have been examined. The algorithm can be carried out in a finite amount of time because each step is either a comparison or an assignment, there are a finite number of these steps, and each of these two operations takes a finite amount of time. Finally, Algorithm 1 is general, because it can be used to find the maximum of any finite sequence of integers. ∎

3.1.2 Searching Algorithms

The problem of locating an element in an ordered list occurs in many contexts. For instance, a program that checks the spelling of words searches for them in a dictionary, which is just an ordered list of words. Problems of this kind are called **searching problems**. We will discuss several algorithms for searching in this section.We will study the number of steps used by each of these algorithms in Section 3.3.

The general searching problem can be described as follows: Locate an element x in a list of distinct elements $a_1, a_2, \ldots, a_n$, or determine that it is not in the list. The solution to this search problem is the

location of the term in the list that equals x (that is, i is the solution if $x = a_i$) and is 0 if x is not in the list.

THE LINEAR SEARCH

The first algorithm that we will present is called the **linear search or sequential search**, algorithm. The linear search algorithm begins by comparing x and a_1. When $x = a_1$, the solution is the location of a_1, namely, 1.

ALGORITHM 2
The Linear Search Algorithm.

procedure *linear search*(x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
if $i \leq n$ **then** *location* $:= i$
else *location*$:= 0$
return *location*{*location* is the subscript of the term that equals x, or is 0 if x is not found}

When $x \neq a_1$, compare x with a_2. If $x = a_2$, the solution is the location of a_2, namely, 2. When $x \neq a_2$, compare x with a_3. Continue this process, comparing x successively with each term of the list until a match is found, where the solution is the location of that term, unless no match occurs. If the entire list has been searched without locating x, the solution is 0. The pseudocode for the linear search algorithm is displayed as Algorithm 2.

THE BINARY SEARCH

We will now consider another searching algorithm. This algorithm can be used when the list has terms occurring in order of increasing size (for instance: if the terms are numbers, they are listed from smallest to largest; if they are words, they are listed in lexicographic, or alphabetic, order). This second searching algorithm is called the **binary search**

algorithm. It proceeds by comparing the element to be located to the middle term of the list. The list is then split into two smaller sublists of the same size, or where one of these smaller lists has one fewer term than the other. The search continues by restricting the search to the appropriate sublist based on the comparison of the element to be located and the middle term. In Section 3.3, it will be shown that the binary search algorithm is much more efficient than the linear search algorithm. Example 3 demonstrates how a binary search works.

EXAMPLE. 3

To search for 19 in the list

1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22,

first split this list, which has 16 terms, into two smaller lists with eight terms each, namely,

1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22.

Then, compare 19 and the largest term in the first list. Because $10 < 19$, the search for 19 can be restricted to the list containing the 9th through the 16th terms of the original list. Next, split this list, which has eight terms, into the two smaller lists of four terms each, namely,

12 13 15 16 18 19 20 22.

Because $16 < 19$ (comparing 19 with the largest term of the first list) the search is restricted to the second of these lists, which contains the 13th through the 16th terms of the original list. The list 18 19 20 22 is split into two lists, namely,

18 19 20 22.

Because 19 is not greater than the largest term of the first of these two lists, which is also 19, the search is restricted to the first list: 18 19, which contains the 13th and 14th terms of the original list. Next, this list of two terms is split into two lists of one term each: 18 and 19. Because $18 < 19$, the search is restricted to the second list: the list containing the 14th term of the list, which is 19. Now that the search has been narrowed down to one term, a comparison is made, and 19 is located as the 14th term in the original list. ∎

We now specify the steps of the binary search algorithm. To search for the integer x in the list $a_1, a_2, \ldots, a_n$, where $a_1 < a_2 < \ldots < a_n$, begin by comparing x with the middle term a_m of the list, where $m = \lfloor (n+1)/2 \rfloor$. (Recall that $\lfloor x \rfloor$ is the greatest integer not exceeding x.) If $x > a_m$, the search for x is restricted to the second half of the list, which is $a_{m+1}, a_{m+2}, \ldots, a_n$. If x is not greater than a_m, the search for x is restricted to the first half of the list, which is $a_1, a_2, \ldots, a_m$.

ALGORITHM 3
The Binary Search Algorithm.

procedure *binary search* (x : integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1$ {i is left endpoint of search interval}
$j := n$ {j is right endpoint of search interval}
while $i < j$
  $m := \lfloor (i+j)/2 \rfloor$
  if $x > a_m$ then $i := m + 1$
  else $j := m$
if $x = a_i$ **then** *location* $:= i$
else *location*$:= 0$
return *location*{location is the subscript i of the term a_i equal to x, or 0 if x is not found}

The search has now been restricted to a list with no more than $\lceil n/2 \rceil$ elements. (Recall that $\lceil x \rceil$ is the smallest integer greater than or equal to x.) Using the same procedure, compare x to the middle term of the restricted list. Then restrict the search to the first or second half of the list. Repeat this process until a list with one term is obtained. Then determine whether this term is x. Pseudocode for the binary search algorithm is displayed as Algorithm 3.

Algorithm 3 proceeds by successively narrowing down the part of the sequence being searched. At any given stage only the terms from a_i to a_j are under consideration. In other words, i and j are the smallest and largest subscripts of the remaining terms, respectively. Algorithm 3 continues narrowing the part of the sequence being searched until only

one term of the sequence remains. When this is done, a comparison is made to see whether this term equals x.

3.1.3 Sorting

Ordering the elements of a list is a problem that occurs in many contexts. For example, to Demo produce a telephone directory it is necessary to alphabetize the names of subscribers. Similarly, producing a directory of songs available for downloading requires that their titles be put in alphabetic order. Putting addresses in order in an e-mail mailing list can determine whether there are duplicated addresses. Creating a useful dictionary requires that words be put in alphabetical order. Similarly, generating a parts list requires that we order them according to increasing part number.

Suppose that we have a list of elements of a set. Furthermore, suppose that we have a way to order elements of the set. **Sorting** is putting these elements into a list in which the elements are in increasing order. For instance, sorting the list 7, 2, 1, 4, 5, 9 produces the list 1, 2, 4, 5, 7, 9. Sorting the list d, h, c, a, f (using alphabetical order) produces the list a, c, d, f, h.

There are many reasons why sorting algorithms interest computer scientists and mathematicians. Among these reasons are that some algorithms are easier to implement, some algorithms are more efficient (either in general, or when given input with certain characteristics, such as lists slightly out of order), some algorithms take advantage of particular computer architectures, and some algorithms are particularly clever. In this section we will introduce two sorting algorithms, the bubble sort and the insertion sort.

THE BUBBLE SORT

The **bubble sort** is one of the simplest sorting algorithms, but not one of the most efficient. It puts a list into increasing order by successively comparing adjacent Links elements, interchanging them if they are in the wrong order. To carry out the bubble sort, we perform the basic operation, that is, interchanging a larger element with a smaller one following it, starting at the beginning of the list, for a full pass. We iterate this procedure until the sort is complete. Pseudocode for the

bubble sort is given as Algorithm 4. We can imagine the elements in the list placed in a column. In the bubble sort, the smaller elements "bubble" to the top as they are interchanged with larger elements. The larger elements "sink" to the bottom. This is illustrated in Example 4.

ALGORITHM 4
The Bubble Sort.

procedure *bubblesort*($a_1, \ldots, a_n$: real numbers with $n \geq 2$)
for $i := 1$ **to** $n - 1$
  for $j := 1$ **to** $n - i$
    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
{$a_1, \ldots, a_n$ is in increasing order}

EXAMPLE. 4

Use the bubble sort to put 3, 2, 4, 1, 5 into increasing order.

Solution: The steps of this algorithm are illustrated in Figure 3.1. Begin by comparing the first two elements, 3 and 2. Because $3 > 2$, interchange 3 and 2, producing the list 2, 3, 4, 1, 5. Because $3 < 4$, continue by comparing 4 and 1. Because $4 > 1$, interchange 1 and 4, producing the list 2, 3, 1, 4, 5. Because $4 < 5$, the first pass is complete. The first pass guarantees that the largest element, 5, is in the correct position.

The second pass begins by comparing 2 and 3. Because these are in the correct order, 3 and 1 are compared. Because $3 > 1$, these numbers are interchanged, producing 2, 1, 3, 4, 5. Because $3 < 4$, these numbers are in the correct order. It is not necessary to do any more comparisons for this pass because 5 is already in the correct position. The second pass guarantees that the two largest elements, 4 and 5, are in their correct positions. The third pass begins by comparing 2 and 1. These are interchanged because $2 > 1$, producing 1, 2, 3, 4, 5. Because $2 < 3$, these two elements are in the correct order. It is not necessary to do any more comparisons for this pass because 4 and 5 are already in the correct positions.

The third pass guarantees that the three largest elements, 3, 4, and 5, are in their correct positions.

The fourth pass consists of one comparison, namely, the comparison of 1 and 2. Because $1 < 2$, these elements are in the correct order. ∎

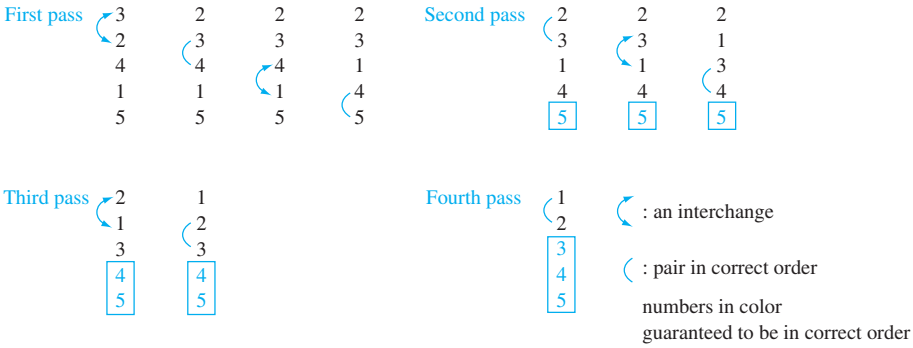


Figure 3.1: The steps of a bubble sort.

THE INSERTION SORT

The **insertion sort** is a simple sorting algorithm, but it is usually not the most efficient. To sort a list with n elements, the insertion sort begins with the second element. The insertion sort compares this second element with the first element and inserts it before the first element if it does not exceed the first element and after the first element if it exceeds the first element. At this point, the first two elements are in the correct order. The third element is then compared with the first element, and if it is larger than the first element, it is compared with the second element; it is inserted into the correct position among the first three elements.

In general, in the jth step of the insertion sort, the jth element of the list is inserted into the correct position in the list of the previously sorted $j-1$ elements. To insert the jth element in the list, a linear search technique is used; the jth element is successively compared with the already sorted $j-1$ elements at the start of the list until the first element that is not less than this element is found or until it has been compared with all $j-1$ elements; the jth element is inserted in the correct position so that the first j elements are sorted. The algorithm continues until the last element is placed in the correct position relative to the already sorted list of the first $n-1$ elements. The insertion sort is described in pseudocode in Algorithm 5.

ALGORITHM 5
The Insertion Sort.

procedure *insertion sort*($a_1, a_2, \ldots, a_n$: real numbers $n \geq 2$)
for $j := 2$ **to** n
  $i := 1$
  while $a_j > a_i$
    $i := i + 1$
  $m := a_j$
  for $k := 0$ **to** $j - i - 1$
    $a_{j-k} := a_{j-k-1}$
  $a_i := m$
{$a_1, \ldots, a_n$ is in increasing order}

EXAMPLE. 5

Use the insertion sort to put the elements of the list 3, 2, 4, 1, 5 in increasing order.

Solution: The insertion sort first compares 2 and 3. Because $3 > 2$, it places 2 in the first position, producing the list **2, 3**, 4, 1, 5 (the sorted part of the list is shown in color). At this point, 2 and 3 are in the correct order. Next, it inserts the third element, 4, into the already sorted part of the list by making the comparisons $4 > 2$ and $4 > 3$. Because $4 > 3$, 4 remains in the third position. At this point, the list is **2, 3, 4**, 1, 5 and we know that the ordering of the first three elements is correct. Next, we find the correct place for the fourth element, 1, among the already sorted elements, 2, 3, 4. Because $1 < 2$, we obtain the list **1, 2, 3, 4**, 5. Finally, we insert 5 into the correct position by successively comparing it to 1, 2, 3, and 4. Because $5 > 4$, it stays at the end of the list, producing the correct order for the entire list. ∎

3.1.4 String Matching

Although searching and sorting are the most commonly encountered problems in computer science, many other problems arise frequently.

One of these problems asks where a particular string of characters P, called the **pattern**, occurs, if it does, within another string $\mathbf{T}$, called the **text**. For instance, we can ask whether the pattern 101 can be found within the string 11001011. By inspection we can see that the pattern 101 occurs within the text 11001011 at a shift of four characters, because 101 is the string formed by the fifth, sixth, and seventh characters of the text. On the other hand, the pattern 111 does not occur within the text 110110001101.

Finding where a pattern occurs in a text string is called **string matching**. String matching plays an essential role in a wide variety of applications, including text editing, spam filters, systems that look for attacks in a computer network, search engines, plagiarism detection, bioinformatics, and many other important applications. For example, in text editing, the string matching problem arises whenever we need to find all occurrences of a string so that we can replace this string with a different string. Search engines look for matching of search keywords with words on web pages. Many problems in bioinformatics arise in the study of DNA molecules, which are made up of four bases: thymine (T), adenine (**A**), cytosine (**C**), and guanine (**G**). The process of DNA sequencing is the determination of the order of the four bases in DNA. This leads to string matching problems involving strings made up from the four letters **T, A, C,** and **G**. For instance, we can ask whether the pattern CAG occurs in the text CATCACAGAGA. The answer is yes, because it occurs with a shift of five characters. Solving questions about the genome requires the use of efficient algorithms for string matching, especially because a string representing a human genome is about 3×10^9 characters long.

We will now describe a brute force algorithm, Algorithm 6, for string matching, called the **naive string matcher**. The input to this algorithm is the pattern we wish to match, $P = p_1 p_2 \dots p_m$, and the text, $T = t_1 t_2 \dots t_n$. When this pattern begins at position $s + 1$ in the text $\mathbf{T}$, we say that $\mathbf{P}$ occurs with shift s in $\mathbf{T}$, that is, when $t_{s+1} = p_1,\ t_{s+2} = p_2, \dots, t_{s+m} = p_m$. To find all valid shifts, the naive string matcher runs through all possible shifts s from $s = 0$ to $s = n - m$, checking whether s is a valid shift. In Figure 2, we display the operation of Algorithm 6 when it is used to search for the pattern

$P = eye$ in the text $T = eceyeye$.

ALGORITHM 6
Naive String Matcher.

procedure *string match* (n, m: positive integers,
$m \leq n, t_1, t_2, \ldots, t_n, p_1, p_2, \ldots, p_m$: characters)
  for $s := 0$ to $n - m$
    $j := 1$
    while ($j \leq m$ and $t_{s+j} = p_j$)
      $j := j + 1$
    if $j > m$ **then print** "s is a valid shift"

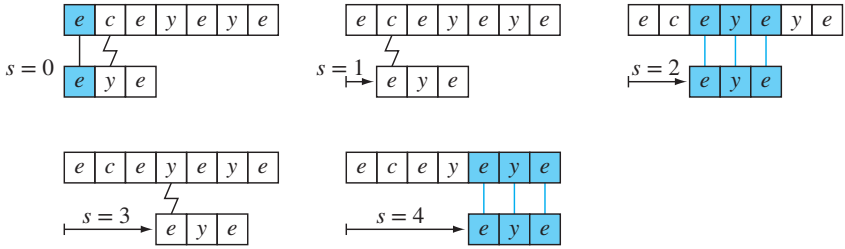


Figure 3.2: The steps of the naive string matcher with $P = eye$ in $T = eceyeye$. Matches are identified with a solid line and mismatches with a jagged line. The algorithm finds two valid shifts, $s = 2$ and $s = 4$.

Many other string matching algorithms have been developed besides the naive string matcher. These algorithms use a surprisingly wide variety of approaches to make them more efficient than the naive string matcher.

3.1.5 Creedy Algorithms

Many algorithms we will study in this book are designed to solve **optimization problems**. The goal of such problems is to find a solution to the given problem that either minimizes or maximizes the value of some parameter. Optimization problems studied later in this

text include finding a route between two cities with least total mileage, determining a way to encode messages using the fewest bits possible, and finding a set of fiber links between network nodes using the least amount of fiber.

Surprisingly, one of the simplest approaches often leads to a solution of an optimization problem. This approach selects the best choice at each step, instead of considering all sequences of steps that may lead to an optimal solution. Algorithms that make what seems to be the "best" choice at each step are called **greedy algorithms**. Once we know that a greedy algorithm finds a feasible solution, we need to determine whether it has found an optimal solution. To do this, we either prove that the solution is optimal or we show that there is a counterexample where the algorithm yields a nonoptimal solution. To make these concepts more concrete, we will consider the **cashier's algorithm** that makes change using coins.

EXAMPLE. 6

Consider the problem of making n cents change with quarters, dimes, nickels, and pennies, and using the least total number of coins. We can devise a greedy algorithm for making change for n cents by making a locally optimal choice at each step; that is, at each step we choose the coin of the largest denomination possible to add to the pile of change without exceeding n cents. For example, to make change for 67 cents, we first select a quarter (leaving 42 cents).We next select a second quarter (leaving 17 cents), followed by a dime (leaving 7 cents), followed by a nickel (leaving 2 cents), followed by a penny (leaving 1 cent), followed by a penny. ∎

We display the cashier's algorithm for n cents, using any set of denominations of coins, as Demo Algorithm 7.

We have described the cashier's algorithm, a greedy algorithm for making change, using any finite set of coins with denominations c_1, c_2, ..., c_r. In the particular case where the four denominations are quarters, dimes, nickels, and pennies, we have $c_1 = 25$, $c_2 = 10$, $c_3 = 5$, and $c_4 = 1$. For this case, we will show that this algorithm leads to an optimal solution in the sense that it uses the fewest coins possible.

Before we embark on our proof, we show that there are sets of coins for which the cashier's algorithm (Algorithm 7) does not necessarily produce change using the fewest coins possible. For example, if we have only quarters, dimes, and pennies (and no nickels) to use, the cashier's algorithm would make change for 30 cents using six coins—a quarter and five pennies—whereas we could have used three coins, namely, three dimes.

ALGORITHM 7
Cashier's Algorithm.

procedure *change*($c_1, c_2, \ldots, c_r$: values of denominations of coins, where
$c_1 > c_2 > \ldots > c_r$; n : a positive integer)
for $i := 1$ **to** r
$d_i := 0$ {d_i counts the coins of denomination c_i used}
while $n \geq c_i$
$d_i := d_i + 1$ {add a coin of denomination c_i}
$n := n - c_i$
{d_i is the number of coins of denomination c_i in the change for $i = 1, 2, \ldots, r$}

LEMMA 3.1.1

If n is a positive integer, then n cents in change using quarters, dimes, nickels, and pennies using the fewest coins possible has at most two dimes, at most one nickel, at most four pennies, and cannot have two dimes and a nickel. The amount of change in dimes, nickels, and pennies cannot exceed 24 cents.

Proof: We use a proof by contradiction. We will show that if we had more than the specified number of coins of each type, we could replace them using fewer coins that have the same value. We note that if we had three dimes we could replace them with a quarter and a nickel, if we had two nickels we could replace them with a dime, if we had five pennies we could replace them with a nickel, and if we had two dimes and a nickel we could replace them with a quarter. Because we can

have at most two dimes, one nickel, and four pennies, but we cannot have two dimes and a nickel, it follows that 24 cents is the most money we can have in dimes, nickels, and pennies when we make change using the fewest number of coins for n cents. ∎

THEOREM 3.1.1

The cashier's algorithm (Algorithm 7) always makes changes using the fewest coins possible when change is made from quarters, dimes, nickels, and pennies.

Proof: We will use a proof by contradiction. Suppose that there is a positive integer n such that there is a way to make change for n cents using quarters, dimes, nickels, and pennies that uses fewer coins than the greedy algorithm finds. We first note that q', the number of quarters used in this optimal way to make change for n cents, must be the same as q, the number of quarters used by the greedy algorithm. To show this, first note that the greedy algorithm uses the most quarters possible, so $q' \leq q$. However, it is also the case that q' cannot be less than q. If it were, we would need to make up at least 25 cents from dimes, nickels, and pennies in this optimal way to make change. But this is impossible by Lemma 3.1.1.

Because there must be the same number of quarters in the two ways to make change, the value of the dimes, nickels, and pennies in these two ways must be the same, and these coins are worth no more than 24 cents. There must be the same number of dimes, because the greedy algorithm used the most dimes possible and by Lemma 3.1.1, when change is made using the fewest coins possible, at most one nickel and at most four pennies are used, so that the most dimes possible are also used in the optimal way to make change. Similarly, we have the same number of nickels and, finally, the same number of pennies. ∎

A greedy algorithm makes the best choice at each step according to a specified criterion. The next example shows that it can be difficult to determine which of many possible criteria to choose.

EXAMPLE. 7

Suppose we have a group of proposed talks with preset start and end times. Devise a greedy algorithm to schedule as many of these talks as possible in a lecture hall, under the assumptions that once a talk starts, it continues until it ends, no two talks can proceed at the same time, and a talk can begin at the same time another one ends. Assume that talk j begins at time s_j (where s stands for *start*) and ends at time e_j (where e stands for *end*).

Solution: To use a greedy algorithm to schedule the most talks, that is, an optimal schedule, we need to decide how to choose which talk to add at each step. There are many criteria we could use to select a talk at each step, where we chose from the talks that do not overlap talks already selected. For example, we could add talks in order of earliest start time, we could add talks in order of shortest time, we could add talks in order of earliest finish time, or we could use some other criterion.

We now consider these possible criteria. Suppose we add the talk that starts earliest among the talks compatible with those already selected. We can construct a counterexample to see that the resulting algorithm does not always produce an optimal schedule. For instance, suppose that we have three talks: Talk 1 starts at 8 A.M. and ends at 12 noon, Talk 2 starts at 9 A.M. and ends at 10 A.M., and Talk 3 starts at 11 A.M. and ends at 12 noon. We first select the Talk 1 because it starts earliest. But once we have selected Talk 1 we cannot select either Talk 2 or Talk 3 because both overlap Talk 1. Hence, this greedy algorithm selects only one talk. This is not optimal because we could schedule Talk 2 and Talk 3, which do not overlap.

Now suppose we add the talk that is shortest among the talks that do not overlap any of those already selected. Again we can construct a counterexample to show that this greedy algorithm does not always produce an optimal schedule. So, suppose that we have three talks: Talk 1 starts at 8 A.M. and ends at 9:15 A.M., Talk 2 starts at 9 A.M. and ends at 10 A.M., and Talk 3 starts at 9:45 A.M. and ends at 11 A.M. We select Talk 2 because it is shortest, requiring one hour. Once we select Talk 2, we cannot select either Talk 1 or Talk 3 because neither is compatible with Talk 2. Hence, this greedy algorithm selects only one talk. However, it is possible to select two talks, Talk 1 and Talk 3, which are compatible.

However, it can be shown that we schedule the most talks possible if in each step we select the talk with the earliest ending time among the talks compatible with those already selected. We will prove this in Chapter 5 using the method of mathematical induction. The first step we will make is to sort

the talks according to increasing finish time. After this sorting, we relabel the talks so that $e_1 \leq e_2 \leq \ldots \leq e_n$. The resulting greedy algorithm is given as Algorithm 8. ∎

ALGORITHM 8
Greedy Algorithm for Scheduling Talks.

procedure *schedule*($s_1 \leq s_2 \leq \ldots \leq s_n$: start times of talks,
$e_1 \leq e_2 \leq \ldots \leq e_n$: ending times of talks)
sort talks by finish time and reorder so that $e_1 \leq e_2 \leq \ldots \leq e_n$
$S := \emptyset$
for $j := 1$ **to** n
    if talk j is compatible with S **then**
        $S := S \cup \{\text{talk } j\}$
return $S\{S$ is the set of talks scheduled$\}$

3.1.6 The Halting Problem

We will now describe a proof of one of the most famous theorems in computer science. We will show that there is a problem that cannot be solved using any procedure. That is, we will show there are unsolvable problems. The problem we will study is the halting problem. It asks whether there is a procedure that does this: It takes as input a computer program and input to the program and determines whether the program will eventually stop when run with this input. It would be convenient to have such a procedure, if it existed. Certainly being able to test whether a program entered into an infinite loop would be helpful when writing and debugging programs.

Before we present a proof that the halting problem is unsolvable, first note that we cannot simply run a program and observe what it does to determine whether it terminates when run with the given input. If the program halts, we have our answer, but if it is still running after any fixed length of time has elapsed, we do not know whether it will never halt or we just did not wait long enough for it to terminate. After all, it is not hard to design a program that will stop only after more

than a billion years has elapsed. We will describe Turing's proof that the halting problem is unsolvable; it is a proof by contradiction.

Proof: Assume there is a solution to the halting problem, a procedure called $H(P, I)$. The procedure $H(P, I)$ takes two inputs, one a program P and the other I, an input to the program P. $H(P, I)$ generates the string "halt" as output if H determines that P stops when given I as input. Otherwise, $H(P, I)$ generates the string "loops forever" as output. We will now derive a contradiction.

When a procedure is coded, it is expressed as a string of characters; this string can be interpreted as a sequence of bits. This means that a program itself can be used as data. Therefore, a program can be thought of as input to another program, or even itself. Hence, H can take a program P as both of its inputs, which are a program and input to this program. H should be able to determine whether P will halt when it is given a copy of itself as input.

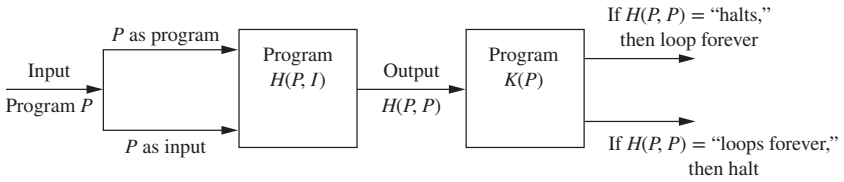


Figure 3.3: Showing that the halting problem is unsolvable.

To show that no procedure H exists that solves the halting problem, we construct a simple procedure $K(P)$, which works as follows, making use of the output $H(P, P)$. If the output of $H(P, P)$ is "loops forever," which means that P loops forever when given a copy of itself as input, then $K(P)$ halts. If the output of $H(P, P)$ is "halt," which means that P halts when given a copy of itself as input, then $K(P)$ loops forever. That is, $K(P)$ does the opposite of what the output of $H(P, P)$ specifies. (See Figure 3.3.)

Now suppose we provide K as intput to K. We note that if the output of $H(K, K)$ is "loops forever," then by the definition of K, we see that $K(K)$ halts. This means that by the definition of H, the output of $H(K, K)$ is "halt," which is a contradiction. Otherwise, if the output of $H(K, K)$ is "halts," then by the definition of K, we see that $K(K)$

loops forever, which means that by the definition of H, the output of $H(K, K)$ is "loops forever." This is also a contradiction. Thus, H cannot always give the correct answers. Consequently, there is no procedure that solves the halting problem.

3.2 The Growth of Functions

3.2.1 Introduction

In Section 3.1 we discussed the concept of an algorithm. We introduced algorithms that solve a variety of problems, including searching for an element in a list and sorting a list. In Section 3.3 we will study the number of operations used by these algorithms. In particular, we will estimate the number of comparisons used by the linear and binary search algorithms to find an element in a sequence of n elements. We will also estimate the number of comparisons used by the bubble sort and by the insertion sort to sort a list of n elements. The time required to solve a problem depends on more than only the number of operations it uses. The time also depends on the hardware and software used to run the program that implements the algorithm. However, when we change the hardware and software used to implement an algorithm, we can closely approximate the time required to solve a problem of size n by multiplying the previous time required by a constant. For example, on a supercomputer we might be able to solve a problem of size n a million times faster than we can on a PC. However, this factor of one million will not depend on n (except perhaps in some minor ways). One of the advantages of using **big-O notation**, which we introduce in this section, is that we can estimate the growth of a function without worrying about constant multipliers or smaller order terms. This means that, using big-O notation, we do not have to worry about the hardware and software used to implement an algorithm. Furthermore, using, we can assume that the different operations used in an algorithm take the same time, which simplifies the analysis considerably.

Big-O notation is used extensively to estimate the number of operations an algorithm uses as its input grows. With the help of this notation, we can determine whether it is practical to use a particular

algorithm to solve a problem as the size of the input increases. Furthermore, using big-O notation, we can compare two algorithms to determine which is more efficient as the size of the input grows. For instance, if we have two algorithms for solving a problem, one using $100n^2 + 17n + 4$ operations and the other using n^3 operations, big-O notation can help us see that the first algorithm uses far fewer operations when n is large, even though it uses more operations for small values of n, such as $n = 10$.

This section introduces big-O notation and the related big-Omega and big-Theta notations. We will explain how big-O, big-Omega, and big-Theta estimates are constructed and establish estimates for some important functions that are used in the analysis of algorithms.

3.2.2 Big-O Notation

The growth of functions is often described using a special notation. Definition 1 describes this notation.

Definition 3.2.1 *Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $O(g(x))$ if there are constants C and k such that*

$$|f(x)| \leq C|g(x)|$$

whenever $x > k$. [This is read as "f (x) is big-oh of g(x)."]

> **Remark!** Intuitively, the definition that $f(x)$ is $O(g(x))$ says that $f(x)$ grows slower than some fixed multiple of $g(x)$ as x grows without bound.

The constants C and k in the definition of big-O notation are called **witnesses** to the relationship $f(x)$ is $O(g(x))$. To establish that $f(x)$ is $O(g(x))$ we need only one pair of witnesses to this relationship. That is, to show that $f(x)$ is $O(g(x))$, we need find only one pair of constants C and k, the witnesses, such that $|f(x)| \leq C|g(x)|$ whenever $x > k$.

Note that when there is one pair of witnesses to the relationship $f(x)$ is $O(g(x))$, there are infinitely many pairs of witnesses. To see

this, note that if C and k are one pair of witnesses, then any pair C' and k', where $C < C'$ and $k < k'$, is also a pair of witnesses, because $|f(x)| \leq C|g(x)| \leq C'|g(x)|$ whenever $x > k' > k$.

THE HISTORY OF BIG-O NOTATION

Big-O notation has been used in mathematics for more than a century. In computer science it is widely used in the analysis of algorithms, as will be seen in Section 3.3. The German mathematician Paul Bachmann first introduced big-O notation in 1892 in an important book on number theory. The big-O symbol is sometimes called a **Landau symbol** after the German mathematician Edmund Landau, who used this notation throughout his work. The use of big-O notation in computer science was popularized by Donald Knuth, who also introduced the big-Ω and big-Θ notations defined later in this section.

WORKING WITH THE DEFINITION OF BIG-O NOTATION

A useful approach for finding a pair of witnesses is to first select a value of k for which the size of $|f(x)|$ can be readily estimated when $x > k$ and to see whether we can use this estimate to find a value of C for which $|f(x)| \leq C|g(x)|$ for $x > k$. This approach is illustrated in Example 1.

EXAMPLE. 1

Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.

Solution: We observe that we can readily estimate the size of $f(x)$ when $x > 1$ because $x < x^2$ and $1 < x^2$ when $x > 1$. It follows that

$$0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$$

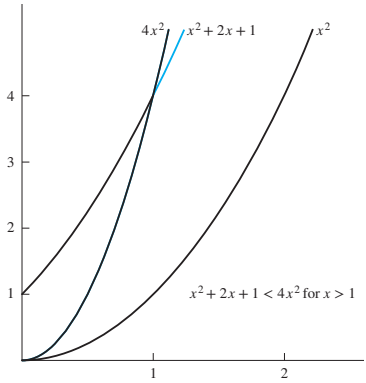
whenever $x > 1$, as shown in Figure 3.4. Consequently, we can take $C = 4$ and $k = 1$ as witnesses to show that $f(x)$ is $O(x^2)$. That is, $f(x) = x^2 + 2x + 1 < 4x^2$ whenever $x > 1$. (Note that it is not necessary to use absolute values here because all functions in these equalities are positive when x is positive.) Alternatively, we can estimate the size of $f(x)$ when $x > 2$. When $x > 2$, we have $2x \leq x^2$ and $1 \leq x^2$. Consequently, if $x > 2$, we have

$$0 \leq x^2 + 2x + 1 \leq x^2 + x^2 + x^2 = 3x^2$$

. It follows that $C = 3$ and $k = 2$ are also witnesses to the relation $f(x)$ is $O(x^2)$.

Observe that in the relationship "$f(x)$ is $O(x^2)$," x^2 can be replaced by any function that has larger values than x^2 for all $x \geq k$ for some positive real number k. For example, $f(x)$ is $O(x^3)$, $f(x)$ is $O(x^2 + x + 7)$, and so on.

It is also true that x^2 is $O(x^2+2x+1)$, because $x^2 < x^2+2x+1$ whenever $x > 1$. This means that $C = 1$ and $k = 1$ are witnesses to the relationship x^2 is $O(x^2 + 2x + 1)$. ∎



The part of the graph of $f(x) = x^2 + 2x + 1$ that satisfies $f(x) < 4x^2$ is shown in color.

Figure 3.4: The function $x^2 + 2x + 1$ is $O(x^2)$.

Note that in Example 1 we have two functions, $f(x) = x^2 + 2x + 1$ and $g(x) = x^2$, such that $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$ — the latter fact following from the inequality $x^2 \leq x^2 + 2x + 1$, which holds for all nonnegative real numbers x. We say that two functions $f(x)$ and $g(x)$ that satisfy both of these big-O relationships are of the **same order**. We will return to this notion later in this section.

Remark! The fact that $f(x)$ is $O(g(x))$ is sometimes written $f(x) = O(g(x))$. However, the equals sign in this notation does not represent a genuine equality. Rather, this notation tells us that an inequality holds relating the values of the functions f and g for sufficiently large numbers in the domains of these functions.

> However, it is acceptable to write $f(x) \in O(g(x))$ because $O(g(x))$ represents the set of functions that are $O(g(x))$.

When $f(x)$ is $O(g(x))$, and $h(x)$ is a function that has larger absolute values than $g(x)$ does for sufficiently large values of x, it follows that $f(x)$ is $O(h(x))$. In other words, the function $g(x)$ in the relationship $f(x)$ is $O(g(x))$ can be replaced by a function with larger absolute values. To see this, note that if

$$|f(x)| \leq C|g(x)| \quad \text{if } x > k,$$

and if $|h(x)| > |g(x)|$ for all $x > k$, then

$$|f(x)| \leq C|h(x)| \quad \text{if } x > k.$$

Hence, $f(x)$ is $O(h(x))$.

When big-O notation is used, the function g in the relationship $f(x)$ is $O(g(x))$ is often chosen to have the smallest growth rate of the functions belonging to a set of reference functions, such as functions of the form x^n, where n is a positive real number.

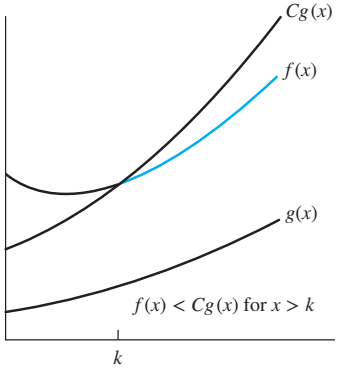
In subsequent discussions, we will almost always deal with functions that take on only positive values. All references to absolute values can be dropped when working with big-O estimates for such functions. Figure 3.5 illustrates the relationship $f(x)$ is $O(g(x))$.

Example 2 illustrates how big-O notation is used to estimate the growth of functions.

EXAMPLE. 2

Show that $7x^2$ is $O(x^3)$.

Solution: Note that when $x > 7$, we have $7x^2 < x^3$. (We can obtain this inequality by multiplying both sides of $x > 7$ by x^2.) Consequently, we can take $C = 1$ and $k = 7$ as witnesses to establish the relationship $7x^2$ is $O(x^3)$. Alternatively, when $x > 1$, we have $7x^2 < 7x^3$, so that $C = 7$ and $k = 1$ are also witnesses to the relationship $7x^2$ is $O(x^3)$. ∎



The part of the graph of $f(x)$ that satisfies $f(x) < Cg(x)$ is shown in color.

Figure 3.5: The function $f(x)$ is $O(g(x))$.

Remark! In Example 2 we did not choose the smallest possible power of x the reference function in the big-O estimate. Note that $7x^2$ is also big-O of x^2 and x^2 grows much slower than x^3. In fact, x^2 would be the smallest possible power of x suitable as the reference function in the big-O estimate.

Example 3 illustrates how to show that a big-O relationship does not hold.

EXAMPLE. 3

Show that n^2 is not $O(n)$.

Solution: To show that n^2 is not $O(n)$, we must show that no pair of witnesses C and k exist such that $n^2 \leq Cn$ whenever $n > k$. We will use a proof by contradiction to show this.

Suppose that there are constants C and k for which $n^2 \leq Cn$ whenever $n > k$. Observe that when $n > 0$ we can divide both sides of the inequality $n^2 \leq Cn$ by n to obtain the equivalent inequality $n \leq C$. However, no matter what C and k are, the inequality $n \leq C$ cannot hold for all n with $n > k$. In particular, once we set a value of k, we see that when n is larger than the maximum of k and C, it is not true that $n \leq C$ even though $n > k$. This contradiction shows that n^2 is not $O(n)$. ∎

EXAMPLE. 4

Example 2 shows that $7x^2$ is $O(x^3)$. Is it also true that x^3 is $O(7x^2)$?

Solution: To determine whether x^3 is $O(7x^2)$, we need to determine whether witnesses C and k exist, so that $x^3 \leq C(7x^2)$ whenever $x > k$. We will show that no such witnesses exist using a proof by contradiction.

If C and k are witnesses, the inequality $x^3 \leq C(7x^2)$ holds for all $x > k$. Observe that the inequality $x^3 \leq C(7x^2)$ is equivalent to the inequality $x \leq 7C$, which follows by dividing both sides by the positive quantity x^2. However, no matter what C is, it is not the case that $x \leq 7C$ for all $x > k$ no matter what k is, because x can be made arbitrarily large. It follows that no witnesses C and k exist for this proposed big-O relationship. Hence, x^3 is not $O(7x^2)$. ∎

3.2.3 Big-O Estimates for Some Important Functions

Polynomials can often be used to estimate the growth of functions. Instead of analyzing the growth of polynomials each time they occur, we would like a result that can always be used to estimate the growth of a polynomial. Theorem 3.2.1 does this. It shows that the leading term of a polynomial dominates its growth by asserting that a polynomial of degree n or less is $O(x^n)$.

THEOREM 3.2.1

Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$, where $a_0,\ a_1, \ldots, a_{n-1}$, a_n are real numbers. Then $f(x)$ is $O(x_n)$.

Proof: Using the triangle inequality, if $x > 1$ we have

$$\begin{aligned} |f(x)| &= |a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0| \\ &\leq |a_n|x^n + |a_{n-1}|x^{n-1} + \ldots + |a_1|x + |a_0| \\ &= x^n(|a_n| + |a_{n-1}|/x + \ldots + |a_1|/x_{n-1} + |a_0|/x^n) \\ &\leq x^n(|a_n| + |a_{n-1}| + \ldots + |a_1| + |a_0|) \end{aligned}$$

This shows that

$$|f(x)| \leq Cx^n,$$

where $C = |a_n|+|a_{n-1}|+\ldots+|a_0|$ whenever $x > 1$. Hence, the witnesses $C = |a_n| + |a_{n-1}| + \ldots + |a_0|$ and $k = 1$ show that $f(x)$ is $O(x^n)$. ∎

We now give some examples involving functions that have the set of positive integers as their domains.

EXAMPLE. 5

How can big-O notation be used to estimate the sum of the first n positive integers?

Solution: Because each of the integers in the sum of the first n positive integers does not exceed n, it follows that

$$1 + 2 + \ldots + n \leq n + n + \ldots + n = n^2.$$

From this inequality it follows that $1 + 2 + 3 + \ldots + n$ is $O(n^2)$, taking $C = 1$ and $k = 1$ as witnesses. (In this example the domains of the functions in the big-O relationship are the set of positive integers.) ∎

In Example 6 big-O estimates will be developed for the factorial function and its logarithm. These estimates will be important in the analysis of the number of steps used in sorting procedures.

EXAMPLE. 6

Give big-O estimates for the factorial function and the logarithm of the factorial function, where the factorial function $f(n) = n!$ is defined by

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$$

whenever n is a positive integer, and $0! = 1$. For example,

$$1! = 1,\ 2! = 1 \cdot 2 = 2,\ 3! = 1 \cdot 2 \cdot 3 = 6,\ 4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24.$$

Note that the function $n!$ grows rapidly.

For instance, $20! = 2,432,902,008,176,640,000$.

Solution: A big-O estimate for $n!$ can be obtained by noting that each term in the product does not exceed n. Hence,

$$\begin{aligned} n! &= 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \\ &\leq n \cdot n \cdot n \cdot \ldots \cdot n \\ &= n^n. \end{aligned}$$

This inequality shows that $n!$ is $O(n^n)$, taking $C = 1$ and $k = 1$ as witnesses. Taking logarithms of both sides of the inequality established for $n!$, we obtain

$$log\ n! \leq log\ n^n = n log\ n.$$

This implies that $log\ n!$ is $O(n log\ n)$, again taking $C = 1$ and $k = 1$ as witnesses. ∎

EXAMPLE. 7

In next sections, we will show that $n < 2^n$ whenever n is a positive integer. Show that this inequality implies that n is $O(2^n)$, and use this inequality to show that $logn$ is $O(n)$.

Solution: Using the inequality $n < 2^n$, we quickly can conclude that n is $O(2^n)$ by taking $k = C = 1$ as witnesses. Note that because the logarithm function is increasing, taking logarithms (base 2) of both sides of this inequality shows that

$$log\ n < n.$$

It follows that

$$log\ n \text{ is } O(n).$$

(Again we take $C = k = 1$ as witnesses.)

If we have logarithms to a base b, where b is different from 2, we still have $log\ bn$ is $O(n)$ because

$$log_b\ n = \frac{log\ n}{log\ b} < \frac{n}{log\ b}$$

whenever n is a positive integer. We take $C = 1/log\ b$ and $k = 1$ as witnesses.

∎

As mentioned before, big-O notation is used to estimate the number of operations needed to solve a problem using a specified procedure or

algorithm. The functions used in these estimates often include the following:

$$1,\ log\ n,\ n,\ nlog\ n,\ n^2,\ 2^n,\ n!$$

Using calculus it can be shown that each function in the list is smaller than the succeeding function, in the sense that the ratio of a function and the succeeding function tends to zero as n grows without bound. Figure 3.6 displays the graphs of these functions, using a scale for the values of the functions that doubles for each successive marking on the graph. That is, the vertical scale in this graph is logarithmic.

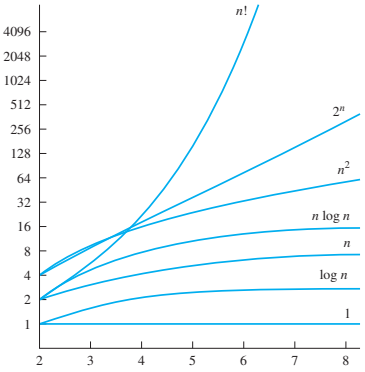


Figure 3.6: A display of the growth of functions commonly used in big-O estimates.

USEFUL BIG-O ESTIMATES INVOLVING LOGARITHMS, POWERS, AND EXPONENTIAL FUNCTIONS

We now give some useful facts that help us determine whether big-O relationships hold between pairs of functions when each of the functions is a power of a logarithm, a power, or an exponential function of the form b^n where $b > 1$.

Theorem 3.2.1 shows that if $f(n)$ is a polynomial of degree d or less, then $f(n)$ is $O(n^d)$. Applying this theorem, we see that if $d > c > 1$, then n^c is $O(n^d)$. We leave it to the reader to show that the reverse of this relationship does not hold. Putting these facts together, we see

that if $d > c > 1$, then

$$n^c \text{ is } O(n^d), \text{ but } n^d \text{ is not } O(n^c)$$

.

In Example 7 we showed that $log_b\ n$ is $O(n)$ whenever $b > 1$. More generally, whenever $b > 1$ and c and d are positive, we have

$$(log_b\ n)^c \text{ is } O(n^d), \text{ but } n^d \text{ is not } (O(log_b\ n)^c).$$

This tells us that every positive power of the logarithm of n to the base b, where $b > 1$, is big-O of every positive power of n, but the reverse relationship never holds.

In Example 7, we also showed that n is $O(2^n)$. More generally, whenever d is positive and $b > 1$, we have

$$n^d \text{ is } O(b^n), \text{ but } b^n \text{ is not } O(n^d).$$

This tells us that every power of n is big-O of every exponential function of n with a base that is greater than one, but the reverse relationship never holds. Furthermore, when $c > b > 1$ we have

$$b^n \text{ is } O(c^n), \text{ but } c^n \text{ is not } O(b^n).$$

This tells us that if we have two exponential functions with different bases greater than one, one of these functions is big-O of the other if and only if its base is smaller or equal.

Finally, we note that if $c > 1$, we have

$$c^n \text{ is } O(n!), \text{ but } n! \text{ is not } O(c^n).$$

We can use the big-O estimates discussed here to help us order the growth of different functions, as Example 8 illustrates.

EXAMPLE. 8

Arrange the functions $f_1(n) = 8\sqrt{n}$, $f_2(n) = (log\ n)^2$, $f_3(n) = 2nlog\ n$, $f_4(n) = n!$, $f_5(n) = (1.1)^n$, and $f_6(n) = n^2$ in a list so that each function is big-O of the next function.

Solution: From the big-O estimates described in this subsection, we see that $f_2(n) = (log\ n)^2$ is the slowest growing of these functions. (This follows because $log\ n$ grows slower than any positive power of n.) The next three functions, in order, are $f_1(n) = 8\sqrt{n}n = f_3(n) = 2nlog\ n$, and $f_6(n) = n^2$. (We know this because $f_1(n) = 8n^{1/2}, f_3(n) = 2nlog\ n$ is a function that grows faster than n but slower than n^c for every $c > 1$, and $f_6(n) = n^2$ is of the form n^c where $c = 2$.) The next function in the list is $f_5(n) = (1.1)^n$, because it is an exponential function with base 41.1. Finally, $f_4(n) = n!$ is the fastest growing function on the list, because $f(n) = n!$ grows faster than any exponential function of n. ∎

3.2.4 The Growth of Combinations of Functions

Many algorithms are made up of two or more separate subprocedures. The number of steps used by a computer to solve a problem with input of a specified size using such an algorithm is the sum of the number of steps used by these subprocedures. To give a big-O estimate for the number of steps needed, it is necessary to find big-O estimates for the number of steps used by each subprocedure and then combine these estimates.

Big-O estimates of combinations of functions can be provided if care is taken when different big-O estimates are combined. In particular, it is often necessary to estimate the growth of the sum and the product of two functions. What can be said if big-O estimates for each of two functions are known? To see what sort of estimates hold for the sum and the product of two functions, suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$.

From the definition of big-O notation, there are constants C_1, C_2, k_1, and k_2 such that

$$|f_1(x)| \leq C_1|g_1(x)|$$

when $x > k_1$, and

$$|f_2(x)| \leq C_2|g_2(x)|$$

when $x > k_2$. To estimate the sum of $f_1(x)$ and $f_2(x)$, note that

$$\begin{aligned} |(f_1 + f_2)(x)| &= |f_1(x) + f_2(x)| \\ &\leq |f_1(x)| + |f_2(x)| \end{aligned}$$

When x is greater than both k_1 and k_2, it follows from the inequalities for $|f_1(x)|$ and $|f_2(x)|$ that

$$\begin{aligned} |f_1(x)| + |f_2(x)| &\leq C_1|g_1(x)| + C_2|g_2(x)| \\ &\leq C_1|g(x)| + C_2|g(x)| \\ &= (C_1 + C_2)|g(x)| \\ &= C|g(x)|, \end{aligned}$$

where $C = C_1 + C_2$ and $g(x) = max(|g_1(x)|,\ |g_2(x)|)$. [Here $max(a, b)$ denotes the maximum, or larger, of a and b.]

This inequality shows that $|(f_1 + f_2)(x)| \leq C|g(x)|$ whenever $x > k$, where $k = max(k_1, k_2)$.

We state this useful result as Theorem 3.2.2.

THEOREM 3.2.2

Suppose that $f_1(x)$ is $O(g_1(x))$ and that $f_2(x)$ is $O(g_2(x))$. Then $(f_1 + f_2)(x)$ is $O(g(x))$, where $g(x) = (max(|g_1(x)|,\ |g_2(x)|)$ for all x.

We often have big-O estimates for f_1 and f_2 in terms of the same function g. In this situation, Theorem 3.2.2 can be used to show that $(f_1 + f_2)(x)$ is also $O(g(x))$, because $max(g(x), g(x)) = g(x)$. This result is stated in Corollary 1.

COROLLARY 1 Suppose that $f_1(x)$ and $f_2(x)$ are both $O(g(x))$. Then $(f_1 + f_2)(x)$ is $O(g(x))$.

In a similar way big-O estimates can be derived for the product of the functions f_1 and f_2.

When x is greater than $max(k_1, k_2)$ it follows that

$$\begin{aligned} |(f_1 f_2)(x)| &= |f_1(x)||f_2(x)| \\ &\leq C_1|g_1(x)|C_2|g_2(x)| \\ &\leq C_1C_2|(g_1g_2)(x)| \\ &\leq C|(g_1g_2)(x)|, \end{aligned}$$

where $C = C_1C_2$. From this inequality, it follows that $f_1(x)f_2(x)$ is $O(g_1g_2(x))$, because there are constants C and k, namely, $C = C_1C_2$ and $k = max(k_1, k_2)$, such that $|(f_1f_2)(x)| \leq C|g_1(x)g_2(x)|$ whenever $x > k$. This result is stated in Theorem 3.2.3

THEOREM 3.2.3

Suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$. Then $(f_1 f_2)(x)$ is $O(g_1(x)g_2(x))$.

The goal in using big-O notation to estimate functions is to choose a function $g(x)$ as simple as possible, that grows relatively slowly so that $f(x)$ is $O(g(x))$. Examples 9 and 10 illustrate how to use Theorems 3.2.2 and Theorem 3.2.3 to do this. The type of analysis given in these examples is often used in the analysis of the time used to solve problems using computer programs.

EXAMPLE. 9

Give a big-O estimate for $f(n) = 3n\log(n!) + (n^2 + 3)\log n$, where n is a positive integer.

Solution: First, the product $3n\log(n!)$ will be estimated. From Example 6 we know that $log\ (n!)$ is $O(n\log n)$. Using this estimate and the fact that $3n$ is $O(n)$, Theorem 3.2.2 gives the estimate that $3n\log(n!)$ is $O(n^2\log n)$.

Next, the product $(n^2+3)\log n$ will be estimated. Because $(n^2+3) < 2n^2$ when $n > 2$, it follows that $n^2 + 3$ is $O(n^2)$. Thus, from Theorem 3.2.3 it follows that $(n^2+3)\log n$ is $O(n^2\log n)$. Using Theorem 3.2.3 to combine the two big-O estimates for the products shows that $f(n) = 3n\log(n!) + (n^2+3)\log n$ is $O(n^2\log n)$. ∎

EXAMPLE. 10

Give a big-O estimate for $f(x) = (x+1)\log(x^2+1) + 3x^2$.

Solution: First, a big-O estimate for $(x+1)\log(x^2+1)$ will be found. Note that $(x+1)$ is $O(x)$. Furthermore, $x^2 + 1 \le 2x^2$ when $x > 1$. Hence,

$$\log(x^2+1) \le \log(2x^2) = \log 2 + \log x^2 = \log 2 + 2\log x \le 3\log x,$$

if $x > 2$. This shows that $\log(x^2+1)$ is $O(\log x)$.

From Theorem 3.2.3 it follows that $(x+1)\log(x^2+1)$ is $O(x\log x)$. Because $3x^2$ is $O(x^2)$, Theorem 3.2.2 tells us that $f(x)$ is $O(max(x\log x,\ x^2))$.

Because $x \log x \leq x^2$, for $x > 1$, it follows that $f(x)$ is $O(x^2)$. ∎

3.2.5 Big-Omega and Big-Theta Notation

Big-O notation is used extensively to describe the growth of functions, but it has limitations. In particular, when $f(x)$ is $O(g(x))$, we have an upper bound, in terms of $g(x)$, for the size of $f(x)$ for large values of x. However, big-O notation does not provide a lower bound for the size of $f(x)$ for large x. For this, we use **big-Omega (big-Ω) notation**. When we want to give both an upper and a lower bound on the size of a function $f(x)$, relative to a reference function $g(x)$, we use **big-Theta (big-Θ) notation**. Both big-Omega and big-Theta notation were introduced by Donald Knuth in the 1970s. His motivation for introducing these notations was the common misuse of big-O notation when both an upper and a lower bound on the size of a function are needed.

Definition 3.2.2 *Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Omega(g(x))$ if there are constants C and k with C positive such that*

$$|f(x)| \geq C|g(x)|$$

whenever $x > k$. [This is read as "$f(x)$ is big-Omega of $g(x)$."]

There is a strong connection between big-O and big-Omega notation. In particular, $f(x)$ is $\Omega(g(x))$ if and only if $g(x)$ is $O(f(x))$. We leave the verification of this fact as a straightforward exercise for the reader.

EXAMPLE. 11

The function $f(x) = 8x^3 + 5x^2 + 7$ is $\Omega(g(x))$, where $g(x)$ is the function $g(x) = x^3$. This is easy to see because $f(x) = 8x^3 + 5x^2 + 7 \geq 8x^3$ for all positive real numbers x. This is equivalent to saying that $g(x) = x^3$ is $O(8x^3 + 5x^2 + 7)$, which can be established directly by turning the inequality

around. ∎

Often, it is important to know the order of growth of a function in terms of some relatively simple reference function such as x^n when n is a positive integer or c^x, where $c > 1$. Knowing the order of growth requires that we have both an upper bound and a lower bound for the size of the function. That is, given a function $f(x)$, we want a reference function $g(x)$ such that $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$. Big-Theta notation, defined as follows, is used to express both of these relationships, providing both an upper and a lower bound on the size of a function.

Definition 3.2.3 *Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Theta(g(x))$ if $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$. When $f(x)$ is $\Theta(g(x))$, we say that f is big-Theta of $g(x)$, that $f(x)$ is of order $g(x)$, and that $f(x)$ and $g(x)$ are of the same order.*

When $f(x)$ is $\Theta(g(x))$, it is also the case that $g(x)$ is $\Theta(f(x))$. Also note that $f(x)$ is $\Theta(g(x))$ if and only if $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$. Furthermore, note that $f(x)$ is $\Theta(g(x))$ if and only if there are positive real numbers C_1 and C_2 and a positive real number k such that

$$C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$$

whenever $x > k$. The existence of the constants C_1, C_2, and k tells us that $f(x)$ is $\Omega(g(x))$ and that $f(x)$ is $O(g(x))$, respectively.

Usually, when big-Theta notation is used, the function $g(x)$ in $\Omega(g(x))$ is a relatively simple reference function, such as x^n, c^x, $log\ x$, and so on, while $f(x)$ can be relatively complicated.

EXAMPLE. 12

We showed (in Example 5) that the sum of the first n positive integers is $O(n^2)$. Determine whether this sum is of order n^2 without using the summation formula for this sum.

Solution: Let $f(n) = 1 + 2 + 3 + \ldots + n$. Because we already know that $f(n)$ is $O(n^2)$, to show that $f(n)$ is of order n^2 we need to find a positive

constant C such that $f(n) > Cn^2$ for sufficiently large integers n. To obtain a lower bound for this sum, we can ignore the first half of the terms. Summing only the terms greater than $\lceil n/2 \rceil$, we find that

$$\begin{aligned} 1 + 2 + \ldots + n &\geq \lceil n/2 \rceil + (\lceil n/2 \rceil + 1) + \ldots + n \\ &\geq \lceil n/2 \rceil + \lceil n/2 \rceil + \ldots + \lceil n/2 \rceil \\ &= (n - \lceil n/2 \rceil + 1)\lceil n/2 \rceil \\ &\geq (n/2)(n/2) \\ &= n^2/4. \end{aligned}$$

This shows that $f(n)$ is $\Omega(n^2)$. We conclude that $f(n)$ is of order n^2, or in symbols, $f(n)$ is $\Theta(n^2)$. ∎

EXAMPLE. 13

Show that $3x^2 + 8x log\ x$ is $\Theta(x^2)$.

Solution: Because $0 \leq 8x log\ x \leq 8x^2$, it follows that $3x^2 + 8x log\ x \leq 11x^2$ for $x > 1$. Consequently, $3x^2 + 8x log\ x$ is $O(x^2)$. Clearly, x^2 is $O(3x^2 + 8x log\ x)$. Consequently, $3x^2 + 8x log\ x$ is $\Theta(x^2)$. ∎

One useful fact is that the leading term of a polynomial determines its order. For example, if $f(x) = 3x^5 + x^4 + 17x^3 + 2$, then $f(x)$ is of order x^5. This is stated in Theorem 3.2.4.

THEOREM 3.2.4

Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$, where $a_0,\ a_1, \ldots, a_n$ are real numbers with $a_n \neq 0$. Then $f(x)$ is of order x_n.

EXAMPLE. 14

The polynomials $3x^8 + 10x^7 + 221x^2 + 1444$, $x^{19} - 18x^4 - 10{,}112$, and $x^{99} + 40,001x^{98} + 100,003x$ are of orders x^8, x^{19}, and x^{99}, respectively. ∎

Unfortunately, as Knuth observed, big-O notation is often used by careless writers and speakers as if it had the same meaning as big-Theta notation. Keep this in mind when you see big-O notation used. The

recent trend has been to use big-Theta notation whenever both upper and lower bounds on the size of a function are needed.

3.3 Complexity of Algorithms

3.3.1 Introduction

When does an algorithm provide a satisfactory solution to a problem? First, it must always produce the correct answer. Second, it should be efficient. The efficiency of algorithms will be discussed in this section.

How can the efficiency of an algorithm be analyzed? One measure of efficiency is the time used by a computer to solve a problem using the algorithm, when input values are of a specified size. A second measure is the amount of computer memory required to implement the algorithm when input values are of a specified size.

Questions such as these involve the **computational complexity** of the algorithm. An analysis of the time required to solve a problem of a particular size involves the **time complexity** of the algorithm. An analysis of the computer memory required involves the **space complexity** of the algorithm. Considerations of the time and **space complexity** of an algorithm are essential when algorithms are implemented. It is important to know whether an algorithm will produce an answer in a microsecond, a minute, or a billion years. Likewise, the required memory must be available to solve a problem, so that space complexity must be taken into account.

Considerations of space complexity are tied in with the particular data structures used to implement the algorithm. Because data structures are not dealt with in detail in this book, space complexity will not be considered. We will restrict our attention to time complexity.

3.3.2 Time Complexity

The time complexity of an algorithm can be expressed in terms of the number of operations used by the algorithm when the input has a particular size. The operations used to measure time complexity can be

the comparison of integers, the addition of integers, the multiplication of integers, the division of integers, or any other basic operation.

Time complexity is described in terms of the number of operations required instead of actual computer time because of the difference in time needed for different computers to perform basic operations. Moreover, it is quite complicated to break all operations down to the basic bit operations that a computer uses. Furthermore, the fastest computers in existence can perform basic bit operations (for instance, adding, multiplying, comparing, or exchanging two bits) in 10^{-11} second (10 picoseconds), but personal computers may require 10^{-8} second (10 nanoseconds), which is 1000 times as long, to do the same operations.

We illustrate how to analyze the time complexity of an algorithm by considering Algorithm 1 of Section 3.1, which finds the maximum of a finite set of integers.

EXAMPLE. 1

Describe the time complexity of Algorithm 1 of Section 3.1 for finding the maximum element in a finite set of integers.

Solution: The number of comparisons will be used as the measure of the time complexity of the algorithm, because comparisons are the basic operations used.

To find the maximum element of a set with n elements, listed in an arbitrary order, the temporary maximum is first set equal to the initial term in the list. Then, after a comparison $i \leq n$ has been done to determine that the end of the list has not yet been reached, the temporary maximum and second term are compared, updating the temporary maximum to the value of the second term if it is larger. This procedure is continued, using two additional comparisons for each term of the list—one $i \leq n$, to determine that the end of the list has not been reached and another $\max < a_i$, to determine whether to update the temporary maximum. Because two comparisons are used for each of the second through the nth elements and one more comparison is used to exit the loop when $i = n + 1$, exactly $2(n - 1) + 1 = 2n - 1$ comparisons are used whenever this algorithm is applied. Hence, the algorithm for finding the maximum of a set of n elements has time complexity $\Theta(n)$, measured in terms of the number of comparisons used. Note that for this algorithm the number of comparisons is independent of particular input of n numbers. ∎

Next, we will analyze the time complexity of searching algorithms.

EXAMPLE. 2

Describe the time complexity of the linear search algorithm (specified as Algortihm 2 in Section 3.1).

Solution: The number of comparisons used by Algorithm 2 in Section 3.1 will be taken as the measure of the time complexity. At each step of the loop in the algorithm, two comparisons are performed—one $i \leq n$, to see whether the end of the list has been reached and one $x \leq a_i$, to compare the element x with a term of the list. Finally, one more comparison $i \leq n$ is made outside the loop. Consequently, if $x = a_i$, $2i + 1$ comparisons are used. The most comparisons, $2n + 2$, are required when the element is not in the list. In this case, $2n$ comparisons are used to determine that x is not a_i, for $i = 1, 2, \ldots, n$, an additional comparison is used to exit the loop, and one comparison is made outside the loop. So when x is not in the list, a total of $2n + 2$ comparisons are used. Hence, a linear search requires $\Theta(n)$ comparisons in the worst case, because $2n + 2$ is $\Theta(n)$. ∎

WORST-CASE COMPLEXITY

The type of complexity analysis done in Example 2 is a worst-case analysis. By the worst-case performance of an algorithm, we mean the largest number of operations needed to solve the given problem using this algorithm on input of specified size. Worst-case analysis tells us how many operations an algorithm requires to guarantee that it will produce a solution.

EXAMPLE. 3

Describe the time complexity of the binary search algorithm (specified as Algorithm 3 in Section 3.1) in terms of the number of comparisons used (and ignoring the time required to compute $m = \lfloor (i + j)/2 \rfloor$ in each iteration of the loop in the algorithm).

Solution: For simplicity, assume there are $n = 2k$ elements in the list $a_1, a_2, \ldots, a_n$, where k is a nonnegative integer. Note that $k = log\ n$. (If

n, the number of elements in the list, is not a power of 2, the list can be considered part of a larger list with $2k + 1$ elements, where $2k < n < 2k + 1$. Here $2k + 1$ is the smallest power of 2 larger than n.)

At each stage of the algorithm, i and j, the locations of the first term and the last term of the restricted list at that stage, are compared to see whether the restricted list has more than one term. If $i < j$, a comparison is done to determine whether x is greater than the middle term of the restricted list.

At the first stage the search is restricted to a list with 2^{k-1} terms. So far, two comparisons have been used. This procedure is continued, using two comparisons at each stage to restrict the search to a list with half as many terms. In other words, two comparisons are used at the first stage of the algorithm when the list has 2^k elements, two more when the search has been reduced to a list with 2^{k-1} elements, two more when the search has been reduced to a list with 2^{k-2} elements, and so on, until two comparisons are used when the search has been reduced to a list with $2^1 = 2$ elements. Finally, when one term is left in the list, one comparison tells us that there are no additional terms left, and one more comparison is used to determine if this termis x.

Hence, at most $2k + 2 = 2log\ n + 2$ comparisons are required to perform a binary search when the list being searched has 2^k elements. (If n is not a power of 2, the original list is expanded to a list with 2^{k+1} terms, where $k = \lfloor log\ n \rfloor$, and the search requires at most $2\lceil log\ n \rceil + 2$ comparisons.) It follows that in the worst case, binary search requires $O(log\ n)$ comparisons.

Note that in the worst case, $2log\ n + 2$ comparisons are used by the binary search. Hence, the binary search uses $\Theta(log\ n)$ comparisons in the worst case, because $2log\ n + 2 = \Theta(log\ n)$. From this analysis it follows that in the worst case, the binary search algorithm is more efficient than the linear search algorithm, because we know by Example 2 that the linear search algorithm has $\Theta(n)$ worst-case time complexity. ∎

AVERAGE-CASE COMPLEXITY

Another important type of complexity analysis, besides worst-case analysis, is called **average-case** analysis. The average number of operations used to solve the problem over all possible inputs of a given size is found in this type of analysis. Average-case time complexity analysis is usually much more complicated than worst-case analysis. However, the average-case analysis for the linear search algorithm can be done without difficulty, as shown in Example 4.

EXAMPLE. 4

Describe the average-case performance of the linear search algorithm in terms of the average number of comparisons used, assuming that the integer x is in the list and it is equally likely that x is in any position.

Solution: By hypothesis, the integer x is one of the integers $a_1, a_2, \ldots, a_n$ in the list. If x is the first term a_1 of the list, three comparisons are needed, one $i \leq n$ to determine whether the end of the list has been reached, one $x \neq a_i$ to compare x and the first term, and one $i \leq n$ outside the loop. If x is the second term a_2 of the list, two more comparisons are needed, so that a total of five comparisons are used. In general, if x is the ith term of the list a_i, two comparisons will be used at each of the i steps of the loop, and one outside the loop, so that a total of $2i+1$ comparisons are needed. Hence, the average number of comparisons used equals

$$\frac{3+5+7+\ldots+(2n+1)}{n} = \frac{2(1+2+3+\ldots+n)+n}{n}.$$

Using the formula from line 2 of Table 2 in Section 2.4,

$$1+2+3+\ldots+n = \frac{n(n+1)}{2}.$$

Hence, the average number of comparisons used by the linear search algorithm (when x is known to be in the list) is

$$\frac{2\lfloor n(n+1)/2 \rfloor}{n} + 1 = n+2,$$

which is $\Theta(n)$. ∎

Remark! In the analysis in Example 4 we assumed that x is in the list being searched. It is also possible to do an average-case analysis of this algorithm when x may not be in the list.

Remark! Although we have counted the comparisons needed to determine whether we have reached the end of a loop, these comparisons are often not counted. From this point on we will

ignore such comparisons.

WORST-CASE COMPLEXITY OF TWO SORTING ALGORITHMS

We analyze the worst-case complexity of the bubble sort and the insertion sort in Examples 5 and 6.

EXAMPLE. 5

What is the worst-case complexity of the bubble sort in terms of the number of comparisons made?

Solution: The bubble sort described before Example 4 in Section 3.1 sorts a list by performing a sequence of passes through the list. During each pass the bubble sort successively compares adjacent elements, interchanging them if necessary. When the ith pass begins, the $i-1$ largest elements are guaranteed to be in the correct positions. During this pass, $n-i$ comparisons are used. Consequently, the total number of comparisons used by the bubble sort to order a list of n elements is

$$(n-1)+(n-2)+\ldots+2+1=\frac{(n-1)n}{2}$$

Note that the bubble sort always uses this many comparisons, because it continues even if the list becomes completely sorted at some intermediate step. Consequently, the bubble sort uses $(n-1)n/2$ comparisons, so it has $\Theta(n^2)$ worst-case complexity in terms of the number of comparisons used. ∎

EXAMPLE. 6

What is the worst-case complexity of the insertion sort in terms of the number of comparisons made?

Solution: The insertion sort (described in Section 3.1) inserts the jth element into the correct position among the first $j-1$ elements that have already been put into the correct order. It does this by using a linear search technique, successively comparing the jth element with successive terms until a term that is greater than or equal to it is found or it compares a_j with itself and stops because a_j is not less than itself. Consequently, in the worst case, j comparisons are required to insert the jth element into the correct position.

Therefore, the total number of comparisons used by the insertion sort to sort a list of n elements is

$$2 + 3 + \ldots + n = \frac{n(n+1)}{2} - 1,$$

Note that the insertion sort may use considerably fewer comparisons if the smaller elements started out at the end of the list. We conclude that the insertion sort has worst-case complexity $\Theta(n^2)$. ∎

In Examples 5 and 6 we showed that both the bubble sort and the insertion sort have worst-case time complexity $\Theta(n^2)$. However, the most efficient sorting algorithms can sort n items in $O(n \log n)$ time. From this point on, we will assume that sorting n items can be done in $O(n \log n)$ time.

You can run animations found on many different websites that simultaneously run different sorting algorithms on the same lists. Doing so will help you gain insights into the efficiency of different sorting algorithms. Among the sorting algorithms that you can find are the bubble sort, the insertion sort, the shell sort, the merge sort, and the quick sort. Some of these animations allow you to test the relative performance of these sorting algorithms on lists of randomly selected items, lists that are nearly sorted, and lists that are in reversed order.

3.3.3 Complexity of Matrix Multiplication

The definition of the product of two matrices can be expressed as an algorithm for computing the product of two matrices. Suppose that $\mathbf{C} = [c_{ij}]$ is the $m \times n$ matrix that is the product of the $m \times k$ matrix $\mathbf{A} = [a_{ij}]$ and the $k \times n$ matrix $\mathbf{B} = [b_{ij}]$. The algorithm based on the definition of the matrix product is expressed in Algorithm 1.

We can determine the complexity of this algorithm in terms of the number of additions and multiplications used.

ALGORITHM 1
Matrix Multiplication.

procedure *matrix multiplication*(**A, B**: matrices)
for $i := 1$ **to** m
  for $j := 1$ **to** n
    $c_{ij} := 0$
    for $q := 1$ **to** k
      $c_{ij} := c_{ij} + a_{iq}b_{qj}$
return C {**C**= $[c_{ij}]$ is the product of **A** and **B**}

EXAMPLE. 7

How many additions of integers and multiplications of integers are used by Algorithm 1 to multiply two $n \times n$ matrices with integer entries?

Solution: There are n^2 entries in the product of **A** and **B**. To find each entry requires a total of n multiplications and $n-1$ additions. Hence, a total of n^3 multiplications and $n^2(n-1)$ additions are used. ∎

Surprisingly, there are more efficient algorithms for matrix multiplication than that given in Algorithm 1. As Example 7 shows, multiplying two $n \times n$ matrices directly from the definition requires $O(n^3)$ multiplications and additions. Using other algorithms, two $n \times n$ matrices can be multiplied using $O(n^{\sqrt{7}})$ multiplications and additions.

We can also analyze the complexity of the algorithm we described in Chapter 2 for computing the Boolean product of two matrices, which we display as Algorithm 2.

The number of bit operations used to find the Boolean product of two $n \times n$ matrices can be easily determined.

EXAMPLE. 8

How many bit operations are used to find $\mathbf{A} \odot \mathbf{B}$, where **A** and **B** are $n \times n$ zero–one matrices?

Solution: There are n^2 entries in $\mathbf{A} \odot \mathbf{B}$. Using Algorithm 2, a total of n

*OR*s and n *AND*s are used to find an entry of $\mathbf{A} \odot \mathbf{B}$. Hence, $2n$ bit operations are used to find each entry. Therefore, $2n^3$ bit operations are required to compute $\mathbf{A} \odot \mathbf{B}$ using Algorithm 2. ∎

ALGORITHM 2
The Boolean Product of Zero–One Matrices.

procedure *Boolean product of Zero–One Matrices* (**A, B**: zero–one matrices)
for $i := 1$ **to** m
  for $j := 1$ **to** n
    $c_{ij} := 0$
    for $q := 1$ **to** k
      $c_{ij} := c_{ij} \vee (a_{iq} \wedge b_{qj})$
return C {**C**= $[c_{ij}]$ is the Boolean product of **A** and **B**}

MATRIX-CHAIN MULTIPLICATION

There is another important problem involving the complexity of the multiplication of matrices. How should the **matrix-chain** $\mathbf{A}_1\mathbf{A}_2 \dots \mathbf{A}_n$ be computed using the fewest multiplications of integers, where $\mathbf{A}_1\mathbf{A}_2 \dots \mathbf{A}_n$ are $m_1 \times m_2$, $m_2 \times m_3, \dots,$ $m_n \times m_{n+1}$ matrices, respectively, and each has integers as entries? Note that $m_1m_2m_3$ multiplications of integers are performed to multiply an $m_1 \times m_2$ matrix and an $m_2 \times m_3$ matrix using Algorithm 1.

Example 9 illustrates this problem.

EXAMPLE. 9

In which order should the matrices $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$ — where $\mathbf{A}_1$ is 30×20, $\mathbf{A}_2$ is 20×40, and $\mathbf{A}_3$ is 40×10, all with integer entries—be multiplied to use the least number of multiplications of integers?

Solution: There are two possible ways to compute $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_3$. These are $\mathbf{A}_1(\mathbf{A}_2\mathbf{A}_3)$ and $(\mathbf{A}_1\mathbf{A}_2)\mathbf{A}_3$.

If $\mathbf{A}_2$ and $\mathbf{A}_3$ are first multiplied, a total of $20 \cdot 40 \cdot 10 = 8000$ multiplications of integers are used to obtain the 20×10 matrix $\mathbf{A}_2\mathbf{A}_3$. Then, to multiply

$\mathbf{A}_1$ and $\mathbf{A}_2\mathbf{A}_3$ requires $30 \cdot 20 \cdot 10 = 6000$ multiplications. Hence, a total of

$$8000 + 6000 = 14,000$$

multiplications are used. On the other hand, if $\mathbf{A}_1$ and $\mathbf{A}_2$ are first multiplied, then $30 \cdot 20 \cdot 40 = 24,000$ multiplications are used to obtain the 30×40 matrix $\mathbf{A}_1\mathbf{A}_2$. Then, to multiply $\mathbf{A}_1\mathbf{A}_2$ and $\mathbf{A}_3$ requires $30 \cdot 40 \cdot 10 = 12,000$ multiplications. Hence, a total of

$$24,000 + 12,000 = 36,000$$

multiplications are used.

Clearly, the first method is more efficient. ∎

3.3.4 Algorithmic Paradigms

In Section 3.1 we introduced the basic notion of an algorithm. We provided examples of many different algorithms, including searching and sorting algorithms. We also introduced the concept of a greedy algorithm, giving examples of several problems that can be solved by greedy algorithms. Greedy algorithms provide an example of an **algorithmic paradigm**, that is, a general approach based on a particular concept that can be used to construct algorithms for solving a variety of problems.

Some of the algorithms we have already studied are based on an algorithmic paradigm known as brute force, which we will describe in this section. Algorithmic paradigms, studied later in this book, include divide-and-conquer algorithms, dynamic programming, backtracking, and probabilistic algorithms. There are many important algorithmic paradigms besides those described in this book.

BRUTE-FORCE ALGORITHMS

Brute force is an important, and basic, algorithmic paradigm. In a **brute-force algorithm**, a problem is solved in the most straightforward manner based on the statement of the problem and the definitions of terms. Brute-force algorithms are designed to solve problems without regard to the computing resources required. For example, in some brute-force algorithms the solution to a problem is found by examining

every possible solution, looking for the best possible. In general, brute-force algorithms are naive approaches for solving problems that do not take advantage of any special structure of the problem or clever ideas.

Note that Algorithm 1 in Section 3.1 for finding the maximum number in a sequence is a brute-force algorithm because it examines each of the n numbers in a sequence to find the maximum term. The algorithm for finding the sum of n numbers by adding one additional number at a time is also a brute-force algorithm, as is the algorithm for matrix multiplication based on its definition (Algorithm 1). The bubble, insertion, and selection sorts (described in Section 3.1 in Algorithms 4 and 5 are also considered to be brute-force algorithms; all three of these sorting algorithms are straightforward approaches much less efficient than other sorting algorithms such as the merge sort and the quick sort

Although brute-force algorithms are often inefficient, they are often quite useful. A bruteforce algorithm may be able to solve practical instances of problems, particularly when the input is not too large, even if it is impractical to use this algorithm for larger inputs. Furthermore, when designing new algorithms to solve a problem, the goal is often to find a new algorithm that is more efficient than a brute-force algorithm. One such problem of this type is described in Example 10.

EXAMPLE. 10

Construct a brute-force algorithm for finding the closest pair of points in a set of n points in the plane and provide a worst-case big-O estimate for the number of bit operations used by the algorithm.

Solution: Suppose that we are given as input the points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. Recall that the distance between (x_i, y_i) and (x_j, y_j) is $\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$. A brute-force algorithm can find the closest pair of these points by computing the distances between all pairs of the n points and determining the smallest distance. (We can make one small simplification to make the computation easier; we can compute the square of the distance between pairs of points to find the closest pair, rather than the distance between these points. We can do this because the square of the distance between a pair of points is smallest when the distance between these points is smallest.)

To estimate the number of operations used by the algorithm, first note that there are $n(n-1)/2$ pairs of points $((x_i, y_i), (x_j, y_j))$ that we loop through

(as the reader should verify). For each such pair we compute $(x_j - x_i)^2 + (y_j - y_i)^2$, compare it with the current value of *min*, and if it is smaller than min, replace the current value of min by this new value. It follows that this algorithm uses $\Theta(n^2)$ operations, in terms of arithmetic operations and comparisons. ∎

ALGORITHM 3
Brute-Force Algorithm for Closest Pair of Points.

procedure *closest-pair*$((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$: pairs of real numbers)
$min = \infty$
for $i := 2$ **to** n
  for $j := 1$ **to** $i - 1$
    if $(x_j - x_i)^2 + (y_j - y_i)^2 < min$ **then**
      $min := (x_j - x_i)^2 + (y_j - y_i)^2$
      closest pair $:= ((x_i, y_i), (x_j, y_j))$
return *closest pair*

3.3.5 Understanding the Complexity of Algorithms

Table 3.1 displays some common terminology used to describe the time complexity of algorithms. For example, an algorithm that finds the largest of the first 100 terms of a list of n elements by applying Algorithm 1 to the sequence of the first 100 terms, where n is an integer with $n \geq 100$, has **constant complexity** because it uses 99 comparisons no matter what n is (as the reader can verify). The linear search algorithm has **linear** (worst-case or average-case) **complexity** and the binary search algorithm has **logarithmic** (worst-case) complexity. Many important algorithms have $n \log n$, or **linearithmic** (worst-case) **complexity**, such as the merge sort. (The word *linearithmic* is a combination of the words *linear* and *logarithmic*.)

An algorithm has **polynomial complexity** if it has complexity $\Theta(n^b)$, where b is an integer with $b \geq 1$. For example, the bubble sort algorithm is a polynomial-time algorithm because it uses $\Theta(n^2)$ com-

Table 3.1: Commonly Used Terminology for the Complexity of Algorithms.

Complexity	**Terminology**
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	Linearithmic complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n)$, where$b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

parisons in the worst case. An algorithm has exponential complexity if it has time complexity $\Theta(b^n)$, where $b > 1$. The algorithm that determines whether a compound proposition in n variables is satisfiable by checking all possible assignments of truth variables is an algorithm with exponential complexity, because it uses $\Theta(2^n)$ operations. Finally, an algorithm has factorial complexity if it has $\Theta(n!)$ time complexity. The algorithm that finds all orders that a traveling salesperson could use to visit n cities has factorial complexity.

TRACTABILITY

A problem that is solvable using an algorithm with polynomial (or better) worst-case complexity is called **tractable**, because the expectation is that the algorithm will produce the solution to the problem for reasonably sized input in a relatively short time. However, if the polynomial in the big-Θ estimate has high degree (such as degree 100) or if the coefficients are extremely large, the algorithm may take an extremely long time to solve the problem. Consequently, that a problem can be solved using an algorithm with polynomial worst-case time complexity is no guarantee that the problem can be solved in a reasonable amount of time for even relatively small input values. Fortunately, in practice, the degree and coefficients of polynomials in such estimates are often small.

The situation is much worse for problems that cannot be solved

using an algorithm with worst-case polynomial time complexity. Such problems are called **intractable**. Usually, but not always, an extremely large amount of time is required to solve the problem for the worst cases of even small input values. In practice, however, there are situations where an algorithm with a certain worst-case time complexity may be able to solve a problem much more quickly for most cases than for its worst case. When we are willing to allow that some, perhaps small, number of cases may not be solved in a reasonable amount of time, the average-case time complexity is a better measure of how long an algorithm takes to solve a problem. Many problems important in industry are thought to be intractable but can be practically solved for essentially all sets of input that arise in daily life. Another way that intractable problems are handled when they arise in practical applications is that instead of looking for exact solutions of a problem, approximate solutions are sought. It may be the case that fast algorithms exist for finding such approximate solutions, perhaps even with a guarantee that they do not differ by very much from an exact solution.

Some problems even exist for which it can be shown that no algorithm exists for solving them. Such problems are called **unsolvable** (as opposed to **solvable** problems that can be solved using an algorithm). The first proof that there are unsolvable problems was provided by the great English mathematician and computer scientist Alan Turing when he showed that the halting problem is unsolvable. Recall that we proved that the halting problem is unsolvable in Section 3.1.

P VERSUS NP

The study of the complexity of algorithms goes far beyond what we can describe here. Note, however, that many solvable problems are believed to have the property that no algorithm with polynomial worst-case time complexity solves them, but that a solution, if known, can be checked in polynomial time. Problems for which a solution can be checked in polynomial time are said to belong to the **class NP** (tractable problems are said to belong to **class P**). The abbreviation NP stands for *nondeterministic polynomial* time. The satisfiability problem, discussed in Section 1.3, is an example of an NP problem—we can quickly verify that an assignment of truth values to the variables of a

compound proposition makes it true, but no polynomial time algorithm has been discovered for finding such an assignment of truth values.

There is also an important class of problems, called **NP-complete problems**, with the property that if any of these problems can be solved by a polynomial worst-case time algorithm, then all problems in the class NP can be solved by polynomial worst-case time algorithms. The satisfiability problem is also an example of an NP-complete problem. It is an NP problem and if a polynomial time algorithm for solving it were known, there would be polynomial time algorithms for all problems known to be in this class of problems (and there are many important problems in this class). This last statement follows from the fact that every problem in NP can be reduced in polynomial time to the satisfiability problem. Although more than 3000 NPcomplete problems are now known, the satisfiability problem was the first problem shown to be NP-complete. The theorem that asserts this is known as the **Cook-Levin theorem** after Stephen Cook and Leonid Levin, who independently proved it in the early 1970s.

The **P versus NP problem** asks whether NP, the class of problems for which it is possible to check solutions in polynomial time, equals P, the class of tractable problems. If $P \neq NP$, there would be some problems that cannot be solved in polynomial time, but whose solutions could be verified in polynomial time. The concept of NP-completeness is helpful in research aimed at solving the P versus NP problem, because NP-complete problems are the problems in NP considered most likely not to be in P, as every problem in NP can be reduced to an NP-complete problem in polynomial time. A large majority of theoretical computer scientists believe that $P \neq NP$, which would mean that no NP-complete problem can be solved in polynomial time. One reason for this belief is that despite extensive research, no one has succeeded in showing that $P = NP$. In particular, no one has been able to find an algorithm with worst-case polynomial time complexity that solves any NP-complete problem. The P versus NP problem is one of the most famous unsolved problems in the mathematical sciences (which include theoretical computer science). It is one of the seven famous Millennium Prize Problems, of which six remain unsolved. A prize of $1,000,000 is offered by the Clay Mathematics Institute for its solution.

Table 3.2: The Computer Time Used by Algorithms.

Size	**Bit Operations Used**					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	$3 \cdot 10^{-11}s$	$10^{-10}s$	$3 \cdot 10^{-10}s$	$10^{-9}s$	$10^{-8}s$	$3 \cdot 10^{-7}s$
10^2	$7 \cdot 10^{-11}s$	$10^{-9}s$	$7 \cdot 10^{-9}s$	$10^{-7}s$	$4 \cdot 10^{11}yr$	$*$
10^3	$1.0 \cdot 10^{-10}s$	$10^{-8}s$	$1 \cdot 10^{-7}s$	$10^{-5}s$	$*$	$*$
10^4	$1.3 \cdot 10^{-10}s$	$10^{-7}s$	$1 \cdot 10^{-6}s$	$10^{-3}s$	$*$	$*$
10^5	$1.7 \cdot 10^{-10}s$	$10^{-6}s$	$2 \cdot 10^{-5}s$	$0.1s$	$*$	$*$
10^6	$2 \cdot 10^{-10}s$	$10^{-5}s$	$2 \cdot 10^{-4}s$	$10.2s$	$*$	$*$

PRACTICAL CONSIDERATIONS

Note that a big-Θ estimate of the time complexity of an algorithm expresses how the time required to solve the problem increases as the input grows in size. In practice, the best estimate (that is, with the smallest reference function) that can be shown is used. However, big-Θ estimates of time complexity cannot be directly translated into the actual amount of computer time used. One reason is that a big-Θ estimate $f(n)$ is $\Theta(g(n))$, where $f(n)$ is the time complexity of an algorithm and $g(n)$ is a reference function, means that $C_1 g(n) \leq f(n) \leq C_2 g(n)$ when $n > k$, where C_1, C_2, and k are constants. So without knowing the constants C_1, C_2, and k in the inequality, this estimate cannot be used to determine a lower bound and an upper bound on the number of operations used in the worst case. As remarked before, the time required for an operation depends on the type of operation and the computer being used. Often, instead of a big-Θ estimate on the worst-case time complexity of an algorithm, we have only a big-O estimate. Note that a big-O estimate on the time complexity of an algorithm provides an upper, but not a lower, bound on the worst-case time required for the algorithm as a function of the input size. Nevertheless, for simplicity, we will often use big-O estimates when describing the time complexity of algorithms, with the understanding that big-Θ estimates would provide more information.

Table 3.2 displays the time needed to solve problems of various

sizes with an algorithm using the indicated number n of bit operations, assuming that each bit operation takes 10^{-11} seconds, a reasonable estimate of the time required for a bit operation using the fastest computers available in 2018. Times of more than 10^{100} years are indicated with an asterisk. In the future, these times will decrease as faster computers are developed. We can use the times shown in Table 3.2 to see whether it is reasonable to expect a solution to a problem of a specified size using an algorithm with known worst-case time complexity when we run this algorithm on a modern computer. Note that we cannot determine the exact time a computer uses to solve a problem with input of a particular size because of a myriad of issues involving computer hardware and the particular software implementation of the algorithm.

It is important to have a reasonable estimate for how long it will take a computer to solve a problem. For instance, if an algorithm requires approximately 10 hours, it may be worthwhile to spend the computer time (and money) required to solve this problem. But, if an algorithm requires approximately 10 billion years to solve a problem, it would be unreasonable to use resources to implement this algorithm. One of the most interesting phenomena of modern technology is the tremendous increase in the speed and memory space of computers. Another important factor that decreases the time needed to solve problems on computers is **parallel processing**, which is the technique of performing sequences of operations simultaneously.

Efficient algorithms, including most algorithms with polynomial time complexity, benefit most from significant technology improvements. However, these technology improvements offer little help in overcoming the complexity of algorithms of exponential or factorial time complexity. Because of the increased speed of computation, increases in computer memory, and the use of algorithms that take advantage of parallel processing, many problems that were considered impossible to solve five years ago are now routinely solved, and certainly five years from now this statement will still be true. This is even true when the algorithms used are intractable.

References

1. Rosen K. H. Discrete mathematics and its applications / K. H. Rosen. — McGraw-Hill Education, 2018. — 942 p.

2. Raymond Smullyan, What Is the Name of This Book?: The Riddle of Dracula and Other Logical Puzzles, Prentice- Hall, Englewood Cliffs, NJ, 1978.

3. Kurgalin S. The discrete math workbook: a companion manual using python / S. Kurgalin, S. Borzunov. — Springer Nature, 2020. — 507 p.

4. Epp S. S. Discrete mathematics with applications, metric edition / S. S. Epp. — Brooks/Cole, 2019. — 984 p.

5. Jenkyns T. Fundamentals of discrete math for computer science: a problem-solving primer / T. Jenkyns, B. Stephenson. — Springer International Publishing, 2018. — 512 p.

6. Rosen K. H. Student's solutions guide for discrete mathematics and its applications / K. H. Rosen. — McGraw-Hill Education, 2018. — 544 p.

Електронне навчальне видання

Дворниченко Аліна Василівна,
Лисенко Олександр Володимирович

ДИСКРЕТНА МАТЕМАТИКА ТА ТЕОРІЯ АЛГОРИТМІВ

Конспект лекцій

для студентів спеціальності *113 "Прикладна математика"*
денної форми навчання

У чотирьох частинах
Частина I

(Англійською мовою)

Відповідальний за випуск І. В. Коплик
Редактор А. В. Дворниченко
Комп'ютерне верстання А. В. Дворниченко

Формат 60x84/16. Ум. друк. арк. 15,98. Обл.-вид. арк. 17,08.

Видавець і виготовлювач
Сумський державний університет,
вул. Римського-Корсакова, 2, м. Суми, 40007
Свідоцтво суб'єкта видавничої справи ДК № 3062 від 17.12.2007.